Algorithmes de vision par ordinateur et d'apprentissage profond appliqués aux effets visuels et à la réalité augmentée

Houssam Halmaoui

Université Hassan I Settat

Habilitation universitaire

en

Informatique

Spécialité : Traitement des images et du signal

présentée et soutenue publiquement le 28 décembre 2022 par

Houssam HALMAOUI

Algorithmes de vision par ordinateur et d'apprentissage profond appliqués aux effets visuels et à la réalité augmentée

COMPOSITION DU JURY

Abderrahim Marzouk	Président	Professeur, FST, Settat
Mostafa Bellafkih	Rapporteur	Professeur, INPT, Rabat
Ahmed Mouhsen	Rapporteur	Professeur, FST, Settat
Nabil Laachfoubi	Rapporteur	Professeur, FST, Settat

Remerciements

Je tiens à exprimer ma sincère gratitude au Professeur Abdelkrim Haqiq, Directeur du Laboratoire IR2M dont je suis membre associé, pour ses orientations et ses conseils qui ont permis de rendre mon travail de recherche meilleur grâce à ses connaissances et son expérience.

Je remercie tous les membres du Jury qui ont généreusement donné de leur temps pour lire ce rapport et offrir de précieuses suggestions, notamment Messieurs les Professeurs Abderrahim Marzouk, Mostafa Bellafkih, Ahmed Mouhsen et Nabil Laachfoubi.

Merci enfin à tous les membres de ma famille et mes amis qui m'ont encouragé et soutenu.

Table des matières

Chapitre 1

Introduction

La vision par ordinateur est utilisée aujourd'hui dans l'ensemble du processus de production numérique des films [AFP07, Rad13, Ber14], de l'acquisition vidéo à la projection, en passant par toutes les étapes de post-production : effets visuels (VFX), infographie 3D (CG ou CGI), étalonnage des couleurs et montage. Les approches classiques du traitement de l'image telles que la segmentation, la correspondance stéréo, le flux optique, le suivi de mouvement, le Alpha Matting, la compression ou le Inpainting, ont été adaptées par les éditeurs de logiciels VFX de l'industrie cinématographique à des fins artistiques. D'une part, le principe de fonctionnement de ces méthodes n'est pas connu des artistes VFX qui consacrent plus de temps au processus créatif et à l'apprentissage des nouvelles fonctionnalités des logiciels. D'autre part, dans la communauté de recherche sur le traitement des images, il y a beaucoup moins d'applications VFX que la robotique, le médical, la vidéosurveillance ou la sécurité routière, et cela, parce que les besoins des artistes VFX sont peu connus des chercheurs. Les autres difficultés dans le domaine de la recherche des VFX est le temps de publication lent des nouvelles méthodes et la sélection de l'article le plus utile parmi des centaines.
Le but de ce rapport est de présenter les algorithmes fondamentaux de vision par ordinateur et d'apprentissage automatique utilisés dans le cinéma et plus particulièrement les VFX.

La plupart des films utilisent des VFX d'insertion, dans des scènes réelles, de personnages, d'environnements et d'actions irréels, sous forme de modèles 3D animés. Le résultat doit être très réaliste au point de tromper le spectateur. Cette technique d'insertion de modèles 3D, dite Matchmoving est très similaire à la réalité augmentée, à la différence qu'elle est appliquée en off-line (pas en temps réel) à cause de l'effet de réalisme recherché qui est couteux en temps de calcul et de travail. Toutefois, il existe aujourd'hui des technologies (souvent chers et encombrantes) qui permettent de faire le traitement en ligne, mais elles sont le plus souvent utilisées pour la pré-visualisation et non pas pour la production finale, à cause de l'aspect artistique impliqué dans la création des scènes (nécessité de retoucher les images).
Dans nos travaux de recherche, nous avons étudié les algorithmes impliqués dans les étapes de Matchmoving.

Le processus de Matchmoving se divise en plusieurs étapes :

- Tracking 3D/Caméra (suivi de mouvement) pour extraite des informations tridimensionnelles sur la scène afin d'effectuer l'insertion.
- Composition du modèle 3D avec la scène de manière cohérente avec la géométrie de la scène.
- Rendu photoréaliste du modèle 3D de manière cohérente avec l'éclairage de la scène.
- Retouche d'images.

Ces algorithmes de Tracking, de composition, d'imagerie 3D et de retouche d'images, que nous allons présenter dans ce rapport, ne se résument pas uniquement aux applications de Matchmoving et de réalité augmentée, mais sont les algorithmes fondamentaux des VFX.

Aujourd'hui, avec le succès de l'apprentissage profond (Deep Learning) dans le domaine des images, la plupart des algorithmes traditionnels citées ci-dessus, ont été surpassés par des modèles équivalents, fondés sur le Deep Learning.
Nous allons présenter dans ce rapport diverses méthodes traditionnelles et des modèles récents fondées sur l'apprentissage, ainsi que les différents algorithmes et applications que nous avons proposés.

Organisation du rapport

Dans le chapitre 2, nous allons présenter les méthodes traditionnelles de mise en correspondance des images permettant de réaliser le suivi de mouvement. Différents algorithmes de détection et de description des points d'intérêt seront présentés.

Ensuite, nous allons découvrir dans le chapitre 3, les algorithmes fondamentaux d'apprentissage automatique et profond dans le domaine des images, qui seront utilisés dans les chapitres suivants.

Le chapitre 4 présente les méthodes récentes de mise en correspondance, fondées sur des modèles d'apprentissage automatique et profond.

L'étape de rendu 3D sera abordée dans le chapitre 5. Les méthodes traditionnelles de rastérisation et de lancer de rayons seront étudiés et comparés aux méthodes récentes de rendu neuronal fondées sur l'apprentissage profond.

Les algorithmes de composition et de retouche d'images seront présentés dans le chapitre 6.

Enfin, dans le chapitre 7, nous allons présenter les méthodes VFX proposées de réalité augmentée, de stabilisation vidéo et de Matchmoving, ainsi que les modèles de mise en correspondance image, les datasets élaborées et les résultats des évaluations quantitatives et qualitatives de chaque méthode.

Chapitre 2

Algorithmes traditionnels de mise en correspondance

Sommaire

La mise en correspondance des images consiste à localiser avec précision les régions similaires d'images acquises de différents points de vue. La figure 2.1 montre un exemple de mise en correspondance.

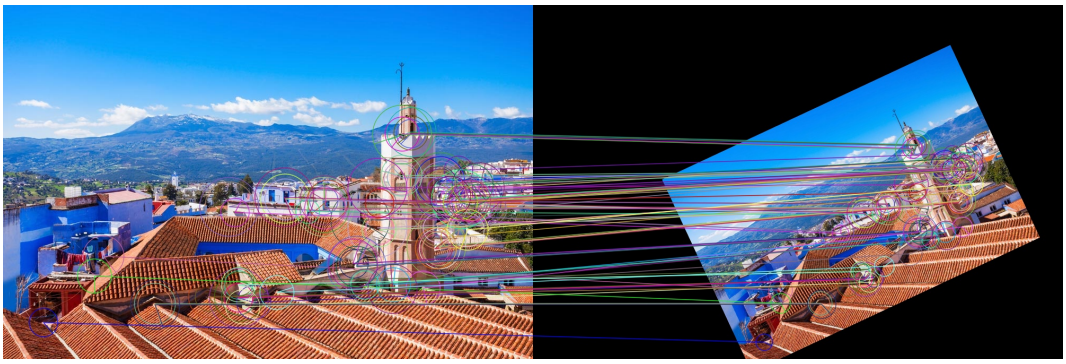

FIGURE 2.1 – Exemple de mise en correspondance.

En dehors des VFX, les applications sont nombreuses : la création de carte 3D d'une scène à partir d'un ensemble d'images acquises de différents points de vue par des méthodes Structure From Motion (SfM) [SF16] ou à partir d'une vidéo par des méthodes SLAM [MAT17], la géolocalisation sur une carte en comparant une image aux images d'une carte 3D préenregistrée, ou la création de panoramas en fusionnant plusieurs images dans une seule image [BL07].
La pipeline traditionnelle de la correspondance image consiste en trois étapes principales : détection, description et correspondance des points d'intérêt images. Les points d'intérêt (Features ou Keypoint en anglais) sont des régions uniques dans l'image. Il s'agit d'un rectangle W de l'image fiable pour la correspondance.

1 Détection des points d'intérêt

1.1 Harris

Le but de la détection est d'extraire les meilleures régions, telles que chacune ait un correspondant unique dans l'autre image.
Le problème de la détection peut être considéré comme une segmentation de l'image en trois régions : uniforme, lignes et coins. Les lignes et les régions uniformes ne sont pas fiables pour la correspondance puisqu'ils ne sont pas localement uniques, étant donné qu'ils ont dans leur voisinage plusieurs régions similaires. Les coins sont localement uniques et donc plus fiables.
Dans [Mor80], une fenêtre glissante est utilisée pour calculer la variation d'intensité d'un bloc W dans plusieurs directions. Pour une région uniforme, la variation est quasi-constante. Pour les lignes, la variation est très faible. Pour les contours, la variation est importante uniquement dans la direction

perpendiculaire au gradient. Pour un coin, la variation est importante dans toutes les directions. Par conséquent, un point d'intérêt est détecté si la variation est importante dans toutes les directions. La formulation mathématique du problème est la suivante :

$$E(u,v) = \sum_{x,y \in W} w(x,y)(I(x+u, y+v) - I(x,y))^2 \tag{2.1}$$

La fonction $E(u,v)$ mesure la différence entre les intensités des pixels dans une fenêtre W et les intensités dans W décalées d'un vecteur (u,v). La fonction binaire $w(x,y)$ prend la valeur de 1 à l'intérieur de la fenêtre et 0 sinon. Le vecteur (u,v) prend les valeurs $[(1,1);(1,0);(0,1);(-1,1)]$. La détection est effectuée en calculant d'abord la valeur minimale de $E(u,v)$ entre les quatre directions, ensuite, seuls les maxima locaux sont retenus et enfin un seuil est appliqué pour réduire le bruit afin de ne détecter que les points d'intérêt fiables. L'inconvénient de ce détecteur est qu'il est anisotrope, car nous considérons seulement les quatre directions pour mesurer la similarité.
Dans l'algorithme de Harris [HS+88], le problème est reformulé pour prendre en compte toutes les directions. Après un développement de Taylor de l'équation 2.1 autour de $(u,v) = (0,0)$ et une écriture matricielle, nous obtenons :

$$E(u,v) = \begin{bmatrix} u \\ v \end{bmatrix}^T \begin{bmatrix} \sum_{x,y} w(x,y) \left(\frac{\partial I(x,y)}{\partial x}\right)^2 & \sum_{x,y} w(x,y) \left(\frac{\partial I(x,y)}{\partial x}\frac{\partial I(x,y)}{\partial y}\right) \\ \sum_{x,y} w(x,y) \left(\frac{\partial I(x,y)}{\partial x}\frac{\partial I(x,y)}{\partial y}\right) & \sum_{x,y} w(x,y) \left(\frac{\partial I(x,y)}{\partial y}\right)^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.2}$$

La matrice dans le terme à droite, que nous désignons par H, est appelée matrice de Harris. Soit (α, β) les valeurs propres de H et (e_1, e_2) les vecteurs propres correspondants. Les courbes sur la figure 2.2 montrent la classification des différentes régions de l'image en fonction des valeurs propres.

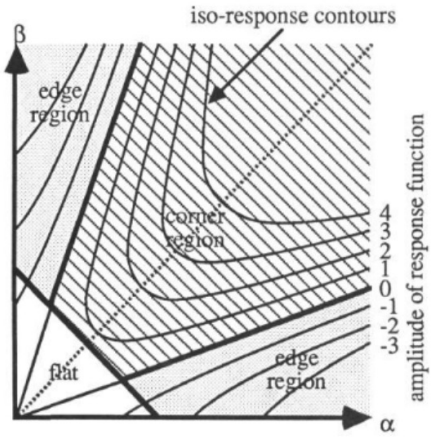

FIGURE 2.2 – Classification des régions en fonction des valeurs propres (α, β) de la matrice de Harris H.

Nous distinguons trois cas :

— Si les valeurs propres sont faibles, l'intensité de la fenêtre est uniforme. En effet, nous avons $\frac{\partial I}{\partial x} = \frac{\partial I}{\partial y} = 0$, donc $\alpha = \beta = 0$.

— Si β est faible et α important, la région correspond à un contour ou à une ligne. Le gradient $(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$ est perpendiculaire au contour, e_1 normal à la direction du contour et la valeur de α non négligeable. Alors que e_2 est parallèle à la direction du contour et la valeur de β est faible.

— Si les deux valeurs propres sont importantes, la fenêtre contient un coin ou un "blob" (régions connectés similaires).

Pour éviter de calculer les valeurs propres, nous utilisons une mesure de qualité C (représentée sur l'axe droit de la figure 2.2) appelée fonction de réponse :

$$C = det(H) - k.Tr(H)^2 \tag{2.3}$$

Lorsque la valeur de C est importante, la région correspond à un coin ou à un blob. La détection est alors effectuée par seuillage de C. La valeur du seuil dépend de l'application. Le paramètre k contrôle la sensibilité du détecteur : lorsque k est faible, plus de points d'intérêt sont détectés, mais la fiabilité du détecteur diminue à cause des régions à faible contraste. Nous prenons généralement $k = 0,04$.

Suppression des non-maxima locaux

La valeur C d'un point d'intérêt sera importante pour plusieurs blocs dans son voisinage. Par conséquent, plusieurs points d'intérêt correspondants à la même structure sont détectées autour de la même région. Nous ne retenons que le bloc ayant la plus grande mesure de qualité en calculant les maxima locaux de C dans un voisinage de taille $N \times N$. En effet, plus nous nous éloignons du centre du point d'intérêt, plus la mesure de qualité diminue. En pratique, nous cherchons à obtenir un grand nombre de détections et pour cela, nous pouvons réduire la valeur de N.
Le détecteur est covariant par rotation puisque le critère de détection dépend uniquement des valeurs propres et non des vecteurs propres. Il s'agit de covariance et non d'invariance puisque la position du point d'intérêt après rotation sera approximativement la même. Pour l'implémentation, le gradient est approximé par :

$$\frac{\partial I(x,y)}{\partial x} = I(x+1,y) - I(x-1,y) \tag{2.4}$$

Afin de réduire le bruit haute fréquence, le gradient est estimé par la convolution de l'image avec la dérivée d'une gaussienne (nous discuterons du choix de la fonction gaussienne dans la suite) :

$$\frac{\partial I(x,y)}{\partial x} = I(x,y) * \frac{\partial G(x,y,\sigma_D)}{\partial x} \tag{2.5}$$

où

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp(-\frac{1}{2\sigma^2}(x^2 + y^2)) \tag{2.6}$$

La fonction binaire w dans la matrice de Harris peut être remplacée par une fonction radiale symétrique pour donner plus de poids aux pixels les plus proches du centre (x_0, y_0) du point d'intérêt :

$$w(x, y) = G(x - x_0, y - y_0, \sigma_I) = \frac{1}{2\pi\sigma_I^2} \exp(-\frac{1}{2\sigma_I^2}((x - x_0)^2 + (y - y_0)^2)) \tag{2.7}$$

où

$$\sigma_I = a.\sigma_D \ , \ avec \quad a \in [1; 2] \tag{2.8}$$

La figure 2.3 montre quelques résultats obtenus par notre implémentation. L'inconvénient du détecteur de Harris est la taille fixe de la fenêtre W, qui limite la détection aux coins de même taille. Nous verrons dans la suite comment modifier l'algorithme de Harris en utilisant la théorie de l'espace d'échelle afin de permettre la détection multiéchelle de point d'intérêts.

FIGURE 2.3 – Comparaison de détecteurs. Première ligne : Harris et Harris Laplace. Deuxième ligne : Hessian Laplace et LoG

1.2 Détection multi-échelle : Harris Laplace

La méthode la plus simple pour effectuer la détection multiéchelles est de sous-échantillonner l'image. L'ensemble des images réduites est appelé résolution pyramidale (voir figure 2.4). Nous effectuons la détection sur chaque image

réduite en utilisant la même fenêtre W. Cela revient à détecter des points d'intérêt à plus grandes échelles lorsque la résolution diminue. Le problème de cette méthode est que le sous-échantillonnage introduit des structures non présentes dans l'image originale. Cependant, cette méthode est utile dans certains cas pour accélérer le traitement. Une alternative [Lin98] pour éviter d'introduire des artefacts est de convoluer l'image par une gaussienne afin de réduire la résolution intrinsèque de l'image, ce qui signifie que le nombre de pixels de l'image ne change pas, mais que les plus petits détails deviennent non détectables (voir figure 2.4). L'espace d'échelle (Scale Space) est l'ensemble des images construites par cette méthode. Une variante de cette méthode consiste à utiliser la diffusion [PM90] pour réduire la résolution sans rendre les bords de l'image flous.

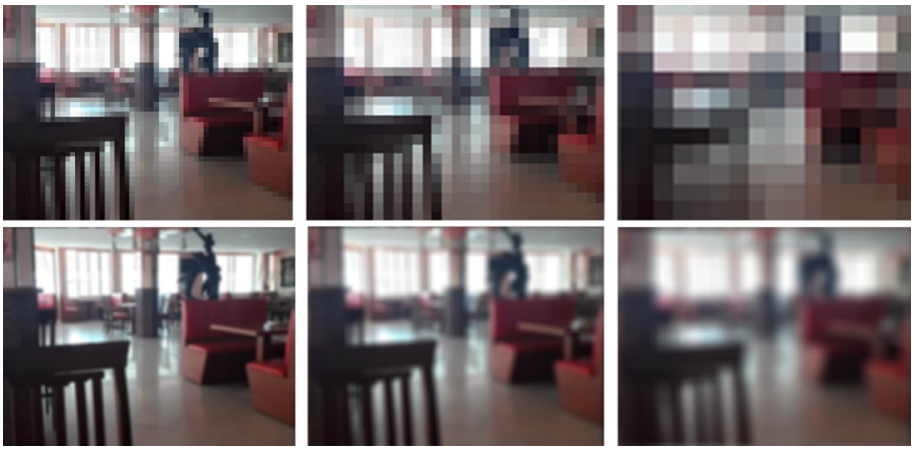

FIGURE 2.4 – Résolution pyramidale (première ligne) et espace d'échelle (deuxième ligne).

Nous commençons par construire l'ensemble de l'espace d'échelle en convoluant l'image par plusieurs gaussiennes :

$$L(x, y, \sigma_D) = G(x, y, \sigma_D) * I(x, y) \tag{2.9}$$

avec σ_D prenant des valeurs croissantes dans un intervalle généralement formé par une suite géométrique $\{\sigma_0, \sigma_0.b, \sigma_0.b^2, ...\}$. L'échelle diminue lorsque σ_D augmente. Nous prenons généralement $\sigma_0 = 1,5$ et $b \in [1,2; 1,4]$. La nouvelle matrice de Harris s'écrit comme suit :

$$H(x, y, \sigma_D, \sigma_I) = G(x, y, \sigma_I) * \begin{bmatrix} \left(\frac{\partial L(x,y,\sigma_D)}{\partial x}\right)^2 & \left(\frac{\partial L(x,y,\sigma_D)}{\partial x}\frac{\partial L(x,y,\sigma_D)}{\partial y}\right) \\ \left(\frac{\partial L(x,y,\sigma_D)}{\partial x}\frac{\partial L(x,y,\sigma_D)}{\partial y}\right) & \left(\frac{\partial L(x,y,\sigma_D)}{\partial y}\right)^2 \end{bmatrix} \tag{2.10}$$

où

$$\frac{\partial L(x, y, \sigma_D)}{\partial x} = \frac{\partial G(x, y, \sigma_D)}{\partial x} * I(x, y) \tag{2.11}$$

Normalisation de la matrice de Harris

La détection est effectuée sur chaque échelle comme précédemment. Cependant, avant de réaliser la détection, nous devons normaliser la matrice H afin d'avoir des matrices comparables entre les différentes échelles. En effet, pour la correspondance image, nous devons comparer les points d'intérêt détectés à différentes échelles en raison de la variation de la taille des objets entre les images. Nous considérons H et H' les matrices de Harris, respectivement, pour les images à haute et basse résolution. Si une fenêtre dans une image basse résolution est centrée autour de (x', y') à l'échelle (σ'_D, σ'_I), la même fenêtre en haute résolution est centrée autour de (x, y) à l'échelle (σ_D, σ_I). Nous avons [DSH00] :

$$H(x, y, \sigma_D, \sigma_I) = \frac{1}{b^2} H'(x', y', \sigma'_D, \sigma'_I) \tag{2.12}$$

Par conséquent, afin de normaliser la matrice de Harris, nous la multiplions par le terme de compensation :

$$b^2 = (\frac{\sigma_D}{\sigma'_D})^2 \tag{2.13}$$

Sélection d'échelle

Un coin ou un blob unique peut être détecté à différentes échelles. Nous devons choisir l'échelle qui représente le mieux la structure de l'élément. Pour cela, nous ne retenons que les maxima locaux sur la dimension d'échelle du laplacien normalisé :

$$NL(x, y, \sigma_D) = \left| \sigma_I^2 \left(\frac{\partial^2 G(x, y, \sigma_D)}{\partial x^2} + \frac{\partial^2 G(x, y, \sigma_D)}{\partial y^2} \right) * I(x, y) \right| \tag{2.14}$$

La figure 2.3 montre quelques résultats obtenus par notre implémentation de Harris Laplace. Nous remarquons que les points d'intérêt sont détectés à des échelles naturelles et que les régions similaires sont détectées à la même échelle (covariance d'échelle).
Notons également qu'il est possible de rendre le détecteur invariant aux transformations affines [MS04]. Cependant, ceci n'est pas nécessaire dans le cas de suivi vidéo puisque les variations géométriques entre deux images consécutives sont négligeables.

1.3 Hessian Laplace et LoG

Pour une meilleure détection des blobs, la matrice de Harris peut être remplacée par la matrice Hessienne normalisée par rapport à l'échelle :

$$S(x, y, \sigma_D) = \sigma_D^2 \begin{bmatrix} \frac{\partial^2 L(x, y, \sigma_D)}{\partial x^2} & \frac{\partial^2 L(x, y, \sigma_D)}{\partial x \partial y} \\ \frac{\partial^2 L(x, y, \sigma_D)}{\partial x \partial y} & \frac{\partial^2 L(x, y, \sigma_D)}{\partial y^2} \end{bmatrix} \tag{2.15}$$

Dans la méthode Hessian Laplace, au lieu d'utiliser la mesure de qualité C pour la détection, nous utilisons le déterminant de S, qui correspond au Hessian Laplace. Pour la sélection d'échelle, nous utilisons la trace de S qui correspond au Laplacien des Gaussiens LoG :

$$LoG(S) = Tr(S) \tag{2.16}$$

Cette méthode permet de mieux rejeter les contours.
Dans la méthode LoG, nous utilisons la trace de S à la fois pour la détection et la sélection d'échelle. Elle permet une meilleure détection des blobs, mais répond également aux contours.
La figure 2.3 montre quelques exemples de détection obtenus par nos implémentations de Hessian Laplace et LoG.
Le LoG peut être approximé par la différence de gaussiennes DoG pour accélérer le traitement comme dans SIFT [Low04] sans grande perte de précision. Notons qu'il existe plusieurs détecteurs rapides tels que FAST [RD06], BRIEF[CLSF10], ORB [RRKB11] et BRISK [LCS11]. L'inconvénient des détecteurs rapides est qu'ils sont moins précis.

1.4 FAST

L'algorithme FAST [RD06] permet la détection rapide des coins grâce à un seuillage sur un cercle de 16 pixels comme le montre la figure 2.5.

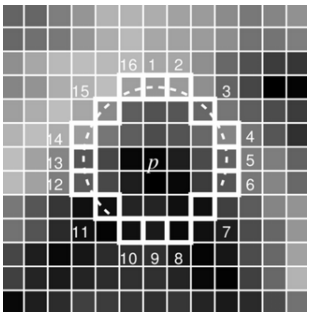

FIGURE 2.5 – Détection des points d'intérêt par l'algorithme FAST.

Nous considérons qu'une région est un point d'intérêt si au moins 12 pixels du cercle vérifient la condition :

$$I_x > I_p + t, \tag{2.17}$$

ou vérifient tous :

$$I_x < I_p - t, \tag{2.18}$$

avec I_x l'intensité d'un pixel du cercle, I_p l'intensité du pixel central et t un seuil.
Une présélection des points d'intérêt est effectuée sur les pixels 1, 5, 9 et 13. Il faut qu'au moins 3 parmi ces pixels vérifient la condition précédente. Si c'est le cas, nous appliquons le test sur les 16 pixels.

L'algorithme de suppression des non-maxima locaux peut être appliqué en considérons la mesure suivante :

$$V = \sum_{i=1}^{16} |I_p - I_x| \tag{2.19}$$

Si deux points d'intérêt ont des positions proches, nous éliminons celui avec la plus faible valeur de V.
Notons qu'il est possible d'entrainer un algorithme d'apprentissage automatique avec FAST. Pour cela, un arbre de décision est construit en appliquant FAST sur un ensemble d'image d'apprentissage, afin de générer l'arbre de décision que nous utilisons ensuite pour détecter les points d'intérêt sur de nouvelles images.
L'inconvénient de FAST est qu'il est très sensible au bruit.

1.5 Précision sous-pixel

Il est possible d'améliorer la détection des points d'intérêt avec une précision sous-pixel, au lieu d'utiliser l'algorithme de suppression des non-maxima locaux.

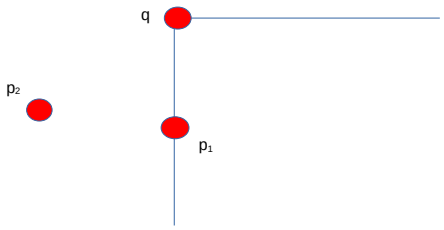

FIGURE 2.6 – Précision sous-pixel.

Nous commençons par détecter les composantes connectées dans la carte de la qualité de mesure C, où plusieurs points peuvent correspondre au même coin (voir figure 2.6).
Nous cherchons le point q qui minimise :

$$\epsilon_i = \nabla I_{p_i} \bullet (q - p_i) \tag{2.20}$$

En effet, si q est dans le coin et p_i dans le voisinage de q, nous avons deux cas :

- Soit p_i est dans une ligne : le gradient ∇I_{p_i} est perpendiculaire à $q - p_i$ et donc $\epsilon_i = 0$.
- Soit p_i est dans une région uniforme : le gradient $\nabla I_{p_i} = 0$ et donc $\epsilon_i = 0$.

En pratique, ϵ_i ne sera jamais nul à cause du bruit, nous procédons donc par minimisation.

Nous cherchons donc dans le voisinage V_q de q à minimiser la fonction E obtenue après développement de l'équation 2.20 :

$$E = \sum_{i \in V_q} (\nabla I_{p_i} . \nabla I_{p_i}^T).q - \sum_{i \in V_q} (\nabla I_{p_i} . \nabla I_{p_i}^T).p_i \tag{2.21}$$

Nous notons :

$$G = \sum_{i \in V_q} (\nabla I_{p_i} . \nabla I_{p_i}^T) \tag{2.22}$$

$$b = \sum_{i \in V_q} (\nabla I_{p_i} . \nabla I_{p_i}^T).p_i \tag{2.23}$$

Si $E = 0$, nous obtenons :

$$q = G^{-1}.b \tag{2.24}$$

L'algorithme est le suivant :
1) Nous commençons par initialiser q avec le centre de la région connectée.
2) Nous calculons dans le voisinage de q les valeurs de G et b.
3) Nous calculons le nouveau $q = G^{-1}.b$ (ou nous minimisons E par moindres carrés).
4) Nous calculons l'erreur $q - q_{precedent}$. Si l'erreur est inférieure à un seuil, nous arrêtons le calcul, sinon nous reprenons à l'étape 2.

1.6 MSER

MSER ((Maximally Stable Extremal Region ou MSER) [MCUP04] est un détecteur de blobs covariant par transformation affine. Le principe est de faire varier un seuil i et de trouver les régions Ω tel que :

$$\forall p \in \Omega, \quad \forall q \in V_\Omega \quad I(p) < I(q) \quad ou \quad I(p) < I(q) \tag{2.25}$$

avec V_Ω le voisinage de Ω.
En d'autres termes, les pixels connectés à l'intérieur des régions détectées sont tous plus clairs ou plus sombres que les pixels des contours externes. Plus le seuil augmente, plus la taille des régions augmente.
La détection des points d'intérêt est effectuée en sélectionnant les régions stables pour la mise en correspondance, qui correspondent aux régions obtenues avec le seuil i qui minimise la mesure :

$$q(i) = \frac{Card(\Omega_{i+1} - \Omega_{i-1})}{Card(\Omega_i)} \tag{2.26}$$

avec $Card(\Omega_i)$ est l'aire de la région Ω_i et $\Omega_i \subset \Omega_{i+1}$.
L'ellipse entourant la région détectée est utilisée comme patch du point d'intérêt.
Les régions stables ont la propriété de rester connectées après transformation affine et sont donc fiables pour la mise en correspondance.

2 Description des points d'intérêt

2.1 SIFT

SIFT est l'algorithme le plus populaire de détection et de description des points d'intérêt.

Détection des échelles

La détection est effectuée par l'opérateur LoG (vu précédemment) qui est approximé par différence des gaussiennes (DoG) en introduisant la notion d'octave illustrée sur la figure 2.7.

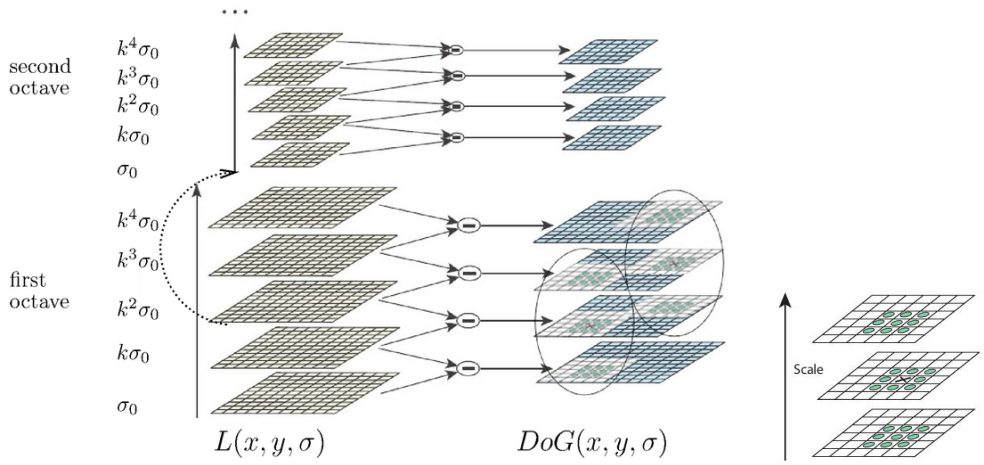

FIGURE 2.7 – Détection des extremum locaux des échelles.

Des DoG avec différents σ de l'espace d'échelle $(\sigma, k.\sigma, k^2.\sigma, k^3.\sigma, , k^4.\sigma)$ sont appliqués pour détecter des coins et des blobs de différentes tailles (faible σ pour les coins de petite taille et σ important pour les coins de grande taille).
Les octaves permettent de passer à l'échelle suivante sans modifier les paramètres $(\sigma, k.\sigma, k^2.\sigma, k^3.\sigma, , k^4.\sigma)$, mais en réduisant la résolution de l'image, ce qui permet d'accélérer les calculs.
L'échelle σ qui forme un extremum local avec les échelles précédente et suivante, ainsi que spatialement (pixels voisins) correspond à un coin potentiel.

Localisation des points d'intérêt

Le problème avec le résultat de l'étape précédente est le manque de précision dans la localisation des points d'intérêt. En effet, à cause de l'utilisation d'un espace discret d'échelles $(\sigma, k.\sigma, k^2.\sigma, k^3.\sigma, , k^4.\sigma)$, la position des extremums n'est pas très exacte comme le montre la figure 2.8 : les vrais extremums sont localisés par un développement de Taylor.

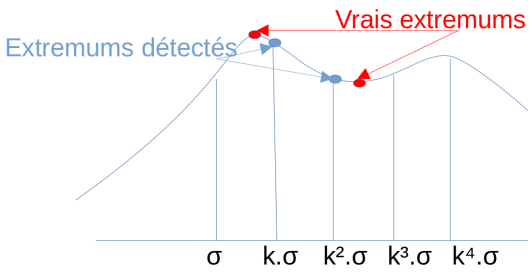

FIGURE 2.8 – Amélioration de la précision de localisation des extremums locaux.

Ensuite, un seuil est appliqué aux matrices DoG pour éliminer les faibles contrastes.
Un autre problème de l'opérateur DoG est qu'il détecte beaucoup de lignes. Or, nous avons vu que pour les lignes, les valeurs propres de la matrice Hessienne vérifient $\lambda_1 \gg \lambda_2$. Nous calculons donc le ratio $\frac{\lambda_1}{\lambda_2}$. Si le ratio est supérieur à un seuil r, alors il s'agit d'une ligne. Nous pouvons aussi utiliser la matrice hessienne avec la condition :

$$\frac{Tr(H)}{Det(H)} < \frac{(r+1)^2}{r} \tag{2.27}$$

Notons que SIFT est un bon détecteur de blobs, alors que Harris est un bon détecteur de coins.

Descripteur et orientation des points d'intérêt

Une fois que nous avons obtenu la position et l'échelle du point d'intérêt, nous calculons un descripteur $Desc$. Soit f et f' deux points d'intérêt dans deux images différentes tel que $f' = T(f)$ avec T une transformation géométrique et photométrique (changement de point de vue, échelle, rotation ou variation de luminance). Un bon descripteur D doit vérifier $Desc(f) = Desc(f')$. Le descripteur le plus simple est un vecteur formé par l'ensemble des valeurs d'intensité des pixels de l'élément détecté. Pour effectuer la correspondance, nous comparons les intensités des points d'intérêt. De nombreux algorithmes de correspondance utilisent cette méthode en raison de sa rapidité et de ses résultats satisfaisants lorsque les variations géométriques et photométriques sont faibles, ce qui est souvent le cas dans les séquences vidéo. En général, les variations peuvent être importantes lorsque la caméra se déplace rapidement ou à cause des variations d'éclairage, et nous devons utiliser un descripteur invariant.
Nous commençons par choisir les pixels pertinents qui contribueront au descripteur. Nous considérons une région de support, plus grande que la taille (échelle) du point d'intérêt détecté, ayant une taille égale à 3 ou 6 fois la taille du point d'intérêt (voir figure 2.9). Ensuite, afin de pouvoir comparer les

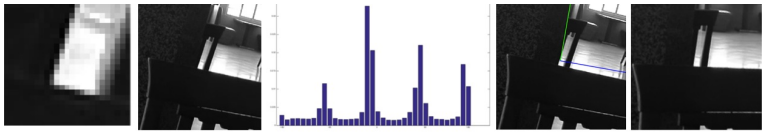

FIGURE 2.9 – De gauche à droite : point d'intérêt détecté, région de support, histogramme d'orientation, détection du gradient dominant, rotation de la région de support.

descripteurs de deux points d'intérêt ayant des orientations différentes, nous détectons dans chaque région de support l'orientation du gradient dominant, et nous changeons l'orientation de sorte que le gradient dominant soit horizontal (voir Figure 2.9). Pour détecter l'orientation du gradient dominant, nous commençons par calculer la magnitude M et l'orientation θ du gradient pour chaque pixel de la région de support D :

$$M(x,y) = \sqrt{(D(x+1,y) - D(x-1,y))^2 + (D(x,y+1) - D(x,y-1))^2} \tag{2.28}$$

$$\theta = \arctan(\frac{D(x,y+1) - D(x,y-1)}{D(x+1,y) - D(x-1,y))}) \tag{2.29}$$

L'histogramme des orientations est calculé sur une échelle de 36 bins séparés par un pas de 10. Chaque pixel est incrémenté de la magnitude M pondérée par une gaussienne $M(x,y).G(x,y,\sigma)$ pour donner plus de poids aux pixels du centre. Avec σ égal à $1,5$ fois l'échelle du point d'intérêt.
L'orientation est déterminée en détectant le pic maximal de l'histogramme, comme le montre la figure 2.9. Si l'histogramme contient plusieurs pics, nous pouvons retenir plusieurs points d'intérêt à la même position et à la même échelle (mais avec différentes orientations) et ne retenir que le meilleur candidat lors de la mise en correspondance. Une autre solution est de détecter tous les pics supérieurs à 80% du pic maximal.
Si nous utilisons un descripteur d'intensités, nous pouvons réorienter le patch de façon à avoir un gradient dominant horizontal, ensuite nous échantillonnons à nouveau la fenêtre en taille 41×41 ou 81×81 pour avoir des descripteurs de la même taille quelle que soit l'échelle de détection. Afin de compenser les variations de luminance, l'intensité du patch final D_f est normalisée avec sa moyenne μ et sa variance s : $D'_f = \frac{(D_f - \mu)}{s}$. Nous obtenons ainsi un point d'intérêt invariant à l'échelle, à la translation et à l'orientation.
Le descripteur SIFT est calculé un peu différemment. Une région 16×16 est considérée autour du point d'intérêt (voir la figure 2.10). Cette région est divisée en sous-régions 4×4. Pour chaque sous-région, nous créons l'histogramme d'orientation sur 8 bins. Nous obtenons ainsi un vecteur descripteur de 128 valeurs (8×16). Notons qu'il existe d'autres méthodes, utilisant des principes similaires à SIFT pour le calcul du descripteur, mais avec des formes de patch différentes comme le montre la figure 2.11.

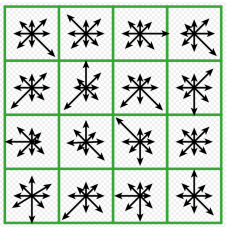

FIGURE 2.10 – Calcul du descripteur SIFT sur une région 16×16 autour du point d'intérêt.

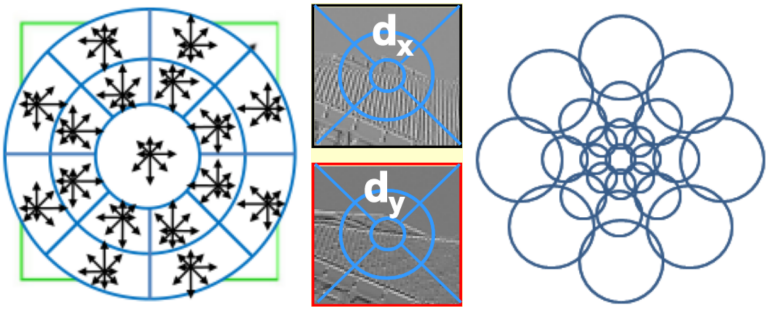

FIGURE 2.11 – Descripteurs similaires à SIFT avec diverses formes de régions. De gauche à droite : GLOH [MS05], CHOG [CTC+12] et DAISY [TLF09].

Mise en correspondance des images

Pour effectuer la correspondance entre deux images, nous comparons les valeurs a_i du descripteur A de chaque point d'intérêt, et pour chaque échelle, dans la première image, aux valeurs b_i du descripteur B de chaque point d'intérêt, et pour chaque échelle, dans l'image suivante. Pour que la correspondance soit exhaustive, un point d'intérêt candidat ne doit être retenu qu'une fois tous les points d'intérêt à toutes les échelles ont été comparés au point d'intérêt de référence. Pour la comparaison, nous utilisons la mesure de somme des différences au carré ou SSD (Sum Squared Difference) et nous retenons le point d'intérêt candidat qui minimise cette distance :

$$ssd(A, B) = \sum_i (a_i - b_i)^2 \tag{2.30}$$

D'autres mesures telles que la distance euclidienne ou la corrélation peuvent être utilisées.
Nous éliminons les candidats dont le SSD dépasse un seuil qui varie généralement entre $0,1$ et $0,8$ selon l'application. Nous limitons la zone de recherche dans la seconde image au voisinage de l'élément de référence dans la première image. La taille du voisinage dépend du déplacement de la caméra entre les deux images.
Nous considérons aussi le second plus proche point d'intérêt ayant comme SSD

ssd_2. Nous calculons le ratio :

$$r_{ssd} = \frac{ssd}{ssd_2} \tag{2.31}$$

Si $r_{ssd} > 0,8$, nous éliminons le point d'intérêt pour éviter des erreurs de mise en correspondance (deux points d'intérêts très semblables ou du bruit). Cette étape est cruciale, car elle permet d'éliminer beaucoup de faux correspondants et peu de vrais correspondants.
Notons que le brevet de SIFT n'est plus valable depuis l'année 2020 et qu'une implémentation est disponible dans la librairie OpenCV.

2.2 ASIFT

ASIFT est une variante SIFT [MY09] permettant de rendre le descripteur invariant par transformation affine. Le principe comme le montre la figure 2.12 est d'appliquer pour chaque patch différentes transformations affines et de le comparer à tous les autres lors de la mise en correspondance. L'inconvénient est que le calcul est très couteux. Notons qu'il existe aussi des variantes affines des détecteurs de Harris et Hessian Laplace [MS04], mais le détecteur ASIFT donne de meilleurs résultats.

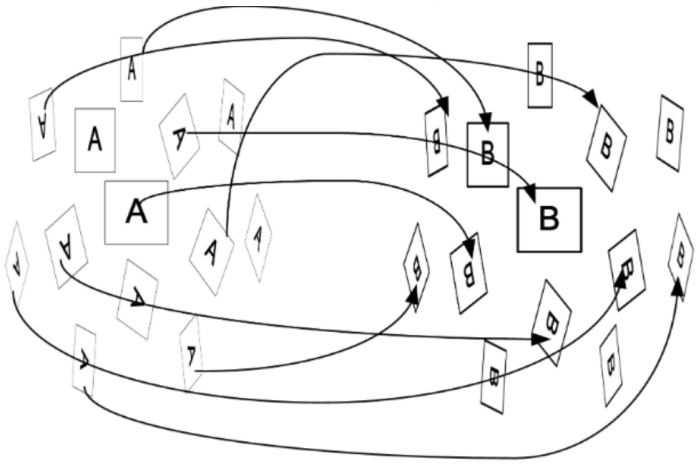

FIGURE 2.12 – Simulation de différentes transformations affines dans ASIFT.

2.3 SURF

L'algorithme SURF [BTVG06] permet d'accélérer SIFT en approximant l'opérateur LoG par des filtres.
D'abord, les dérivées secondes d'une gaussienne $\frac{\partial^2 g(\sigma)}{\partial x^2}$, $\frac{\partial^2 g(\sigma)}{\partial y^2}$ et $\frac{\partial^2 g(\sigma)}{\partial x \partial y}$ sont approximées par des filtres simples comme le montre la figure 2.13.

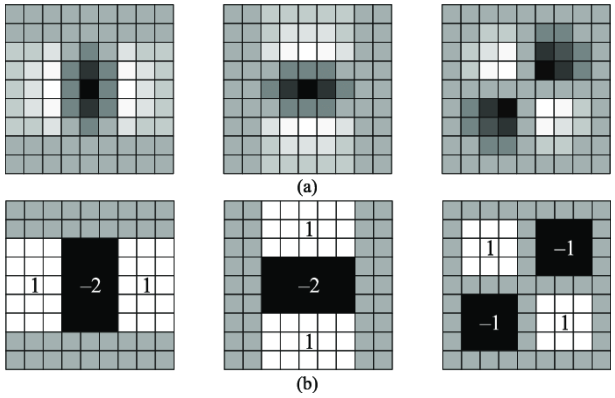

FIGURE 2.13 – Approximation des dérivées secondes d'une gaussienne (première ligne) par des filtres simples (deuxième ligne).

Il suffit ensuite de remplacer la matrice hessienne S de l'équation 2.15 par son approximation. La matrice hessienne devient :

$$S_{approx}(x, y, \sigma_D) = \sigma_D^2 \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{2.32}$$

avec :

$$D_{xx} = \frac{\partial^2 G(x, y, \sigma_D)}{\partial x} * I(x, y), \tag{2.33}$$

tel que $\dfrac{\partial^2 G(x, y, \sigma_D)}{\partial x}$ est approximé par un filtre.
La détection des points d'intérêt et de l'échelle est effectuée comme précédemment par Hessian Laplace (déterminant de S).
Notons que l'algorithme image intégral (summed area table) est utilisé pour le calcul rapide des sommes dans l'opération de filtrage.
La théorie de l'espace d'échelle par octave est utilisée, sauf qu'au lieu de diminuer la taille des images, la taille des filtres est augmentée comme le montre la figure 2.14. Le niveau le plus bas correspond à un filtre de taille 9×9 qui équivaut à une gaussienne avec un $\sigma = 1,2$.

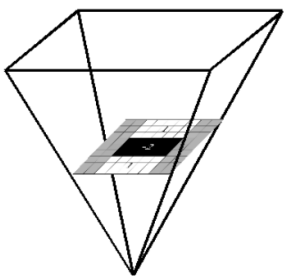

FIGURE 2.14 – Espace d'échelle créé en augmentant la taille des filtres.

Dans [BTVG06], la formule du déterminant de l'hessienne utilisée pour la détection d'échelle est approximée par :

$$det(S_{approx}) = D_{xx}D_{yy} - 0,9D_{xy} \tag{2.34}$$

L'orientation des points d'intérêt est détectée avec les ondelettes de Haar (voir la figure 2.15) en calculant la somme des réponses des ondelettes horizontales et verticales avec un pas de $\frac{\pi}{3}$ sur un voisinage $6.\sigma$ autour du point d'intérêt, avec σ l'échelle détectée. L'orientation correspond à la plus grande somme. La somme intégrale est utilisée encore une fois pour accélérer les calculs.

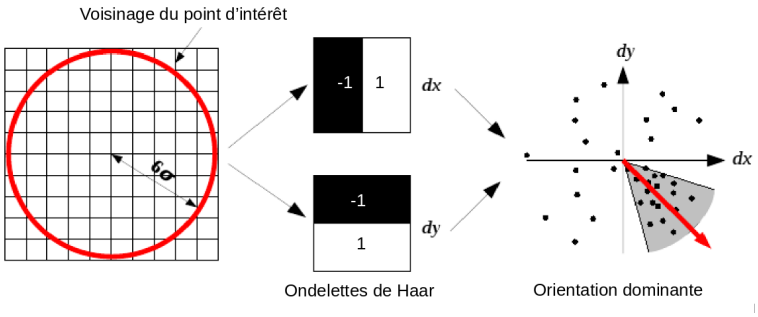

FIGURE 2.15 – Détection de l'orientation du point d'intérêt avec les ondelettes de Haar.

Pour le calcul du descripteur, nous considérons un voisinage de taille 20σ autour du point d'intérêt et que nous divisons en 16 régions (4×4), comme le montre la figure 2.16. Chaque région est ensuite divisée en 25 sous-régions (5×5). Les ondelettes de Haar sont appliquées sur chaque région afin de calculer un vecteur de taille 4 : $[\sum I_i, \sum |I_i|, \sum I_j, \sum |I_j|]$. Nous obtenons à la fin un vecteur de taille 64 pour les 16 régions qui correspond à notre descripteur SURF.

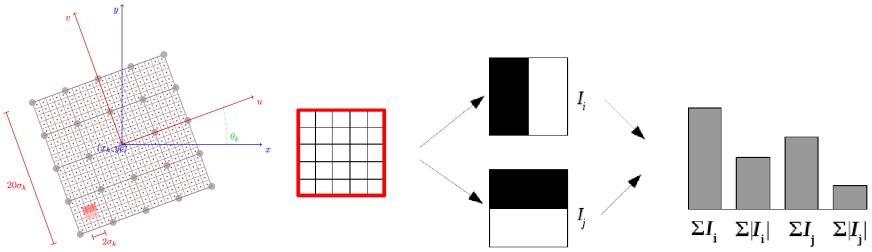

FIGURE 2.16 – Calcul du descripteur SURF.

Notons qu'il existe une version SURF étendu à 128 dimensions tel que les sommes sont calculées pour $dx > 0$ et $dx < 0$ séparément et de même pour dy. Notons enfin que le signe du Laplacien permet de distinguer entre les blobs noirs sur fond blanc et l'inverse. Cela est utile pour n'effectuer la mise en correspondance que pour des blobs du même type.
Enfin, notons que le brevet SURF est toujours valable, mais une implémentation est disponible dans OpenCV pour des fins de recherche.

2.4 BRIEF

L'inconvénient des descripteurs SIFT et SURF est qu'ils sont stockés avec des valeurs en float sur 32 bit. Donc pour chaque point d'intérêt en dimension 128, il nous faut 512 octets.
Il existe des solutions consistant à appliquer une PCA (analyse en composante principale) ou LDA (analyse discriminante linéaire) afin de réduire la dimensionnalité du vecteur, ou d'appliquer un algorithme de hashing LSH (Locality sensitive hashing) pour convertir le vecteur en binaire. Toutefois, nous serons toujours obligés de calculer d'abord le descripteur en float.
Le principe du descripteur BRIEF [CLSF10] consiste à sélectionner n paires de points (p, q) dans un patch de l'image (lissée). La sélection est effectuée selon des distributions uniformes gaussiennes. Si $I_p > I_q$ alors l'élément du descripteur prend la valeur de 1, sinon il prend la valeur de 0. Nous appliquons cela à toutes les n paires pour obtenir un descripteur de dimension n.
La mise en correspondance peut être effectuée dans le cas des descripteurs binaires à l'aide de la distance de Hamming :

$$dist_{i,j} = \#\{i \neq j\} \tag{2.35}$$

Notons que contrairement à SIFT et SURF, l'algorithme BRIEF ne permet que la description des points d'intérêt. La détection peut être effectuée avec un autre détecteur (Harris, SIFT, etc.), mais l'auteur recommande l'algorithme CenSurE [AKB08].

2.5 ORB

L'algorithme ORB [RRKB11] est une combinaison du détecteur FAST et du descripteur BRIEF avec quelques modifications pour permettre une détection rapide, multi-échelle et invariante par rotation.
D'abord, la détection est effectuée avec FAST. Ensuite, les N meilleurs points sont sélectionnés en utilisant la mesure de qualité de Harris.
La détection est effectuée à plusieurs échelles à l'aide d'une pyramide d'images à différentes résolutions.
L'orientation est calculée comme la direction entre le centre du coin et le centroid du patch calculé à partir du moment du premier ordre :

$$centroid = (\frac{m_{11}}{m_{00}}, \frac{m_{01}}{m_{00}}), \tag{2.36}$$

avec

$$m_{pq} = \sum_{x,y} x^p y^p I(x, y) \tag{2.37}$$

Ainsi, l'orientation du point d'intérêt est :

$$\theta = atan2(m_{01}, m_{10}) \tag{2.38}$$

Le descripteur BRIEF est modifié en rBRIEF (rotation-aware BRIEF) pour prendre en compte l'orientation du point d'intérêt. La matrice de rotation correspondant à l'angle θ est appliquée au patch avant de calculer le descripteur BRIEF afin de le rendre invariant par rotation.
D'après l'auteur, les performances de ORB sont assez proches de SIFT, meilleures que SURF et plus rapide que les deux.

2.6 LUCID

Il s'agit d'un descripteur rapide et très simple [ZCKB12], dont le principe est de diviser le patch en trois canaux RGB, de les vectoriser, de les trier et de permuter les indices comme le montre la figure 2.17. L'implémentation sous Matlab peut être réalisée en trois lignes de code. Le descripteur n'est plus

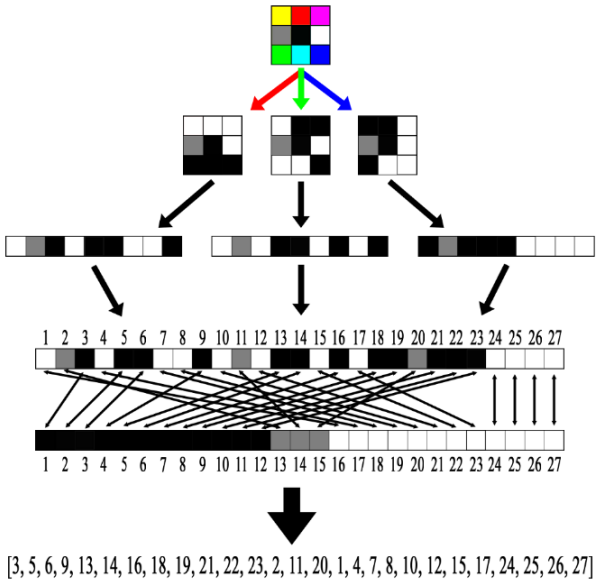

FIGURE 2.17 – Descripteur LUCID et implémentation.

d'actualité aujourd'hui mais il donne de bons résultats dans des applications réelles.

3 Conclusion

Dans ce chapitre, nous avons découvert les algorithmes traditionnels les plus communs de détection, de description et de mise en correspondance des points

d'intérêt. Le choix de la méthode va dépendre des applications en fonction de la précision et du temps de calcul nécessaires. Plusieurs de ces méthodes sont utilisées encore aujourd'hui, à cause de leurs performances, malgré l'apparition de nouvelles méthodes fondées sur l'apprentissage automatique, que nous allons présenter dans les chapitres suivants.

Chapitre 3

Apprentissage automatique et profond

Sommaire

L'objectif de ce chapitre est de découvrir les notions fondamentales d'apprentissage automatique, nécessaires pour la compréhension des méthodes des chapitres suivants de mise en correspondance des images (chapitres 4 et 7) et de rendu neuronal d'images de synthèse (chapitre 5).
Notons que la vision par ordinateur est le domaine qui a le plus tiré profit du développement de l'apprentissage profond dans des applications telles que la classification [KSH12], la détection [GDDM15] et la segmentation [LSD15] d'objets dans des images. Cette réussite est due à l'amélioration de la puissance de calcul qui a permis le déploiement de modèles de plus en plus complexes.

L'apprentissage automatique permet à un ordinateur d'apprendre à effectuer une tâche à partir de l'observation d'un certain nombre de données, sans aucune programmation explicite des règles à suivre. L'image 3.1 illustre le principe simplifié dans le cas d'un problème de classification d'images de chaises. Un modèle (que nous détaillerons dans la suite) est entrainé en observant dans un premier temps un ensemble de données images étiquetées avec des labels "chaise" et "pas une chaise". La labellisation est effectuée manuellement au préalable. Une fois l'apprentissage terminé, le modèle peut prédire la classe d'une nouvelle image (non observée durant l'apprentissage).

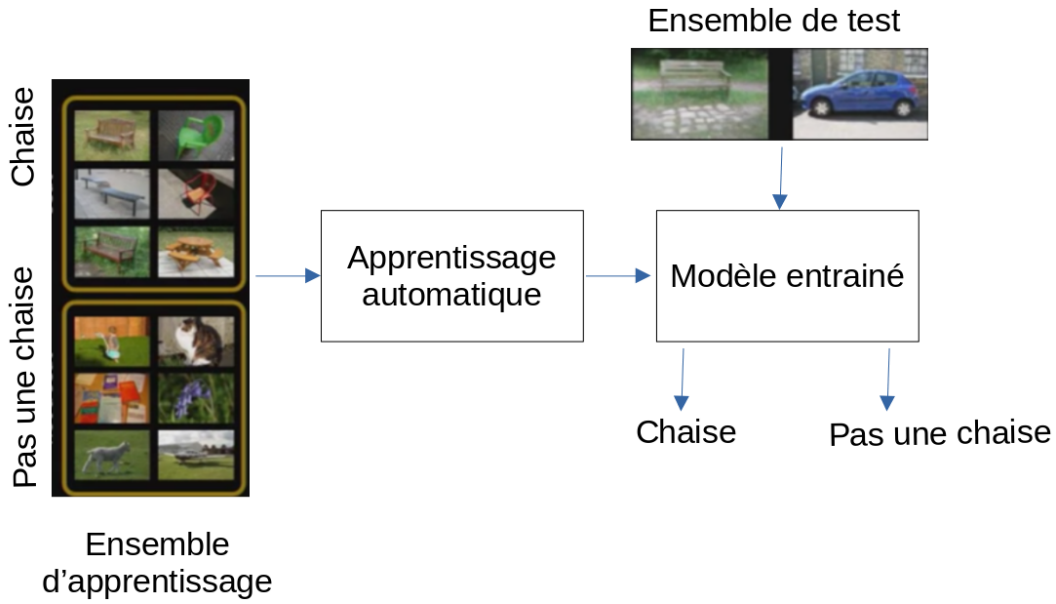

FIGURE 3.1 – Principe simplifié de l'apprentissage automatique

Notons que plus nous utilisons de données, plus le modèle devient performant. Par ailleurs, l'une des principales raisons du succès de l'apprentissage automatique en vision est l'amélioration des capacités de calcul des ordinateurs qui a permis de traiter plus de données, en plus des modèles plus complexes.
Dans la suite, nous allons détailler différentes architectures de modèles d'apprentissage, les algorithmes d'optimisation, l'utilisation des dataset pour

l'apprentissage et les méthodes et les métriques d'évaluation.

1 Régression logistique

1.1 Fonction de prédiction et frontière de décision

La régression logistique est le modèle le plus simple pour classifier des données. Nous considérons dans un premier temps une classification binaire d'une variable d'entrée $\mathbf{x}$ en classe $y = \{0, 1\}$.
$\mathbf{x} = [1\ x_1\ ...\ x_N]$ est une représentation de la donnée à classifier dans un espace de caractéristiques (features) à N dimension (nous considérons $x_0 = 1$).
Nous cherchons les paramètres $\theta = [\theta_0\ \theta_1\ ...\ \theta_N]$ de la fonction de prédiction h suivante :

$$h_\theta(\mathbf{x}) = g(\theta^T\mathbf{x}) = \frac{1}{1 + \exp(-\theta^T\mathbf{x})}, \tag{3.1}$$

g est la fonction sigmoïde (ou logistique) illustrée sur la figure 3.2 pour une variable unidimensionnelle.

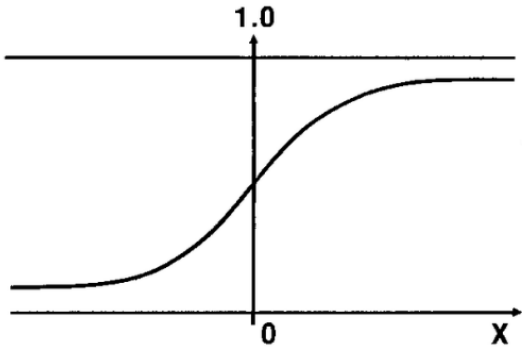

FIGURE 3.2 – Fonction sigmoïde.

Nous avons $g(-\infty) = 0$ et $g(+\infty) = 1$. Pour la prédiction de la classe y_{pred} des données, nous fixons un seuil pour $h(x)$ (généralement de $0,5$) :

$$y_{pred} = \begin{cases} 1 & , si \quad h(\mathbf{x}) \geq 0,5 \\ 0 & , sinon \end{cases} \tag{3.2}$$

La figure 3.3 illustre la frontière de décision trouvée pour une variable $\mathbf{x}$ à 2 dimensions. Nous avons :

$$h(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \tag{3.3}$$

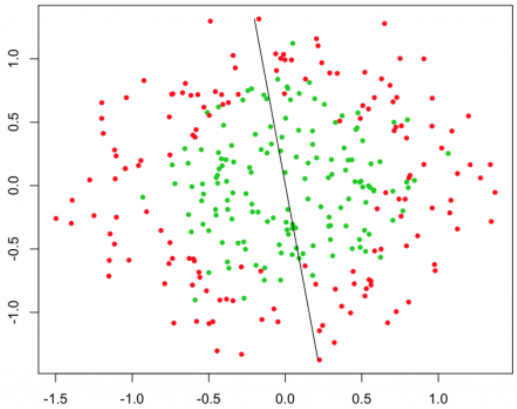

FIGURE 3.3 – Frontière de décision linéaire.

Pour obtenir une frontière non linéaire, nous ajoutons des termes "extras" dans la variable $\mathbf{x}$. Par exemple, pour une décision quadratique comme sur la figure 3.4, nous prenons $\mathbf{x} = [1\ x_1\ x_2\ x_1^2\ x_2^2]$ et nous avons donc :

$$h(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2) \tag{3.4}$$

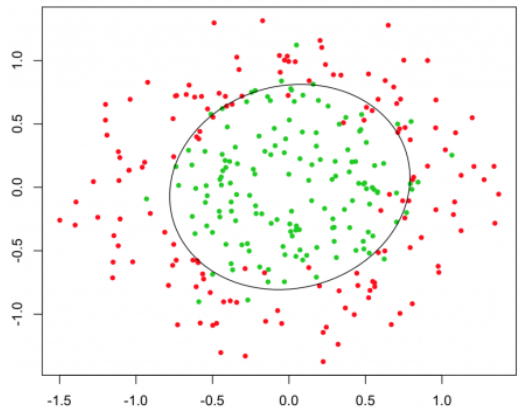

FIGURE 3.4 – Frontière de décision quadratique de degré 4.

Nous pouvons obtenir des frontières plus complexes en ajoutant plus de termes de degrés supérieurs ($\mathbf{x} = [1\ x_1\ x_2\ x_1^2\ x_2^2\ x_1^2 x_2\ x_1^2 x_2^2\ x_1^3 x_2\ ...]$) comme le montre la figure 3.5

L'inconvénient des variables $\mathbf{x}$ de degré supérieur est le sur-apprentissage (voir plus loin), ce qui signifie que la frontière de décision classifie quasi-parfaitement les données d'apprentissage, mais risque d'avoir des performances faibles pour les nouvelles données.

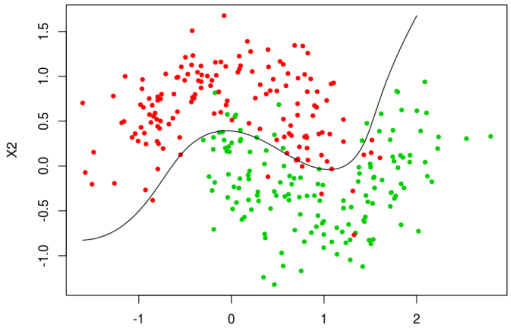

FIGURE 3.5 – Frontière de décision de degré supérieur.

1.2 Optimisation : Fonction de coût et descente du gradient

Le but de l'optimisation est de trouver les paramètres θ.
L'algorithme d'optimisation le plus commun est la descente du gradient.
Nous cherchons θ qui minimise une fonction de coût $J(\theta)$:

$$\arg\min_{\theta} J(\theta), \tag{3.5}$$

avec :

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h(x^i)), \tag{3.6}$$

m est le nombre d'échantillons x^i utilisés pour l'apprentissage.
La fonction $Cost(h(x_j^i)$ est une mesure de l'erreur entre la prédiction $h(x^i)$ et le vrai label y^i de l'échantillon x^i. Nous utilisons une fonction d'entropie croisée BCE (Binary Cross Entropy) :

$$Cost(h(x^i)) = -y^i log(h(x^i)) - (1-y^i) log(1-h(x^i)) \tag{3.7}$$

Les minimums des 2 termes de la BCE (voir figure 3.6) correspondent aux prédictions des 2 classes :

$$Cost(h(x^i)) = \begin{cases} -log(h(x^i)) & , si \quad y^i = 1 \\ -log(1-h(x^i)) & , si \quad y^i = 0 \end{cases} . \tag{3.8}$$

L'algorithme de descente du gradient consiste à mettre à jour de manière itérative θ, jusqu'à convergence vers un minimum local ou global, à l'aide de la formule suivante :

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \tag{3.9}$$

avec α le taux d'apprentissage (learning rate).
La dérivée de J se calcul comme suit :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m} (h(x^i) - y^i) x_j^i \tag{3.10}$$

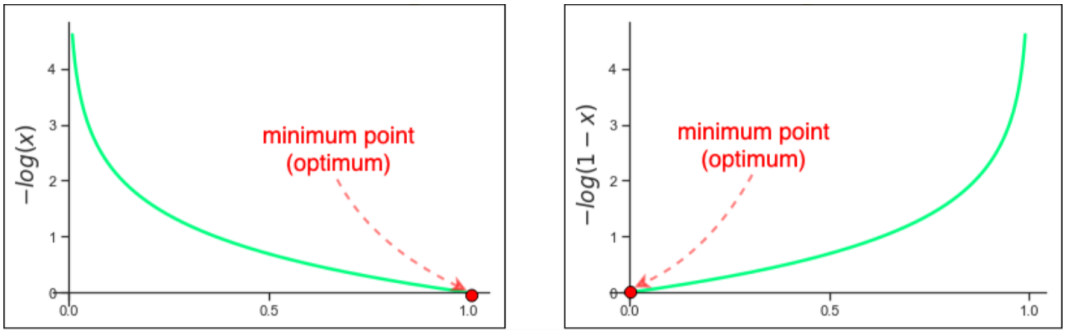

FIGURE 3.6 – Minimum des termes de la BCE.

La mise à jour de *theta* se fait donc avec la formule :

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - y^i) x_j^i \tag{3.11}$$

1.3 Initialisation

Les valeurs initiales de θ peuvent être choisies toutes nulles, de manière aléatoire ou selon une distribution normale.

1.4 Descente du gradient stochastique

Lorsque le nombre d'échantillons m est très grand, nous mettons à jour les paramètres θ à partir d'un seul échantillon en parcourant tout l'ensemble :
Pour i allant de 1 à m, faire :

$$\theta_j := \theta_j + \alpha (y^i - h(x^i)) x_j^i \tag{3.12}$$

Le fait de mettre à jour θ plus souvent permet une convergence plus rapide.

1.5 Mise à l'échelle

Pour une convergence rapide, il faut s'assurer que les intervalles de variation des échantillons sont similaires.
Cela se fait avec une normalisation des échantillons par rapport à leur moyenne μ :

$$x^i = \frac{x^i - \mu}{max(x^i) - min(x^i)} \tag{3.13}$$

Nous obtenons des échantillons de moyenne nulle dans l'intervalle $[-1, 1]$.

1.6 Condition de convergence

La condition d'arrêt de l'algorithme d'optimisation est fixé à la fois par le nombre d'itérations et un seuil minimum de J.
Il est aussi important de vérifier que J diminue. Si ce n'est pas le cas, il faut choisir un taux d'apprentissage plus petit. En effet, un grand taux apprentissage permet des grands sauts de gradient et donc une convergence plus rapide, mais au risque de converger vers un minimum local.

1.7 Optimisation avancée

Il existe d'autres algorithmes d'optimisation plus performants que la descente du gradient, tels que Conjugate Gradient, BFGS, L-BFGS. L'intérêt de ces algorithmes est qu'ils permettent une convergence plus rapide et d'éviter d'ajuster manuellement α.

1.8 Régularisation

Pour éviter le problème de sur-apprentissage cité précédemment, nous ajoutons un terme de régularisation à J :

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} -y^i log(h(x^i)) - (1-y^i)log(1-h(x^i)) + \frac{\lambda}{2m}\sum_{j=1}^{N}\theta_j^2 \qquad (3.14)$$

avec λ le coefficient de régularisation.
Notons qu'il ne faut pas régulariser θ_0.
La dérivée de J devient :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h(x^i) - y^i)x_j^i + \frac{\lambda}{m}\theta_j \qquad (3.15)$$

La mise à jour des paramètres s'effectue donc comme suit :

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h(x^i) - y^i)x_j^i \quad , \; j > 0 \qquad (3.16)$$

1.9 Classification multi-classes "One vs All"

Dans le cas de classification multi-classes où y prend plus de 2 valeurs, par exemple $y = \{1, 2, 3\}$, nous considérons plusieurs prédicteurs binaires $h_\theta^k(x)$ pour chaque classe k. La prédiction finale correspond à :

$$\arg\max_k(h_\theta^k(x)) \qquad (3.17)$$

1.10 Limites de la régression logistique : cas des images

Dans le cadre de notre étude, les données $x = [x_1\ x_2\ ...\ x_N]$ à classifier seront des images, avec N le nombre de pixels. Prenons l'exemple d'une image RGB de dimensions $25 \times 25 \times 3$, nous avons $N = 1875$.
Supposons que nous cherchons une décision non linéaire d'ordre $k = 2$:

$$h(x) = g(\theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2 + \theta_3 x_1 x_2 + \theta_4 x_1 x_3 ...) \tag{3.18}$$

Le nombre total des Features correspondant à une combinaison avec remise :

$$\frac{(k+n-1)!}{k!(n-1)!} = 3,5 \ \ Millions\ de\ Features \tag{3.19}$$

Nous obtenons donc un nombre très important de Features pour une simple image de dimension très réduite 25×25 et généralement, la taille des images ordinaires dépasse les millions de pixels. Cela a pour conséquence un temps d'apprentissage très lent.

2 Réseaux de neurones

La régression logistique peut être représentée schématiquement sous la forme :

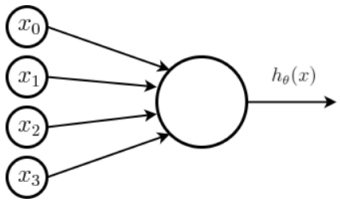

FIGURE 3.7 – Représentation neuronale de la régression logistique

Il s'agit en effet d'un simple réseau de neurones artificiel ANN (Artificial Neural Network) avec un seul neurone tel que :

x : l'entrée,

θ : les poids du réseau,

$x_0 = 1$: le biais,

h_θ : la fonction d'activation (ici, une sigmoïde).

Pour améliorer les performances de classification, nous utilisons plus de neurones et plus de couches cachées. La figure 3.8 montre un exemple avec une seule couche cachée.

Nous notons :

a_i^j : la fonction d'activation de l'unité (neurone) i dans la couche j,

a_0^j et x_0 correspondent aux termes de biais et sont toujours égales à 1.

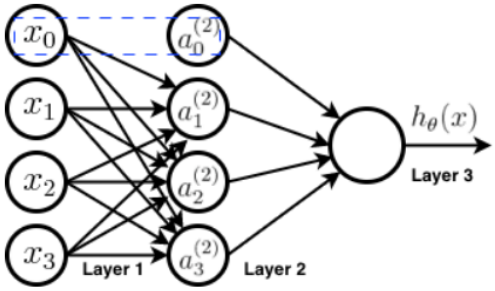

FIGURE 3.8 – Réseau de neurones avec une seule couche cachée.

Θ^j : la matrice de poids entre les couches j et $j+1$, que nous calculerons dans la suite par optimisation.

2.1 Propagation avant (FeedForward Propagation)

Nous cherchons à calculer la sortie du réseau par propagation en avant. Nous obtenons d'après le schéma précédent et après vectorisation :

$$a^2 = g(z^2) \tag{3.20}$$

avec :

$z^2 = \Theta^1 a^1$,

$a^1 = x$,

$a^2 = [a_0^2\ a_1^2\ a_2^2\ a_3^2]^T$.

et

$$h(x) = a^3 = g(z^3) \tag{3.21}$$

avec :

$z^3 = \Theta^2 a^2$.

2.2 Classification multi-classes : One vs All

Dans le cas d'un classifieur multi-classes, la sortie du réseau est constituée de plusieurs fonctions d'activation, une pour chaque classe comme représentée sur la figure 3.9.

Nous avons, pour la classe 1 : $\hat{y} = h(x) = [1\ 0\ ...\ 0]^T$, pour la classe 2 : $\hat{y} = h(x) = [0\ 1\ ...\ 0]^T$, pour la classe 3 : $\hat{y} = h(x) = [0\ 0\ 1\ 0...\ 0]^T$, et ainsi de suite.

Nous notons K le nombre de classes qui est égal au nombre d'unités s_L de la

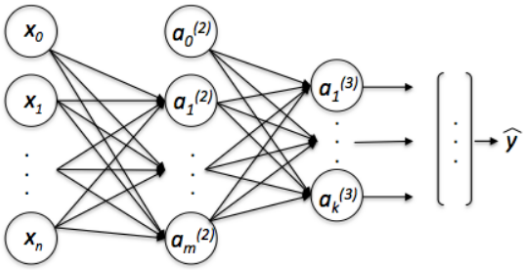

FIGURE 3.9 – Classifieur multi-classes.

dernière couche L (sans compter le biais).
Rappelons que dans le cas de classification binaire, nous utilisons $s_L = 1$ (une seule sortie).

2.3 Fonction de coût

La fonction de coût que nous cherchons à optimiser pour trouver les paramètres Θ est la suivante :

$$J(\Theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i log(h_k(x^i)) + (1-y_k^i) log(1-h_k(x^i))] - \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^l)^2 \tag{3.22}$$

avec :

(x, y) : l'échantillon d'apprentissage, y étant le label,

m : le nombre d'échantillons d'apprentissage,

K : le nombre de classes,

L : le nombre de couches du réseau,

$h_k(x)$: la k-ème sortie de la dernière couche,

Θ_{ji}^l : les éléments de la matrice de poids Θ^l entre les couches l et $l+1$,

s_l : le nombre d'unités de la couche l.

2.4 Optimisation : Rétropropagation du gradient (Backpropagation)

L'optimisation du réseau s'effectue en corrigeant l'erreur de chaque unité, en partant de la sortie vers l'entrée (inverse de la propagation avant).
Notons l'erreur δ_j^l l'erreur du noeud j dans la couche l.
Nous commençons par calculer l'erreur de la dernière couche L, pour chaque unité, entre la prédiction et la valeur du label y :

$$\delta_j^L = a_j^L - y_j, \tag{3.23}$$

que nous pouvons vectoriser :

$$\delta^L = a^L - y, \tag{3.24}$$

Ensuite, nous calculons l'erreur de la couche précédente $L-1$ comme suit :

$$\delta^{L-1} = (\Theta^{L-1})^T \delta^L .* g'(z^{L-1}) \tag{3.25}$$

avec :

$.*$: l'opérateur de multiplication élément par élément,

et

$$g'(z^{L-1}) = a^{L-1} .* (1 - a^{L-1}) \tag{3.26}$$

Nous faisons de même pour l'erreur de la couche précédente $L-2$:

$$\delta^{L-2} = (\Theta^{L-2})^T \delta^{L-1} .* g'(z^{L-2}) \tag{3.27}$$

avec

$$g'(z^{L-2}) = a^{L-2} .* (1 - a^{L-2}) \tag{3.28}$$

Nous continuons ainsi jusqu'à la première couche pour calculer δ^2.
Cela nous permet de déduire la dérivée de la fonction du coût :

$$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^l} = a_j^l \delta_i^{l+1} \tag{3.29}$$

Notons que le terme de régularisation n'a pas été considéré dans ce calcul. Le calcul de $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^l}$ permet d'optimiser les poids du réseau par descente du gradient ou autre algorithme d'optimisation.

2.5 Initialisation aléatoire

L'initialisation des poids Θ avec des valeurs nulles ne marche pas dans le cas des réseaux de neurone, car nous obtenons les mêmes a_i^j pour toutes les unités et toutes les couches cachées calculent donc la même fonction.
Nous initialisons donc chaque Θ_{ij}^l avec une valeur aléatoire dans $[-\epsilon, \epsilon]$.
Une stratégie efficace est de choisir ϵ en fonction du nombre d'unités :

$$\epsilon = \frac{\sqrt{6}}{s_l + s_{l+1}} \tag{3.30}$$

3 Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs ou CNN (Convolutional Neural Networks) utilisent des filtres dits de convolution, bien qu'il s'agisse de filtre de corrélation

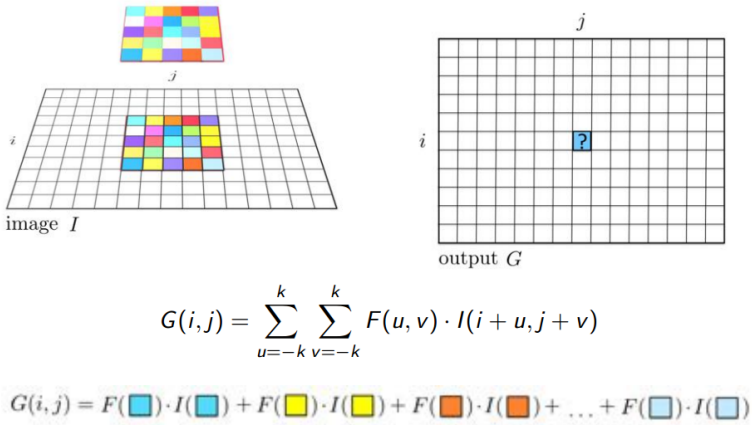

FIGURE 3.10 – Filtre CNN

croisée (Cross Corrélation), mais en apprentissage profond le terme convolution est utilisé par abus de langage.
L'opération de convolution utilisée est illustrée sur la figure 3.10.

Dans le cas des images, les CNN sont plus performants que les ANN (Artificial Neural Networks) vus dans la section précédente, à cause de l'utilisation des filtres.
Le principe des CNN consiste en l'apprentissage par rétro-propagation des coefficients de filtres plutôt que d'une matrice de poids des neurones.
Avant d'aborder les CNNs, il est important de définir quelques notions essentielles.

3.1 Padding

Lorsque nous appliquons un filtre de taille (f, f) à une image de taille (n, n), le résultat du filtrage donne une image rétrécie de taille $(n - f + 1, n - f + 1)$. Afin de garder la même taille d'image, nous ajoutons à l'image, avant le filtrage, des bords de taille $2p$ afin d'obtenir une image de taille $(n + 2p, n + 2p)$, avec :

$$p = \frac{f - 1}{2} \tag{3.31}$$

3.2 Stride

La taille de l'image après filtrage va dépendre aussi du pas de déplacement de la fenêtre de convolution (filtre).
Avec un pas Stride de taille s, nous obtenons une image de dimension :

$$[\frac{n + 2p - f}{s} - 1] \tag{3.32}$$

avec [.] l'opérateur retournant la partie entière inférieure.

3.3 Volume de convolution

Pour une image RGB, nous utilisons des filtres à 3 canaux, le résultat du filtrage donne une image à un seul canal comme l'illustre l'exemple de la figure 3.11.

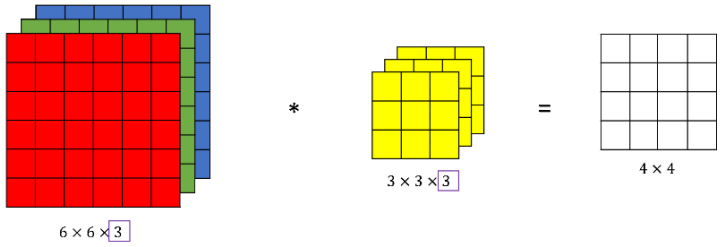

FIGURE 3.11 – Volume de convolution CNN

Chaque canal de l'image est convolué avec le canal du filtre correspondant et les résultats sont additionnés.

3.4 Filtres multiples

Dans le cas des CNN nous utilisons de multiples filtres pour chaque couche du réseau (équivalents des unités des ANN).
La convolution d'une image de taille (n, n, n_c) avec n'_c multiples filtres de taille (f, f, n_c) chacun, donne une image de dimension $(n - f + 1, n - f + 1, n'_c)$: la profondeur de l'image en sortie correspond au nombre de filtres. L'image 3.12 illustre ce principe.

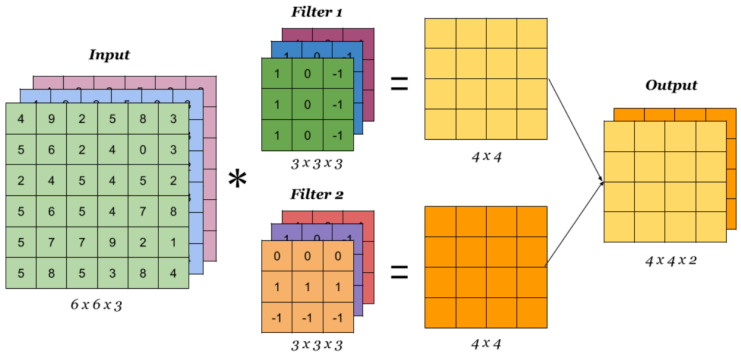

FIGURE 3.12 – Filtres CNN multiples

3.5 Couches CNN

La figure 3.13 montre l'exemple d'une couche CNN.

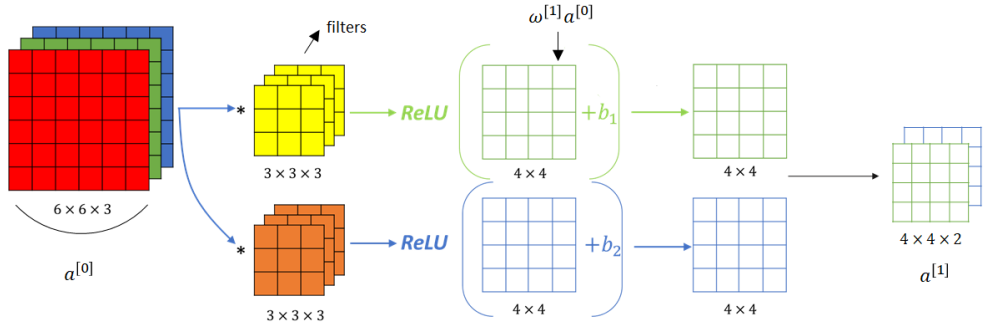

FIGURE 3.13 – CNN à une couche

Nous notons a^0 la première couche correspondant à l'image en entrée. Les poids W^i à optimiser correspondent aux coefficients des filtres.
Une fonction d'activation ReLU, illustrée sur la figure 3.14, est appliquée à la sortie de chaque filtre additionnée avec un biais b^1, nous obtenons donc sur la deuxième couche :

$$a^1 = ReLU(W^1.a^0 + b^1) \tag{3.33}$$

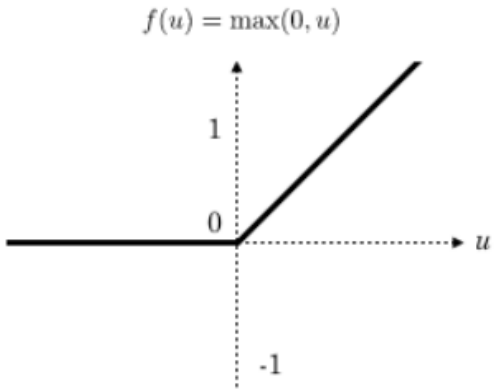

FIGURE 3.14 – Fonction d'activation ReLU

3.6 Pooling

Il s'agit d'appliquer un opérateur à la sortie d'une couche CNN, afin de réduire la taille des images et donc d'accélérer le traitement et en même temps rendre la détection plus robuste en ne gardant que les valeurs d'activation significatives.
La figure 3.15 illustre les deux opérateurs de pooling les plus utilisés, Max Pooling et Average Pooling (maximum et moyenne), appliqués avec des filtres de taille $f = 2$ et un Stride $s = 2$.
La taille de l'image de sortie est $[\frac{n + 2p - f}{s} - 1]$, avec généralement un padding $p = 0$.

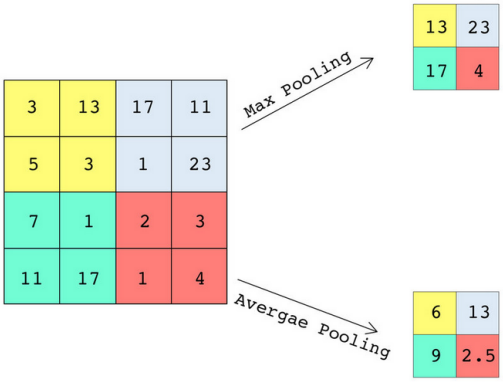

FIGURE 3.15 – Max Pooling et Average Pooling

Max Pooling permet de ne garder que les Features significatifs (contours, yeux, etc.). Average Pooling est utilisé dans les couches très profondes.

4 Architectures fondamentales en vision par ordinateur

Nous allons présenter les CNN fondamentaux dont les schémas inspirent la majorité des modèles de vision par ordinateur.

4.1 LeNet 5

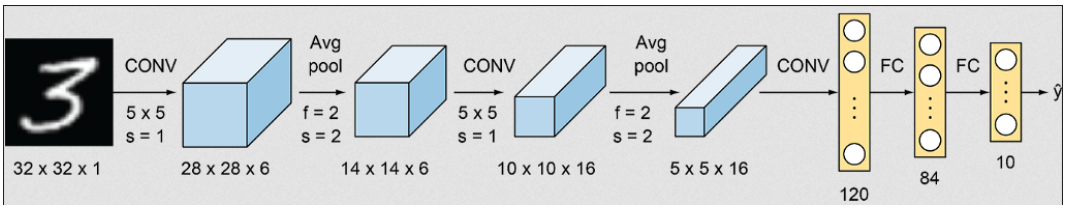

FIGURE 3.16 – Architecture CNN LeNet 5

L'architecture LeNet [LBBH98] alterne des couches convolutives et Average Pooling.
Les couches de convolution utilisent des fonctions d'activation Sigmoïde et Tanh (forme similaire à sigmoïde mais avec des valeurs négatives).
La dernière sortie convolutive est aplatit en un seul vecteur.
Les couches en jaune FC (Fully connected) relient chaque unité à toutes les autres (ANN).
La sortie finale (ici de longueur 10 pour les 10 classes des chiffres) est envoyée

à une fonction d'activation Softmax, illustrée sur la figure 3.17, pour la prédiction multi-classes.

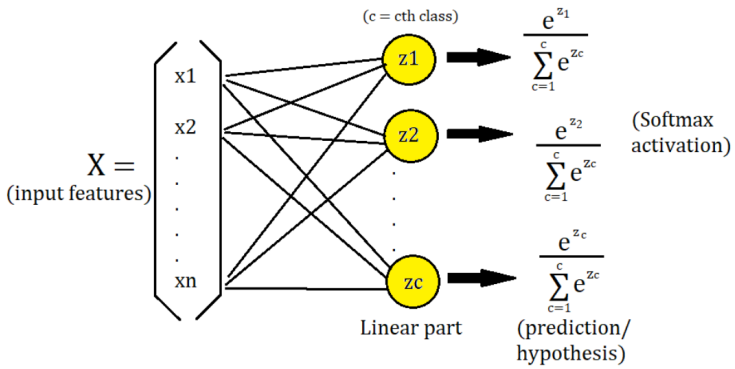

FIGURE 3.17 – Illustration Softmax : normalisation des valeurs de sortie dans le cas multi-classes.

4.2 AlexNet

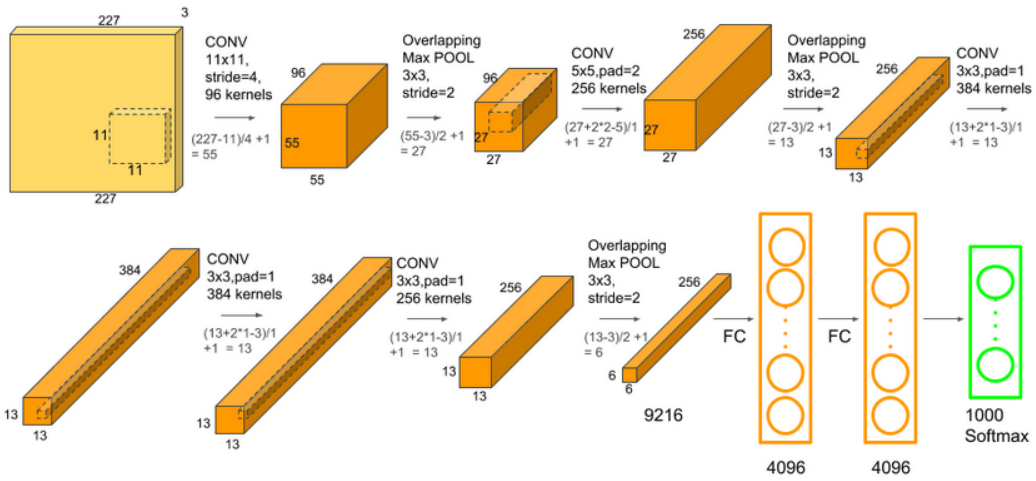

FIGURE 3.18 – Architecture CNN AlexNet

L'architecture AlexNet [KSH17] est similaire à LeNet 5, mais plus profonde. Le modèle prend en entrée des images RGB de plus grande taille et la dataset d'apprentissage "ImageNet" est plus large et contient plus de classes (1000). Une fonction d'activation ReLU est utilisée.
Notons le schéma uniforme de l'architecture : plus une couche est profonde, plus la taille des filtres diminue et leur nombre augmente. Il s'agit de l'architecture qui a convaincu la communauté de vision par ordinateur

d'utiliser le Deep Learning.

4.3 VGG 16

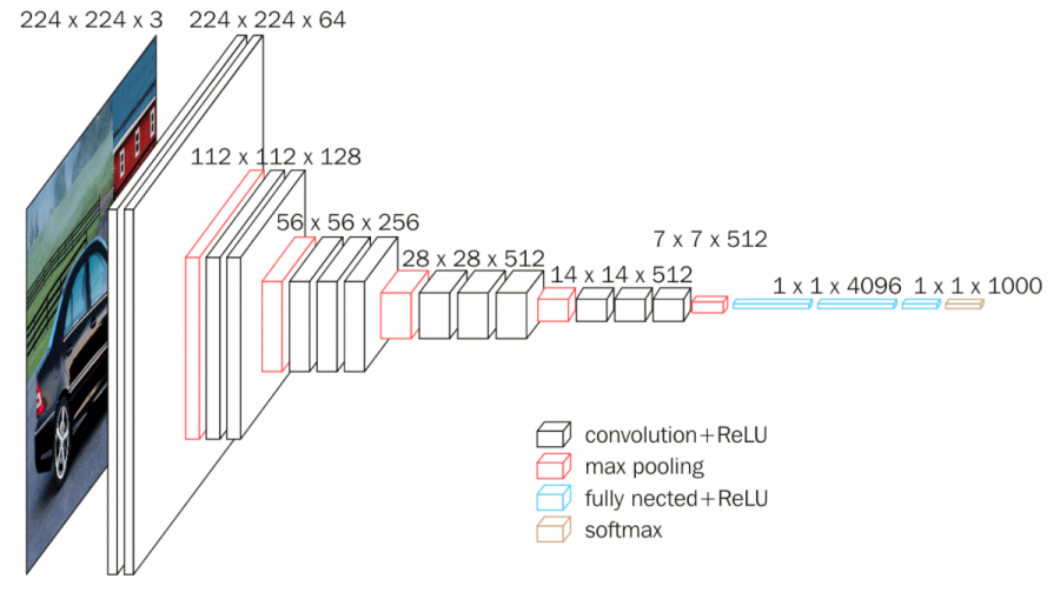

FIGURE 3.19 – Architecture CNN VGG 16

L'idée du modèle VGG 16 [SZ14] est d'utiliser un schéma simple pour éviter d'avoir beaucoup de hyperparamètres : filtres de taille 3×3, stride $s = 1$ et padding='Same' (sortie de même taille que l'entrée) pour toutes les couches convolutives, et Max Pooling 2×2 et stride $s = 2$ pour toutes les couches Pooling.
Le schéma est assez uniforme : le nombre de filtre augmente et la taille de sortie diminue.

4.4 ResNet

Vanishing/Exploding Gradient : Nous avons vu que lors de l'apprentissage de réseaux de neurones par rétro-propagation et descente du gradient (ou autre), les poids du réseaux sont mis à jour à partir de la dérivée de la fonction du coût. Dans le cas de réseaux très profond, la dérivée risque de diminuer ou augmenter de manière exponentielle lorsque nous approchons des couches inférieures, nous parlons de problèmes de "Vanishing/Exploding Gradient". Cela à pour effet de modifier les poids des couches inférieures soit de très peu (apprentissage lent) ou avec de très larges valeurs (divergence).
Une solution partielle est d'initialiser les poids selon une loi normale ou "Xavier Inititalizer" [GB10], ce qui permet d'éviter que les poids n'explose

pas ou ne diminue pas rapidement.

Residual Block : Une autre solution, illustrée sur la figure 3.20, est d'envoyer l'activation d'une couche vers une couche plus profonde.

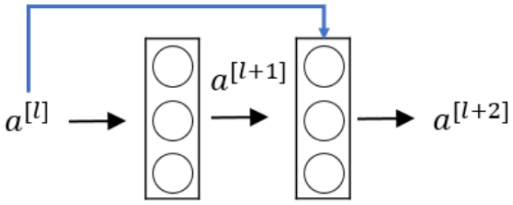

FIGURE 3.20 – Residual Block

Nous obtenons :

$$a^{l+2} = g(z^{l+1} + a^l) \tag{3.34}$$

La contribution des poids des couches précédentes permet d'éviter le problème de Vanishing gradient.
L'architecture ResNet [HZRS16], illustrée sur la figure 3.21, utilise plusieurs Residual Block en chaine et permet l'apprentissage de réseaux très profond (plus de 100 couches), en évitant les problèmes de "Vanishing/Exploding Gradient".

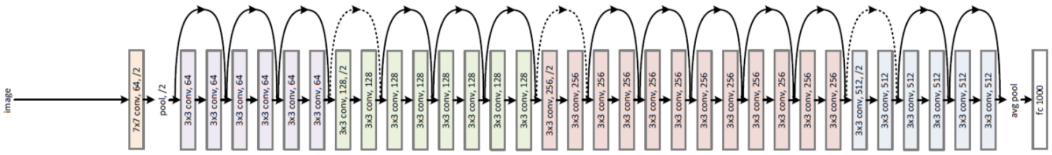

FIGURE 3.21 – Architecture CNN ResNet

La figure 3.22 illustre l'erreur d'un modèle avec et sans Residual Block, en fonction du nombre de couches. Nous constatons que l'erreur du modèle ResNet continue de diminuer (ou se stabilise) pour des modèles profonds.

4.5 Inception

L'idée de l'architecture Inception [SLJ+15] est qu'au lieu de décider de la taille des filtres de convolution, nous utilisons à chaque couche des filtres de plusieurs tailles (1×1), (3×3) et (5×5) et nous combinons les résultats en les concaténant. Pendant l'apprentissage, le modèle sélectionne de manière automatique le filtre ou les filtres les plus utiles. La figure 3.23 illustre un module Inception.

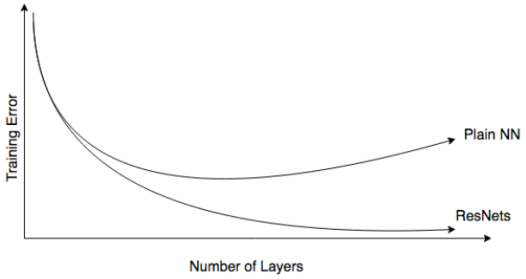

FIGURE 3.22 – Erreur en fonction du nombre de couches pour un modèle avec et sans Residual Block

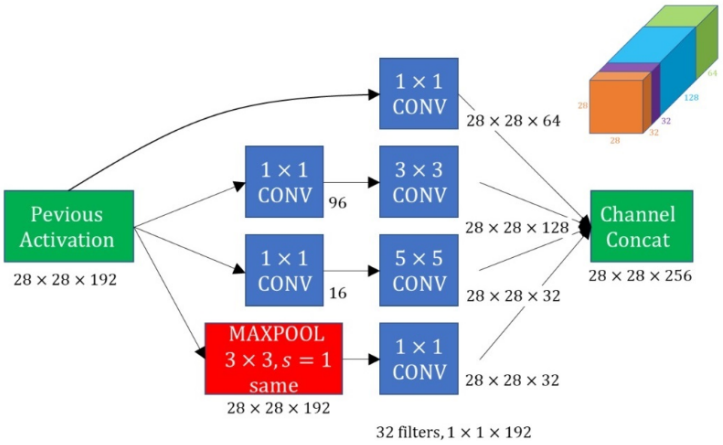

FIGURE 3.23 – Module Inception

Notons que l'utilisation d'une convolution (1×1) a pour intérêt de réduire (ou augmenter) la profondeur de l'image en sortie [LCY13], de la même manière que le Pooling modifie la largeur et la hauteur de l'image. Cela permet d'accélérer le temps de calcul.
L'architetcure Inception, illustrée sur la figure 3.24, utilise plusieurs modules Inception en chaine. Il s'agit d'un modèle dit "GoogLeNet" (hommage à LeNet par Google).

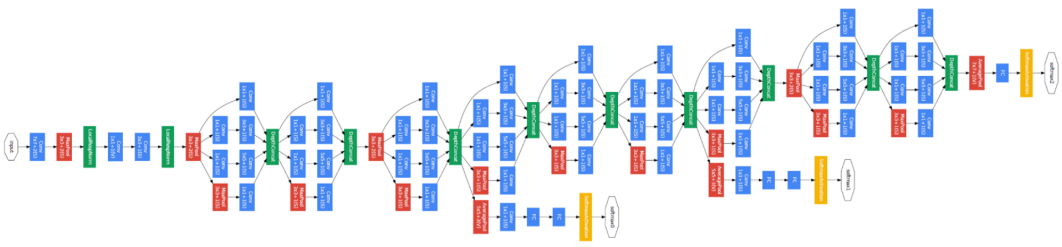

FIGURE 3.24 – Architecture CNN Inception (GoogLeNet).

Notons que le modèle contient des sorties intermédiaires pour vérifier que même les couches peu profondes permettent une bonne prédiction.

Dans [SIVA17], une combinaison Inception et ResNet est proposée.

5 Évaluation

L'évaluation d'un modèle s'effectue en divisant les données, généralement, en 70% pour l'apprentissage et 30% pour le test.
La matrice des poids est calculée en optimisant le modèle sur les données d'apprentissage.
L'évaluation s'effectue en calculant l'erreur J_{test} et d'autres métriques sur les données de test.

5.1 Sous-apprentissage et sur-apprentissage

Nous parlons de sous-apprentissage (Underfitting) d'un modèle, lorsque les erreurs J_{test} et J_{train}, respectivement, sur les ensembles de test et d'apprentissage, sont grandes. Cela signifie que le modèle classifie mal beaucoup de données.
Nous parlons de sur-apprentissage (Overfitting) d'un modèle, lorsque l'erreur J_{train} est faible et l'erreur $J_{test} >> J_{train}$. Cela signifie que le modèle classifie bien les données d'apprentissage et beaucoup moins bien les données de test. Nous parlons d'importante erreur de généralisation aux nouvelles données (non vues durant l'apprentissage).
Le sous-apprentissage peut être résolu en utilisant un réseau de neurones plus profond (deep learning) avec un plus grand nombre de couches. L'inconvénient des réseaux profonds est le risque de sur-apprentissage.
Le sur-apprentissage peut être résolu en ajoutant plus de données et par régularisation.
Le tableau suivant montre les solutions aux problèmes de sous-apprentissage et de sur-apprentissage :

Sous-apprentissage	Sur-apprentissage
Augmenter degré polynomial	Ajouter plus de données
Ajouter des Features	Supprimer des Features
Diminuer la valeur du coefficient de régularisation λ	Augmenter λ

5.2 Sélection du modèle : hyperparamètres

Afin de décider des hyperparamètres du modèle (nombre de couches, nombre d'unités, nombre de Features, coefficient de régularisation λ, etc.), nous devisons les données d'apprentissage en trois groupes : 60% pour l'apprentissage, 20% pour la validation et 20% pour le test. Dans le cas de dataset très large (plus de 100000 échantillons), nous utilisons 80% pour

l'apprentissage, 10% pour la validation et 10% pour le test.
Nous testons plusieurs valeurs d'un paramètre (par exemple λ) du modèle sur les données d'apprentissage et de validation, et sélectionnant la valeur qui donne l'erreur de validation J_{val} la plus faible. Le modèle est ensuite évalué sur l'ensemble de test pour calculer l'erreur de généralisation J_{test}.

5.3 Métriques d'évaluation

Notons :

TP : taux de vrais positifs (échantillons positifs bien classés),

FP : taux de faux positifs (échantillons positifs mal classés),

TN : taux de vrais négatifs (échantillons négatifs bien classés),

FN : taux de vrais négatifs (échantillons négatifs mal classés).

Il existe différentes métriques d'évaluations :

Accuracy : Cette mesure permet de calculer le taux de bonnes classifications. La formule s'écrit :

$$Accuracy = \frac{TP + TN}{nombre\ total\ des\ echantillons} \tag{3.35}$$

Précision : Permet de mesurer les performances de prédiction des échantillons positifs. La formule de la précision est la suivante :

$$Precision = \frac{TP}{nombre\ total\ de\ predictions\ positives} = \frac{TP}{TP + FP} \tag{3.36}$$

Lorsque sa valeur est importante (proche de 1), le modèle ne classifie une donnée en tant que positif que lorsqu'il est sûr du résultat. Un exemple dans le médical est de ne diagnostiquer une maladie que lorsque nous sommes quasi sûrs, le problème est qu'il y a un risque de ne pas signaler certains cas positifs.

Rappel (Recall) : Permet de mesurer les performances du modèle à éviter de prédire beaucoup de faux positifs. La formule de la précision est la suivante :

$$Recall = \frac{TP}{nombre\ total\ des\ echantillons\ positifs} = \frac{TP}{TP + FN} \tag{3.37}$$

Lorsque sa valeur est importante, le modèle classifie plus d'échantillons en tant que positifs. Dans l'exemple de diagnostic médical, cela permettra d'éviter de ne pas diagnostiquer des cas positifs, mais avec le risque d'augmenter le nombre de faux positifs.
Il appartient aux utilisateurs spécialistes (médecin par exemple) de paramétrer le système pour trouver un bon compromis entre précision et rappel.

La précision et le rappel sont mesurés en faisant varier le seuil de prédiction $h(x)$ du modèle et nous traçons la courbe Precision/Recall pour trouver le seuil optimal :

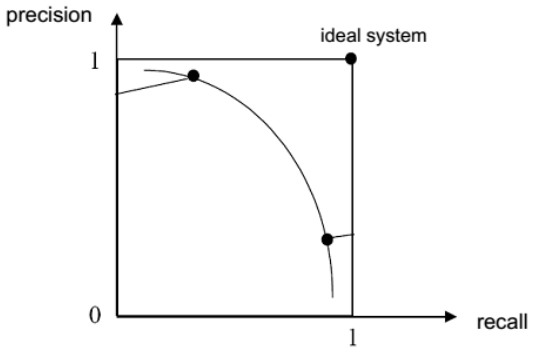

FIGURE 3.25 – Courbe précision rappel : seuil optimal.

Le point le plus haut à droite correspond au cas idéal.

F1-Score : C'est une mesure qui permet de comparer la précision et le rappel de différents algorithmes. Pour cela, il faut éviter de calculer une moyenne entre la précision P et le rappel R et d'utiliser plutôt le F_1 Score :

$$F_1\ SCORE = \frac{2PR}{P+R} \tag{3.38}$$

6 Conclusion

Nous avons présenté dans ce chapitre les modèles d'apprentissage automatique les plus utilisées aujourd'hui dans le domaine des images : Les réseaux de neurones artificiels ANN et convolutifs CNN. Les différents concepts abordés des algorithmes d'optimisation, de sélection des hyper-paramètres, des architectures fondamentales et des méthodes d'évaluation, seront utilisés dans le chapitre suivant de la mise en correspondance par apprentissage, ainsi que dans les autres chapitres.

Chapitre 4

Mise en correspondance par apprentissage automatique

Sommaire

Avant de présenter plus en détails les méthodes de mise en correspondance fondées sur l'apprentissage automatique et l'apprentissage profond, nous allons commencer par découvrir les différents types d'approches et leur évolution.
L'utilisation de l'apprentissage automatique et profond pour la détection, la description et la mise en correspondance des points d'intérêt a commencé un peu avant le "big bang" du "deep learning". Dans [KS04], l'analyse en composante principale PCA est utilisée pour réduire la dimensionnalité du descripteur SIFT. L'inconvénient du PCA est que nous n'avons pas d'informations sur les labels des classes. Pour résoudre ce problème, il est possible de réduire la dimension du descripteur avec une analyse discriminante linéaire (LDA) comme dans [CMM10]. Dans [SVZ14], la réduction de dimensionnalité du descripteur est reformulée en optimisation convexe pour de meilleures performances.
La méthode proposée dans [JGB08] est l'une des premières à utiliser des réseaux de neurones convolutifs (CNN) pour l'apprentissage des descripteurs et la mise en correspondance des points d'intérêt. Le but est d'entrainer un réseau siamois à identifier les patchs similaires. La Dataset utilisée est obtenue par déformations géométriques et photométriques d'images. Le succès de l'apprentissage profond pour la mise en correspondance d'images est du à plusieurs facteurs : la possibilité d'utiliser des CNN pré-entrainés sur des Datasets larges telles que ImageNet [DDS+09], les CNN sont capables d'apprendre différentes textures de l'image dans chaque couche et les descripteurs des points d'intérêt peuvent être obtenus en supprimant les dernières couche (Fully Connected, etc.). Dans [FDB14] il a été démontré qu'un CNN permet d'obtenir de meilleurs résultats de mise en correspondance que SIFT. Dans [ZK15] des architectures plus profondes sont utilisées pour améliorer les performances. Toutefois, il a été démontré dans [BRPM16] que les performances peuvent être améliorées avec un CNN (TFeat) peu profonds (Shallow) en utilisant des triplets d'échantillons d'apprentissage : patch de référence, un patch positif (similaire) et patch négatif (différent). Un simple CNN à 6 Layers (L2-Net) a aussi été utilisé dans [TFW17] mais avec des fonctions objectifs (loss function) complexes. Le modèle HardNet proposé dans [MMRM17], montre qu'avec une simple fonction objective, mais une stratégie plus avancée de sélection des triplets d'apprentissage, il surpasse les performances de L2-Net. Le modèle SOSNet [TYF+19] permet d'améliorer les performances en optimisant les distances intra et inter classes.
Dans d'autres approches "modernes", plutôt que d'utiliser la pipeline classique (détection, description, correspondance) elles utilisent un apprentissage "end to end" unifiant plusieurs parties de la pipeline de manière différentiable. Dans [YTLF16], un modèle LIFT est entrainé pour effectuer à la fois la détection, l'estimation d'orientation et la description des points d'intérêt. Le modèle LF-Net [OTFY18] utilise le même principe avec une architecture différente. Il existe de plus en plus de modèles utilisant ce type d'approches : Superpoint [DMR18], IMIPs [CBS18], DELF [NAS+17], D2-Net [DRP+19], UR2KiD [YNHB20] ou SuperGlue [SDMR20]. Dans [YTO+18] un modèle

est utilisé pour améliorer la mise en correspondance en détectant les valeurs aberrantes (outliers) et qui, combiné à LIFT, il affiche des performances qui dépassent largement la méthode populaire RANSAC [FB81].
Ces approches modernes donnent de très bons résultats pour les images complexes (larges variations géométriques et éclairage) mais les méthodes fondées sur la pipeline classique donne de meilleurs résultats dans les autres cas plus simples.

1 PCA-SIFT

PCA-SIFT [KS04] est une variante de SIFT dont le but est de réduire la dimensionnalité du descripteur en utilisant une matrice de projection estimée par apprentissage. Cette méthode a l'avantage d'être plus précise et plus rapide que SIFT.
Un pré-traitement est d'abord appliqué :
1) Détection SIFT des patchs de points d'intérêt 41×41 et orientation par rapport au gradient dominant.
2) Extraction des cartes de gradient verticales (39×39) et horizontales (39×39).
3) Vectorisation et concaténation ($(2 \times 39 \times 39) = 3042$).
4) Normalisation unitaire pour réduire les variations de luminance.
Ensuite, le principe est d'utiliser un ensemble d'apprentissage de N patchs afin de créer une matrice de données X (de taille $N \times D$) composée de N descripteurs de dimension D (ici égal à 3042). Le but est de réduire la dimension du descripteur de D à K.
Après normalisation de X (soustraction de la moyenne), nous calculons la matrice de covariance C de X :

$$C = X^T X \tag{4.1}$$

Nous appliquons la décomposition en valeurs singulières :

$$C = U \Sigma V \tag{4.2}$$

Nous utilisons les K premières colonnes U' de U (vecteurs propres) pour projeter un nouveau descripteur x dans un nouveau espace de dimension K :

$$x' = U'^T x \tag{4.3}$$

Comme il a été démontré dans [KS04], comparé à SIFT, PCA-SIFT permet d'améliorer les performances de mise correspondance. La taille du descripteur est réduite de 3042 à 20 pour un maximum de performances. Lorsque $K > 40$, les performances chutent, car des informations inutiles sont considérées.

2 Optimisation convexe

Dans [SVZ14], une méthode d'apprentissage des descripteurs par optimisation convexe est proposée. L'algorithme est le suivant :
Le patch x de point d'intérêt détecté (par SIFT, Harris ou autre) est rectifié par rapport à l'orientation du gradient dominant et à la déformation affine.
Les gradients d'orientation de chaque pixel du patch sont calculés comme dans 2.1 sur p directions (bins) entre 0 et 2π. Nous obtenons ainsi p images des gradients.
Chaque image de gradient est convolué avec un ensemble de noyaux (Kernels), appelés "Pooling Regions", ayant des supports et des positions différentes. Nous obtenons un descripteur $\Phi(x)$ de dimension $p.q$, avec q le nombre de noyaux.
Les noyaux utilisés sont définis par :

$$k(u, v; \rho, \alpha, \sigma) \sim \exp(-\frac{(u - \rho\cos(\alpha))^2 + (v - \rho\sin(\alpha))^2}{2\sigma^2}), \tag{4.4}$$

avec (u, v) les coordonnées pixels et (ρ, α) les coordonnées polaires de la gaussienne d'écart type σ.
La figure 4.1 montre différents noyaux obtenus en faisant varier les paramètres (ρ, α, σ).

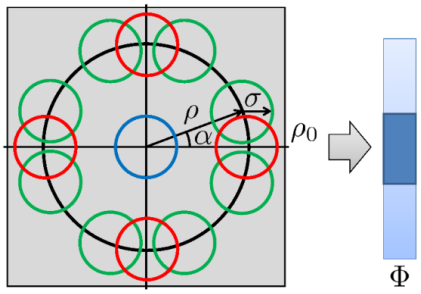

FIGURE 4.1 – Noyaux "Pooling Regions" regroupés dans des cercles (noir et bleu) et descripteur Φ obtenu après convolution avec ces noyaux.

Le descripteur est séparé en plusieurs éléments correspondants à chaque région :

$$\phi_{i,j,c}(x) = \sqrt{w_i}.\Phi_{i,j,c}(x), \tag{4.5}$$

avec i l'indice du cercle, j l'indice du noyau et c le numéro du canal. w_i est un masque binaire de sélection des régions : $w_i = 1$ si la région est sélectionnée, sinon $w_i = 0$. les régions pertinentes pour la mise en correspondance sont sélectionnées par apprentissage. L'ensemble d'apprentissage est séparé en deux ensembles P de pairs de patch positifs (similaires) et N de pairs de patch négatifs (différents). Nous ajoutons une contrainte qui permet de maximiser la distance entre ces deux ensembles :

$$d(\mathbf{x}, \mathbf{y}) + 1 < d(\mathbf{u}, \mathbf{v}), \quad \forall(\mathbf{x}, \mathbf{y}) \in P \quad et \quad \forall(\mathbf{u}, \mathbf{v}) \in N, \tag{4.6}$$

avec d une distance L_2 entre les descripteurs

$$d(x,y) = \sum_{i,j,c} (\sqrt{w_i}.\Phi_{i,j,c}(x) - \sqrt{w_i}.\Phi_{i,j,c}(y))^2 \tag{4.7}$$

À partir de ces deux dernières équations, nous nous retrouvons avec un problème d'optimisation convexe qui consiste à minimiser par apprentissage la fonction objective suivante :

$$\arg\min_{w \geq 0} \sum_{(\mathbf{x},\mathbf{y},\mathbf{u},\mathbf{v})} \mathcal{L}(w^T(\psi(\mathbf{x},\mathbf{y}) - \psi(\mathbf{u},\mathbf{v}))) + \mu_1 ||w||_1 \tag{4.8}$$

tel que :

$$\psi(\mathbf{x},\mathbf{y}) = \sum_{j,c} (\Phi_{i,j,c}(x) - \Phi_{i,j,c}(y))^2 \tag{4.9}$$

et :

$$\mathcal{L}(z) = max(z+1, 0) \tag{4.10}$$

le terme de régularisation $\mu_1 ||w||_1$ force $w = 0$.
L'optimisation permet donc de trouver w, les régions pertinentes pour la mise en correspondance).
Enfin, nous calculons le descripteur :

$$\tilde{\Phi} = \sqrt{w}\Phi \tag{4.11}$$

Le descripteur est normalisé pour le rendre invariant aux changements de luminance. Le facteur de normalisation $T(x)$ est calculé directement à partir du patch, indépendamment des noyaux. $T(x)$ est le ϵ-quantile de la distribution de magnitude de gradient du patch, avec $\epsilon = 0,8$. La normalisation est réalisée pour chaque région par :

$$min\{\frac{\tilde{\Phi}_i(x)}{T(x)}, 1\} \tag{4.12}$$

Le descripteur peut aussi être normalisé par rapport à la moyenne et la variance des gradients du patch.
Pour la réduction de dimensionnalité, nous cherchons une matrice de projection W qui, en plus de réduire la dimension du descripteur, permet de séparer les paires positives et négatives. Cela revient à ajouter, de manière similaire à ce qui a précédé, la contrainte :

$$d_W(\mathbf{x},\mathbf{y}) + 1 < d_W(\mathbf{u},\mathbf{v}), \quad \forall(\mathbf{x},\mathbf{y}) \in P \quad et \quad \forall(\mathbf{u},\mathbf{v}) \in N, \tag{4.13}$$

avec d_W la distance entre les paires après projection avec W :

$$d_w(x,y) = ||W.\phi(\mathbf{x}) - W.\phi(\mathbf{y})||_2^2 \tag{4.14}$$

après développement :

$$d_w(x,y) = \theta(\mathbf{x},\mathbf{y})^T.A.\theta(\mathbf{x},\mathbf{y}) \tag{4.15}$$

avec :

$$\theta(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x}) - \phi(\mathbf{y}) \tag{4.16}$$

et

$$A = W^T.W \tag{4.17}$$

Le problème est reformulé comme précédemment en optimisation convexe :

$$\arg\min_{A\geq 0} \sum_{(\mathbf{x},\mathbf{y},\mathbf{u},\mathbf{v})} \mathcal{L}(\theta(\mathbf{x},\mathbf{y})^T.A.\theta(\mathbf{x},\mathbf{y}) - \theta(\mathbf{u},\mathbf{v})^T.A.\theta(\mathbf{u},\mathbf{v})) + \mu_\star ||A||_\star \tag{4.18}$$

avec $||A||_\star$ est la norme nucléaire de A qui correspond à la somme des valeurs singulières de A.
La paramètre de régularisation $\mu_\star$ permet de régler la dimension de l'espace de projection : la dimension de l'espace de projection dépend du rang de A.
L'optimisation des deux fonctions objectives présentées est effectuée par l'algorithme RDA à cause de la taille des échantillons d'apprentissage et de l'utilisation des sommes.
RDA est efficace pour les problèmes d'optimisation de la forme :

$$\arg\min_{w} \frac{1}{T} \sum_{t=1}^{T} f(w, z_t) + R(w) \tag{4.19}$$

avec z_t est le t-ième échantillon et $R(w)$ le terme de régularisation.
Tous les détails des mises à jour des paramètres w et A des équations 4.8 et 4.18 par RDA sont disponibles dans [SVZ14].

3 Apprentissage de la mise en correspondance par CNN

Dans [JGB08], deux architectures CNN, illustrées sur la figure 4.2, sont proposées pour l'apprentissage des descripteurs et pour la mise en correspondance.

La dataset utilisée est créée par transformations géométriques appliquées sur un ensemble de patchs extraits par DoG (comme dans SIFT). L'utilisation de transformations de synthèse à l'avantage, par rapport aux méthodes traditionnelles, de contrôler la quantité d'invariance des transformations que nous voulons que le modèle apprenne.
Afin de ne retenir pendant l'apprentissage que les points d'intérêt fiables pour la mise en correspondance, une homographie aléatoire est appliquée à chaque image. Étant donné l'homographie connue, nous pouvons déduire la position de chaque pixel dans la deuxième image. La détection par DoG est effectuée sur l'image transformée et une comparaison est effectuée avec la position, l'échelle et l'orientation des points d'intérêt de la première image après transformation. Des seuils de tolérance, de position, d'échelle et d'orientation sont utilisées.

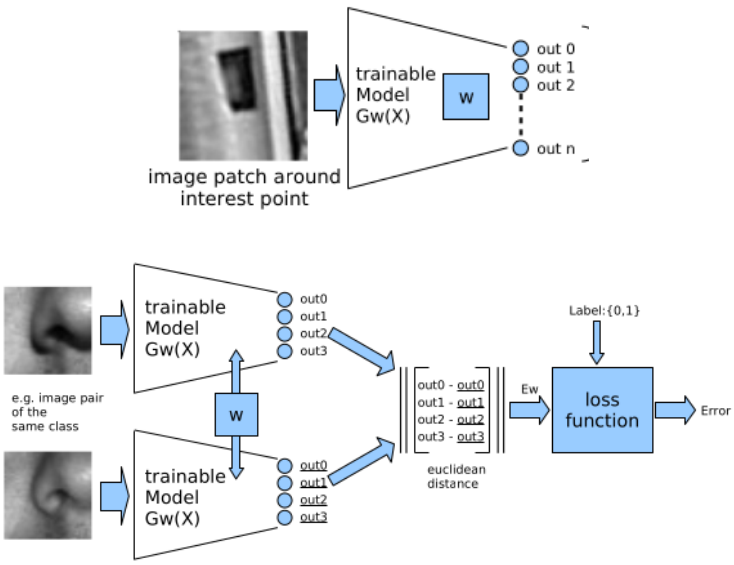

FIGURE 4.2 – Architectures de description et de mise en correspondance des points d'intérêt.

Si les points d'intérêts sont proches, ils sont considérés comme fiables pour la mise en correspondance et sont retenus comme échantillons d'apprentissage. La figure 4.3 illustre ce principe.

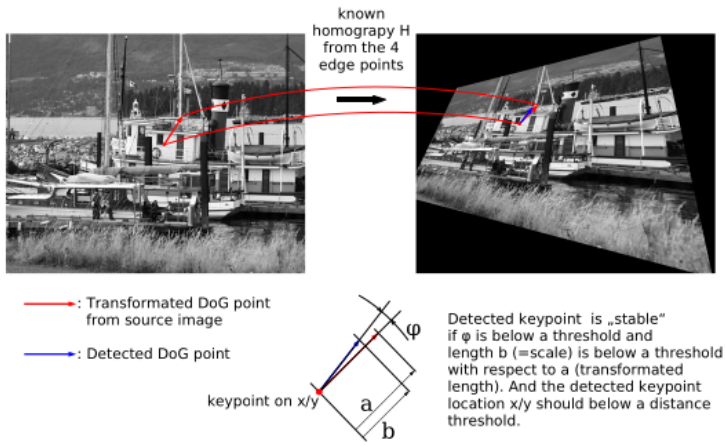

FIGURE 4.3 – Application d'homographie et comparaisons des points d'intérêt.

Mise en correspondance avec l'architecture 1 : Le premier modèle de description de la figure 4.2 reçoit à son entrée un patch X de point d'intérêt et, à l'aide d'une fonction de mapping $G_w(X)$, il renvoie en sortie un descripteur de dimension n qui permet de faire la mise en correspondance directement. En effet, ce modèle doit être entrainé sur les patchs de chaque image dont nous voulons faire la mise en correspondance, en appliquant les transformations

géométriques discutées précédemment.
Ce modèle peut être vu comme un classifieur où chaque élément à sa sortie correspond à une classe.
Prenons l'exemple d'une image contenant 9 point d'intérêts, illustrée sur la figure 4.4. Nous lui appliquons différentes homographies et nous détectons les points stables comme expliqué précédemment. Le nombre de classes que nous retenons dans l'exemple est 8. En effet, le point labellisé 5 dans l'exemple n'est stable dans aucune image et tous les autres points ont au moins un correspondant. Le CNN est entrainé donc avec tous les patchs des points stables avec les labels correspondants.
Pendant la phase de test, nous fournissons au modèle un patch d'un nouveau point de vue et il lui affecte la classe la plus proche.

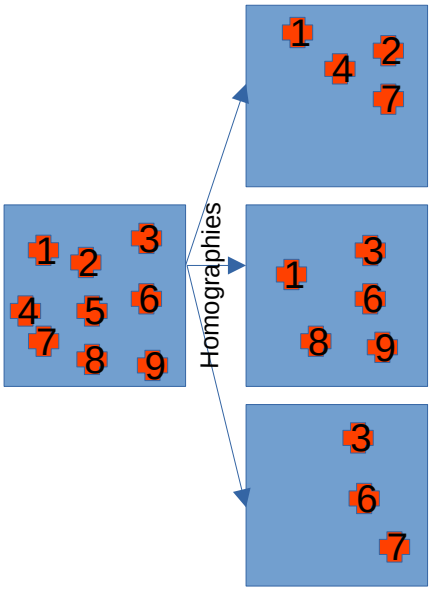

FIGURE 4.4 – Exemple de génération des classes du descripteur.

Mise en correspondance avec l'architecture 2 : Le deuxième modèle de la figure 4.2 utilise une architecture siamoise constituée de deux CNN similaires au modèle précédent, entrainés avec les mêmes paramètres et recevant chacun un patch. La distance euclidienne E_w entre les deux vecteurs descripteurs est calculée. La fonction objective $\mathcal{L}$ de l'équation 4.20 à optimiser durant l'apprentissage, permet de calculer l'erreur de classification des paires de patchs comme similaires ou différents.

$$\mathcal{L} = (1 - Y)\frac{2}{Q}E_w^2 + 2YQ\exp(-\frac{2,77}{Q}E_w), \tag{4.20}$$

avec Y le label de la mise en correspondance, tel que $Y = 1$ pour les patchs différents, sinon $Y = 0$.
Une architecture similaire est utilisée dans [CHL05] pour l'identification de

personnes à partir d'images de visage.
La vérité terrain utilisée pour l'évaluation utilise les homographies de la dataset dans [MS05].
L'utilisation d'un modèle pour faire l'apprentissage directement sur l'image à traiter permet d'améliorer les performances, d'ailleurs, nous trouvons une similarité avec la méthode ASIFT. L'inconvénient est que le nombre de points d'intérêt varie d'une image à l'autre, alors que la taille de sortie du CNN n'est pas ajustable directement. L'architecture 2 à l'avantage d'offrir un descripteur de taille plus réduite de 128. L'inconvénient des deux méthodes est la lenteur de l'apprentissage et du traitement.

4 Descripteurs CNN et métriques d'évaluation

Dans [FDB14], un CNN est utilisé pour la description des points d'intérêt à partir des sorties des couches cachées d'un CNN entrainé pour la classification. Deux modèles sont entrainés par apprentissage supervisé sur la dataset ImageNet [DDS+09] et non supervisés (images sans labels). L'évaluation est réalisée sur la dataset [MTS+05] et sur une nouvelle dataset. Les résultats montre que la méthode surpasse les performances de SIFT et que l'apprentissage non supervisé donne de meilleurs résultats que l'apprentissage supervisé.
Le modèle supervisé utilise une architecture CNN à 5 couches, plus deux couches entièrement connectées (fully connected layers ou FC) et une couche de classification "softmax".
Le modèle non supervisé est entrainé avec $N = 16000$ patchs (extraits aléatoirement) de dimension 64×64. N est donc le nombre de classes. Chaque échantillon est augmenté avec 150 transformations d'échelle, de couleur, de contraste et de rotation. Pour éviter le surapprentissage (Overfitting), un modèle peu profond avec 3 couches et un FC est utilisé.
Notons que les deux modèles ont été entrainés pour la classification et que le but initial de l'étude était de vérifier si la sortie des couches cachées peut être utilisée comme descripteur.
La détection des points d'intérêts est d'abord effectuée par l'algorithme MSER présenté dans le chapitre 2. Ensuite, les descripteurs sont extraits à partir d'une couche cachée du modèle. Enfin, la mise en correspondance est réalisée en comparant les distances euclidiennes des descripteurs.
Pour l'évaluation, les patchs détectés sont comparés à la vérité terrain et sont classés comme vrai positif/négatif (TP) s'ils ont un IOU (Intersection Over Union) des ellipses MSER d'au moins $0,6$, sinon ils sont classés comme faux positif (FP) :

$$IOU = \frac{Aire\ d'intersection}{Aire\ d'union} \tag{4.21}$$

Notons que cette métrique est généralement utilisée pour les problèmes de localisation comme le montre la figure 4.5.

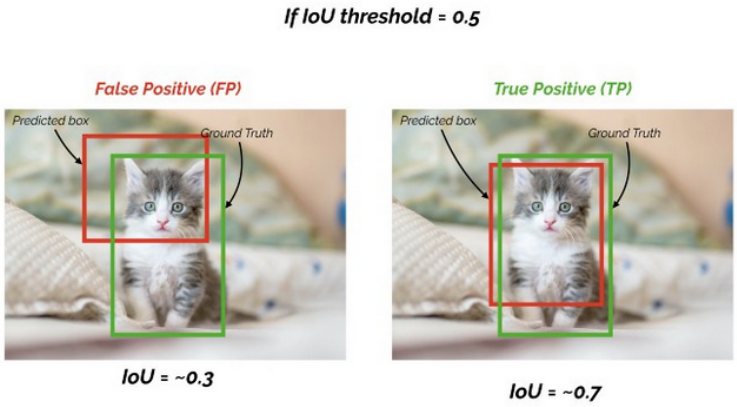

FIGURE 4.5 – Illustration IOU.

L'IOU nous permet donc de calculer le nombre de TP et FP en fonction du seuil choisi.
Nous pouvons donc déduire les valeurs de précision et de rappel (Precision et Recall) à partir de TP, FP et FN (faux négatif) :

$$Precision = \frac{TP}{TP + FP} \tag{4.22}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.23}$$

Métrique mAP (Mean Average Precision) :
La métrique d'évaluation AP (Mean Average Precision) correspond à l'air en dessous de la courbe $Precision - Recall$. Nous déduisons ensuite mAP (Mean Average Precision) en calculant la moyenne des AP (sur toutes les classes et/ou les seuils IOU).
Ces métriques sont les plus utilisés pour l'évaluation des méthodes de mise en correspondance.
Plusieurs couches cachées ont été évaluées avec cette métrique pour le calcul des descripteurs. Les résultats d'évaluation montrent que dans le cas d'apprentissage non supervisé, les couches supérieures donnent de meilleurs résultats. Dans le cas d'apprentissage supervisé, les résultats sont similaires Quelle que soit la couche utilisée pour l'extraction des descripteurs, par contre, la taille des patchs en entrée du CNN à plus d'influence sur les résultats. Les patchs de tailles supérieures sont préférables dans le cas des CNN.
Les CNNs donnent de meilleurs résultats de mise en correspondance que SIFT mais sont plus lents.

5 CNN de mise en correspondance par fonction de similarité

Dans [ZK15], un CNN est entrainé pour l'apprentissage d'une fonction de similarité à partir de paires de patchs. Une dataset est labellisé en paires similaires et différentes.
4 architectures sont proposées (voir les figures 4.6 et 4.7).

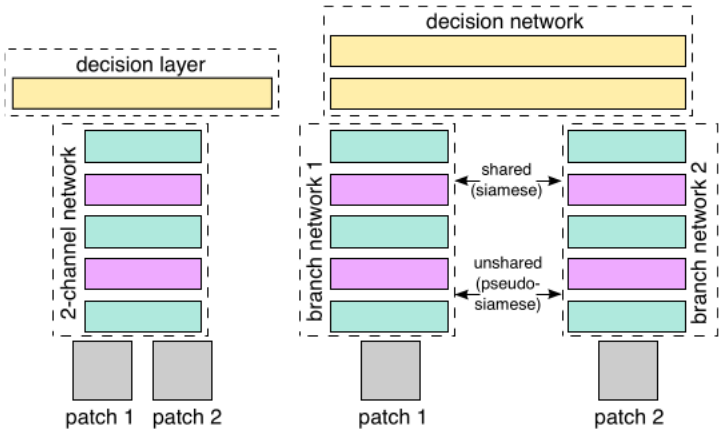

FIGURE 4.6 – 3 architectures CNN de mise en correspondance : simple, siamois avec branches partagées et siamois sans partage des branches. Couche verte : Conv + ReLU ; Couche violette : max-pooling ; Couche jaune : FC (avec ReLU entre deux FC).

Le premier modèle simple utilise un seul CNN : les deux patchs sont considérés comme une seule image à deux canaux.
Dans le deuxième modèle siamois, deux CNNs avec les mêmes poids sont appliqués aux deux patchs (les deux CNN ont des branches partagées). Les deux sorties sont concaténées en un seul vecteur de dimension 512 en entrée de la couche FC. La sortie des couches cachées correspond au descripteur et la dernière sortie à la mesure de similarité.
Le troisième modèle pseudo-siamois est similaire au précédent, à la différence que les deux CNNs ne partagent pas les branches, ce qui signifie que les poids des 2 CNNs sont différents (plus de paramètres).
Le choix de l'architecture dépend de la vitesse de calcul et de la précision.
Pour la mise en correspondance, la couche FC peut être remplacée par une distance L_2.
D'autres modèles peuvent être combinés avec ces architectures pour créer des modèles très profonds [SZ14] et plus performants. L'intérêt est de remplacer les couches avec des grandes tailles de filtres, avec plusieurs filtres 3×3 séparés par des ReLU. Cela permet d'augmenter la non-linéarité de la frontière de décision (Decision Boundary).

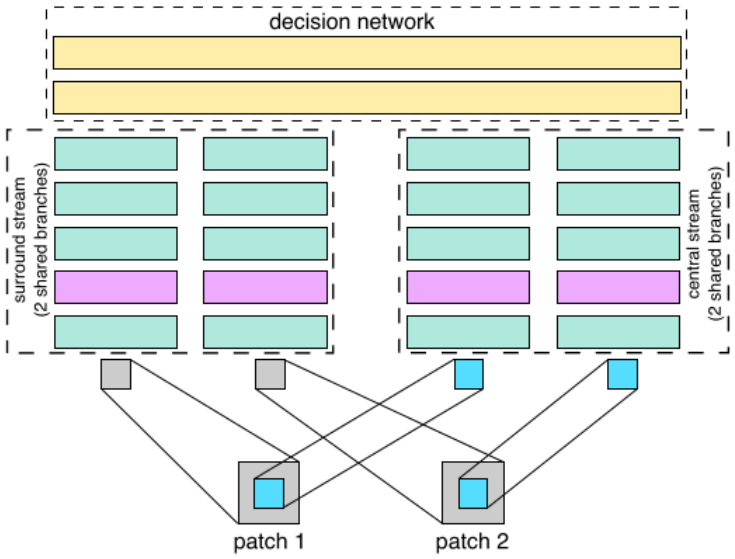

FIGURE 4.7 – Architecture "Surround-Central" pour un traitement multi-résolution.

Le quatrième modèle "Central-Surround", illustré sur la figure 4.7, permet d'effectuer une mise en correspondance multi-échelles. Deux modèles siamois sont utilisés. Le modèle "Surround" reçoit les patchs sous-échantillonnés à une résolution inférieure (de 64×64 à 32×32) et permet donc un traitement basse résolution. Le modèle "Central" reçoit la partie centrale 32×32 du patch originale et permet donc un traitement à plus haute résolution.
La méthode Spatial Pyramid Pooling (SPP) peut être utilisée pour éviter de redimensionner le patch. Cela consiste à ajouter à la sortie des CNNs une couche max-pooling de dimension proportionnelle au patch, ce qui permet de garder toutes les informations du patch.
L'apprentissage des différents modèles est effectué en optimisant la fonction objective suivante :

$$min_w \frac{\lambda}{2}||w||_2 + \sum_{i=1}^{N} max(0, 1 - y_i o_i^{net}) \tag{4.24}$$

avec w les poids du modèle, o_i^{net} la prédiction du i-ème échantillons et $y_i = 1$ le label des patchs similaire, sinon $y_i = -1$ pour les patchs différents.
Les résultats de l'évaluation montrent de meilleures performances que l'optimisation convexe, SIFT et DAISY.

6 TFeat : Apprentissage CNN par triplets de patchs

Dans le modèle TFeat [BRPM16], des échantillons de triplet de patchs sont utilisés au lieu de paires de patch pour l'apprentissage de descripteurs par CNN.
Dans le cas de paire de patch, nous utilisons des échantillons de la forme $\{x_1, x_2, l\}$ avec $l = \{1, -1\}$ le label (similaires ou différents).
Pour un patch x de dimension $m \times n$, nous cherchons un descripteur $f(x)$ de dimension D, tel que la distance $f(x_1) - f(x_2)$ est faible si les deux patchs sont similaires et une distance importante sinon. Cela est réalisé par l'utilisation d'une fonction objective "Contrastive Loss" :

$$\mathcal{L} = \begin{cases} ||f(x_1) - f(x_2)||_2 & , si \quad l = 1 \\ max(0, \mu - ||f(x_1) - f(x_2)||_2) & , si \quad l = -1 \end{cases} . \tag{4.25}$$

avec μ la marge.
$\mathcal{L}$ pénalise les paires positives qui sont séparées par une grande distance ainsi que les paires négatives très proches (distance inférieure à μ).
L'inconvénient de l'utilisation des paires est que la plupart des paires négatives ne contribuent pas à la mise à jour du gradient pendant l'optimisation, étant donné que la distance est le plus souvent supérieure à μ (patchs aléatoires).
Une solution [SSTF+15] est d'identifier les paires "Hard Negative" à partir de leurs distances (proches) pour les utiliser majoritairement pendant l'apprentissage. L'inconvénient de cette méthode est qu'elle est gourmande en temps de traitement.
Dans le cas de triplets, nous utilisons des ensembles d'échantillons $\{a, p, n\}$ avec a le patch de référence (anchor), p un patch similaire à a (positif) et n un patch différent de a (négatif). Nous notons :

$$\delta_+ = ||f(a) - f(p)||_2 \tag{4.26}$$

et

$$\delta_- = ||f(a) - f(n)||_2 \tag{4.27}$$

Nous utilisons la fonction objective :

$$\lambda(\delta_+, \delta_-) = max(0, \mu + \delta_+ - \delta_-) \tag{4.28}$$

Ce qui est équivalent à ajuster les poids d'un CNN en optimisant la fonction objective afin d'obtenir :

$$\delta_- > \delta_+ + \mu \tag{4.29}$$

Ce qui se traduit par une distance δ_- entre les paires négatives plus grande, avec une marge μ, que la distance δ_+ entre les paires positives comme le montre la figure 4.8. Une stratégie pour utiliser les "Hard Negative" est proposée

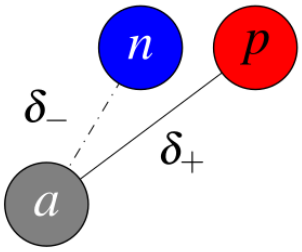

FIGURE 4.8 – Maximisation de la distance entre a et n et minimisation de la distance entre a et p.

afin d'augmenter aussi la distance entre p et n. Pour cela, nous calculons la distance :

$$\delta'_- = ||f(p) - f(n)||_2 \tag{4.30}$$

Nous définissons le "Hard Negative" des triplets par :

$$\delta_* = min(\delta_-, \delta'_-) \tag{4.31}$$

Si $\delta_* = \delta'_-$, nous échangeons a et p pour que p devienne la référence et a le patch positif.
Cela permet de s'assurer que le "Hard Negative" des triplets sont utilisés pour la rétropropagation du gradient.
Nous obtenons donc une nouvelle fonction objective :

$$\lambda(\delta_+, \delta'_*) = max(0, \mu + \delta_+ - \delta_*) \tag{4.32}$$

Cette méthode donne de meilleurs résultats que celles utilisant des paires de patch. Le temps de traitement sur GPU est de $10\mu s$ par patch. Le code est disponible sur [TFe].

7 L2-Net

Le modèle L2-Net [TFW17] utilise une architecture avec uniquement des couches convolutives suivies de couches "Batch Normalisation" et une couche de normalisation LRN (Local Response Normalization) à la fin. L'architecture est illustrée dans la figure 4.9, ainsi qu'une variante "Central Surround".

Pour l'apprentissage, les datasets Brown [BHW10] et HPatches [BLVM17] sont utilisées.
En pratique, lorsque nous effectuons la mise en correspondance, nous nous retrouvons avec beaucoup plus de patchs différents que de patchs similaires. À cause du grand nombre des patchs négatifs, il est impossible de tous les utiliser pour l'apprentissage. Les méthodes précédentes utilisent des classes positive et négative avec des nombres d'échantillons égaux, alors que pour être plus proche

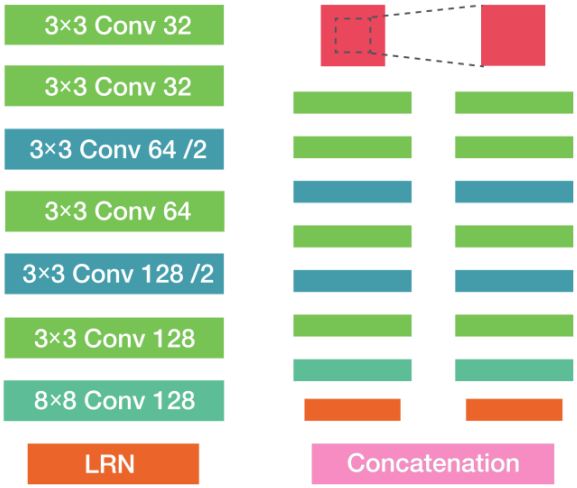

FIGURE 4.9 – Architecture L2-Net et variante "Central Surround".

de la réalité, il faut plus d'échantillons négatifs.
La stratégie permettant de sélectionner les patchs négatifs pertinents et en plus grande quantité est la suivante :
1) Nous considérons un nombre P de points 3D de la scène indexés i et les patchs x_i^j correspondants dans les images indexés j.
2) De manière itérative, nous sélectionnons séquentiellement p_1 points dans l'ensemble des P points, et aléatoirement p_2 points des $P - p_1$ restants. Le choix aléatoire augmente les chances du CNN d'apprendre de nouvelles choses et d'améliorer ce qu'il a déjà appris.
3) Pour chaque point i nous choisissons aléatoirement une paire (x_i^1, x_i^2) est dans l'ensemble des $p = p_1 + p_2$ points sélectionnés. Nous obtenons ainsi un batch d'apprentissage de p paires de patchs :

$$X = \{x_1^1, x_1^2, ..., x_i^1, x_i^2, ..., x_p^1, x_p^2\} \tag{4.33}$$

4) Nous obtenons un batch de descripteurs à la sortie du CNN L2-Net de dimension q chacun :

$$Y = \{y_1^1, y_1^2, ..., y_i^1, y_i^2, ..., y_p^1, y_p^2\} \tag{4.34}$$

5) Nous calculons une matrice D de distance :

$$D = \sqrt{2(1 - Y_1^T Y_2)} \tag{4.35}$$

avec :

$$Y_s = [y_1^s, ..., y_p^s] \tag{4.36}$$

En effet, les éléments d_{ij} de D correspondent à la distance entre les descripteurs de deux points différents :

$$d_{ij} = ||y_i^2 - y_j^1||_2 \tag{4.37}$$

Par conséquent, D contient les distances de p^2 paires. Les paires positives correspondent aux éléments de la diagonale de D et les autres $p^2 - p$ paires sont négatives.

La fonction objective utilisée par L2-Net est constituée de 3 termes : un terme de similarité pour séparer les patchs positifs et négatifs, un terme de "compacité" (Compacteness) qui permet de décorréler les dimensions du descripteur et un terme intermédiaire pour prendre en compte les couches cachées (intermédiaires) du CNN.

Erreur de similarité :
La distance d_{kk} doit être la plus petite distance dans la k-ième ligne et dans la k-ième colonne :

$$d_{kk} = min(d_{ik}, d_{kj}) \ \forall i, j, k \in [1, p] \tag{4.38}$$

L'erreur de similarité s'écrit :

$$E_1 = -\frac{1}{2}(\sum_i \log(s^c_{ii}) + \sum_i \log(s^r_{ii})) \tag{4.39}$$

avec s^c_{ii} et s^r_{ii} sont respectivement les éléments des matrices de similarité des colonnes et des lignes :

$$s^c_{ij} = \frac{\exp(2 - d_{ij})}{\sum_m \exp(2 - d_{mj})} \tag{4.40}$$

$$s^r_{ij} = \frac{\exp(2 - d_{ij})}{\sum_n \exp(2 - d_{jn})} \tag{4.41}$$

L'erreur E_1 permet donc de séparer les patchs négatifs et de rapprocher les patchs positifs dans l'espace euclidien.

Erreur de compacité :
La corrélation des dimensions des descripteurs implique un surapprentissage. La compacité correspond au fait d'avoir moins de redondance dans le descripteur et que chaque dimension contient le maximum d'information. Par conséquent, un descripteur de dimension réduite peut donner un résultat similaire.
Nous utilisons les descripteurs y^s_1 de l'équation 4.36 afin de définir la matrice de corrélation $R_s = [r^s_{ij}]$:

$$r^s_{ij} = \frac{(y^s_i - \hat{y^s_i})(y^s_j - \hat{y^s_j})}{\sqrt{(y^s_i - \hat{y^s_i})(y^s_i - \hat{y^s_i})}\sqrt{(y^s_j - \hat{y^s_j})(y^s_j - \hat{y^s_j})}} \tag{4.42}$$

avec $\hat{y^s_i}$ la moyenne du i-ème terme (vecteur) de Y_s.
L'erreur de compacité est :

$$E_2 = \frac{1}{2}(\sum_{i \neq j}(r^1_{ij})^2 + \sum_{i \neq j}(r^2_{ij})^2) \tag{4.43}$$

La minimisation de cette erreur revient à chercher à obtenir des éléments nuls hors diagonale.

Erreur intermédiaire :
La sortie des couches intermédiaires du CNN doivent aussi êtres proches pour les patchs positifs et différentes pour les patchs négatifs. L'erreur est calculée avec l'équation 4.39 de l'erreur de similarité E_1, en remplaçant dans la matrice des distances la sortie du CNN par les sorties des couches cachées.

8 HardNet

Le modèle [MMRM17] est inspiré de SIFT. Le principe est de trouver le plus proche voisin NN (Nearest Neighbor) du patch et de le comparer au second NN par un ratio de distance. L'architecture utilisée est la même que L2-Net, mais sans les termes d'erreurs de compacité et intermédiaire. Le principe illustré sur la figure 4.10 est fondée sur une stratégie de sélection de triplets.

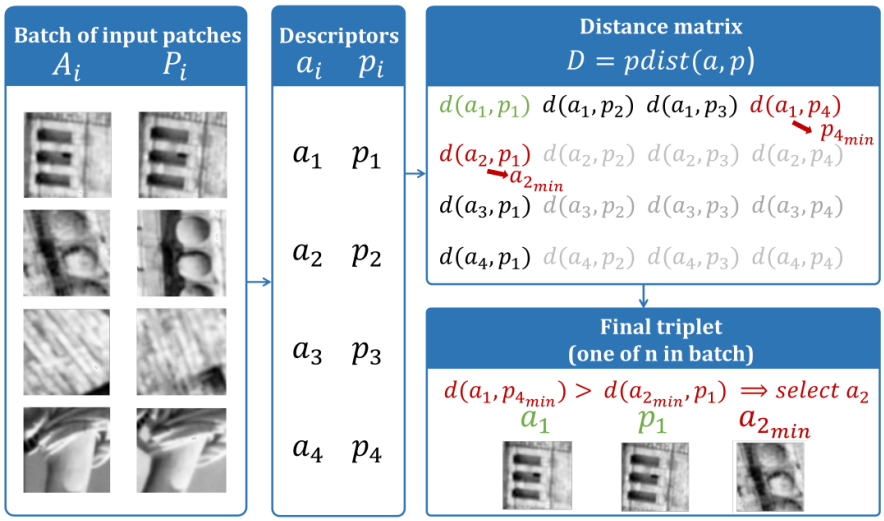

FIGURE 4.10 – Apprentissage de HardNet.

D'abord, un batch de n paires de patchs positifs (A_i, P_i) est traité par le CNN. Il faut éviter qu'il y ait une redondance de paires correspondantes au même point.
Les descripteurs (a_i, p_i) à la sortie du CNN sont utilisés pour le calcul de la matrice de distance D :

$$D = [dist(a_i, p_j)] \tag{4.44}$$

avec

$$dist(a_i, p_j) = \sqrt{2 - 2a_i p_j} \tag{4.45}$$

Pour chaque patch a_i, le second NN $p_{j_{min}}$ est sélectionné (sur les colonnes de D) :

$$j_{min} = \underset{j=1..n, j\neq i}{\arg\min}\, d(a_i, p_j) \tag{4.46}$$

Pour chaque patch p_i, le second NN $a_{k_{min}}$ est sélectionné (sur les lignes de D) :

$$k_{min} = \underset{k=1..n, k\neq i}{\arg\min}\, d(a_k, p_i) \tag{4.47}$$

Dans la figure 4.10, les seconds NN sont affichés en rouge.
Ensuite, la sélection des Triplets est effectuée comme suit :

$$Triplet = \begin{cases} (a_i, p_i, p_{j_{min}}) \ , & si\ d(a_i, p_{j_{min}}) < d(a_{k_{min}}, p_i) \\ (a_i, p_i, a_{k_{min}}) \ , & sinon \end{cases} \tag{4.48}$$

Nous cherchons donc les paramètres du CNN qui minimisent la fonction objective :

$$\mathcal{L} = \frac{1}{n} \sum_{i=1..n} max(0, 1 + d(a_i, p_i) - min(d(a_i, p_{j_{min}}), d(a_{k_{min}}, p_i)) \tag{4.49}$$

Cela permet de minimiser la distance entre les descripteurs de patchs positifs ainsi qu'avec le second NN descripteur.
Les résultats obtenus sont meilleurs que SIFT ou les méthodes de régularisation complexes telles que L2-Net.
Le code est disponible dans [Har].

9 SOSNet

L'idée de SOSNet [TYF+19] est que les différentes paires de patchs positifs doivent avoir des distances similaires dans l'espace du descripteur. Un terme de régularisation de similarité de second ordre est utilisé.
Considérons N paires de patchs positifs. Le modèle L2-Net est utilisé pour extraire les descripteurs $\{x_i, x_i^+\}$ de chaque paire positive i.

Similarité de premier ordre : L'objectif est de réduire la distance entre les paires positives et d'augmenter la distance entre les paires négatives.

$$\mathcal{L}_{FOS} = \frac{1}{N} \sum_{i=1}^{N} max(0, t + d_i^+ - d_i^-)^2 \tag{4.50}$$

avec t est la marge et d_i^+ la distance entre les paires positives :

$$d_i^+ = ||x_i, x_i^+||_2 \tag{4.51}$$

d_i^- est la distance entre les paires négatives, qui sont sélectionnées de façon à former un triplet "Hard Negative" :

$$d_i^- = min_{j,j\neq i}(||x_i - x_j||_2, ||x_i - x_j^+||_2, ||x_i^+ - x_j||_2, ||x_i^+ - x_j^+||_2) \tag{4.52}$$

Notons que contrairement à HardNet la fonction objective est quadratique. Cela permet d'améliorer les performances.

Similarité de second ordre : La mesure SOS (Second Order Similarity) d'une paire positive x_i, x_i^+ est définie par :

$$d^{(2)}(x_i, x_i^+) = \sqrt{\sum_{j\neq i}^{N}(||x_i - x_j||_2 - ||x_i^+ - x_j^+||_2)^2} \tag{4.53}$$

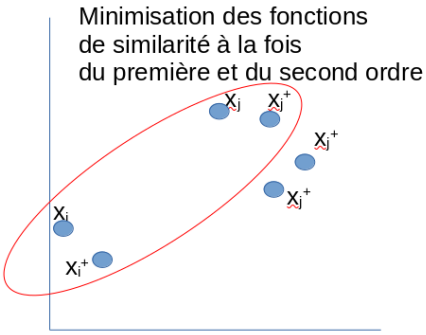

FIGURE 4.11 – Illustration SOS : le but est d'entrainer le modèle de façon à réduire à la fois la distance entre les paires positives d'augmenter la distance entre les paires négatives (premier ordre) et d'avoir des distances similaires entre paires positives des différents échantillons similaires (second ordre).

La SOS mesure la similarité entre x_i et x_i^+ du point de vue des autres paires positives x_j et x_j^+ comme le montre la figure 4.11. Le but est que les distances entre les paires positives soient similaires pour tous les échantillons.
La fonction objective SOS est :

$$\mathcal{L}_{SOS} = \frac{1}{N}\sum_{i=1}^{N} d^{(2)}(x_i, x_i^+) \tag{4.54}$$

La fonction objective du modèle est donc :

$$\mathcal{L} = \mathcal{L}_{FOS} + \mathcal{L}_{SOS} \tag{4.55}$$

Les résultats d'évaluation montrent que le modèle SOSNet donne de meilleurs résultats que TFeat, L2-Net et HardNet.

10 LIFT : approche moderne "End-to-End"

Le modèle LIFT [YTLF16] est un réseau profond permettant l'apprentissage de toute la pipeline détection, estimation de l'orientation et description des points d'intérêt. Cela se fait manière "End-to-End" afin de préserver la différentiabilité, de façon à ce que les différentes étapes de la pipeline soient optimisées ensemble et non de manière indépendante comme c'est le cas des méthodes précédentes.
Chaque étape du modèle LIFT, illustré sur la figure 4.12, est modélisée par un CNN. Les différentes étapes sont reliées par des transformations spatiales.
La suppression des non maxima locaux (similaire au détecteur de Harris) de la carte des points d'intérêt est remplacée par une fonction Softmax afin de préserver la différentiabilité de tout le réseau et permettre un apprentissage par rétropropagation (dérivée de la fonction d'activation). Le patch est ensuite rogné pour s'assurer qu'il ne contient qu'un seul point d'intérêt. Le patch est enfin réorienté avant le calcul du descripteur. L'apprentissage est réalisé à l'aide d'un réseau siamois illustré sur la figure 4.13 (voir plus loin).

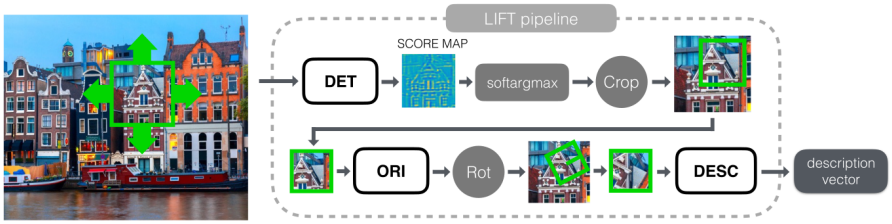

FIGURE 4.12 – Architecture LIFT composé de 3 modèles CNN de détection, d'orientation et de description.

Les datasets Piccadilly et Roman Forum [Data, WS14] sont utilisées. D'abord, une reconstruction 3D des images est réalisée par un algorithme SfM (Structure From Motion) fondé sur SIFT [Wu13].
Les points 3D similaires (et uniques) dans différentes images, sont utilisés comme patchs positifs et les points 3D différents sont utilisés comme patchs négatifs. Les points 3D non reconstruits sont utilisés comme patchs ne correspondants pas à des points d'intérêt pour l'apprentissage du détecteur uniquement.
L'échelle SIFT est utilisée comme échelle des patchs pour obtenir un modèle invariant à l'échelle.
Les images utilisées contiennent des scènes acquises de différents points de vue et de différents éclairages pour obtenir un modèle invariant par transformations géométriques et photométriques.
Le modèle siamois illustré sur la figure 4.13 est utilisé pour l'apprentissage des poids du modèle.
Les patchs p^1 et p^2 sont des patchs similaires, p^3 un patch différent et p^4 un patch ne correspondant pas à un point d'intérêt pour l'apprentissage du

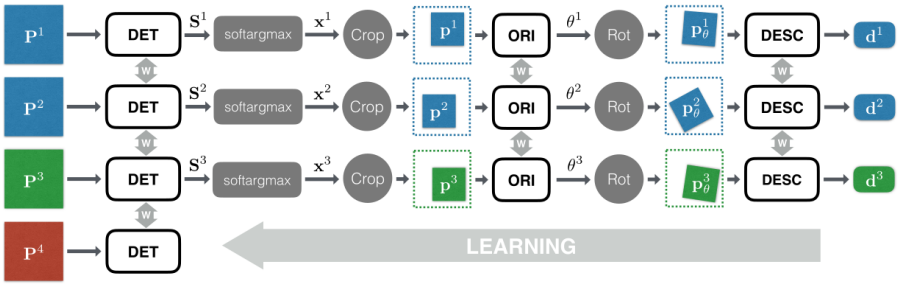

FIGURE 4.13 – Apprentissage de LIFT.

détecteur uniquement.
Il n'est pas possible d'effectuer l'apprentissage de toutes les parties du réseau en même temps, car chaque composante va essayer d'optimiser les paramètres différemment puisqu'ils n'utilisent pas la même fonction objective (pas le même but).
Nous commençons donc par l'apprentissage du descripteur seul. Ensuite, il est combiné avec l'estimateur d'orientation pour l'apprentissage de ce dernier. Enfin le résultat des deux apprentissages est utilisé pour entrainer le détecteur.
Les CNN utilisés pour la détection, l'orientation et la description sont similaires à ceux dans [VYFL15, YVFL16, SSTF+15].
Le descripteur est entrainé avec une fonction objective, à l'aide d'une distance L_2 entre les paires de patchs, similaire à celles des modèles précédents :

$$\mathcal{L}_{desc}(p^k, p^l) = \begin{cases} ||h(p^k) - h(p^l)||_2 & , si \quad (p^k, p^l) \; est \; une \; paire \; positive \\ max(0, C - ||h(p^k) - h(p^l)||_2) & , si \quad (p^k, p^l) \; est \; une \; paire \; negative \end{cases} \tag{4.56}$$

avec h la fonction de prédiction du CNN de description et $C = 4$ la marge.
L'estimateur d'orientation est optimisé avec une fonction objective qui minimise la distance entre les descripteurs de patchs positifs :

$$\mathcal{L}_{orient}(p^1, p^2) = ||h(g(p^1)) - h(g(p^2))||_2 \tag{4.57}$$

avec g la fonction de ré-orientation du patch estimé par la fonction de prédiction du CNN d'orientation ($g(p)$ est la sortie du module "ORI" dans la figure 4.13).
Le modèle de détection est entrainé de façon à optimiser deux termes dans la fonction objective :

$$\mathcal{L}_{det} = \gamma \mathcal{L}_{class}(p^1, p^2, p^3, p^4) + \mathcal{L}_{paire}(p^1, p^2) \tag{4.58}$$

Le premier terme consiste à maximiser le score de classification (sortie du détecteur) pour les paires différentes :

$$\mathcal{L}_{class}(p^1, p^2, p^3, p^4) = \sum_{i=1}^{4} \alpha_i max(0, 1 - softmax(f(p^i))y_i)^2 \tag{4.59}$$

avec f la fonction de prédiction du détecteur, y_i le label (1 pour les points d'intérêt p^1, p^2 et p^3 et -1 pour les non points d'intérêt p^4). Le paramètre de pondération est $\alpha_i = 3/6$ pour les points d'intérêt et $\alpha_i = 1/6$ pour les non points d'intérêt.
Le deuxième terme consiste à minimiser la distance entre les descripteurs (sortie descripteur) des patchs similaires :

$$\mathcal{L}_{paire} = ||h(g(p^1, softmax(f(p^1)))) - h(g(p^2, softmax(f(p^2))))||_2 \quad (4.60)$$

Notons le caractère "End-to-End" de toute la pipeline dans ce dernier terme : f prédiction du détecteur, g prédiction de l'orientation (et ré-orientation), et h prédiction du descripteur.
Étant donné que le modèle entrainé prend en entrée des patchs, afin de l'appliquer sur une image entière, il faut parcourir l'image avec une fenêtre glissante et faire appel au modèle pour chaque patch. Cette méthode est très gourmande en temps de calcul. La solution illustrée sur la figure 4.14 consiste à séparer la détection des autres modules. En effet, la détection permet d'obtenir une carte de points d'intérêt et peut s'appliquer sur une image entière. Ensuite, seuls les patchs détectés sont utilisés dans un batch pour l'estimation de l'orientation et pour la description.
Notons aussi sur la figure 4.14 que la fonction Softmax peut être remplacée par une suppression des non-maxima locaux (NMS) et que la détection peut s'effectuer en multi-échelles en réduisant la résolution de l'image en entrée.

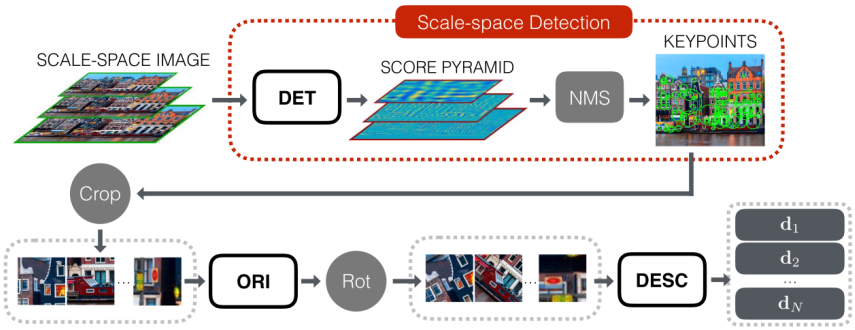

FIGURE 4.14 – Prédiction par LIFT d'une image entière et en multi-échelles.

Les résultats d'évaluation de la mise en correspondance montrent que LIFT permet de meilleures performances que SIFT, ORB, DAISY et d'autres méthodes traditionnelles.

11 Conclusion

Nous avons présenté les nouveaux algorithmes de détection, de description et de mise en correspondance des points d'intérêt, qui ont émergé durant cette

dernière décennie avec le "Big Bang" du Deep Learning. Nous avons découvert les différents modèles utilisés et leur évolution, en partant de méthodes de réduction de dimensionnalité et d'optimisation convexe, jusqu'aux modèles "End to End", en passant par diverses architectures CNN. Ces méthodes nous ont beaucoup inspirés pour proposer de nouveaux modèles et de nouvelles datasets, que nous allons présenter dans le chapitre 7. Dans le chapitre suivant, nous allons discuter d'un autre aspect des effets visuels : l'informatique graphique.

Chapitre 5

Infographie, rendu 3D et détection de la pose humaine

Sommaire

L'informatique graphique (infographie, Computer Graphics ou CG), images générées sur ordinateur (CGI) ou images de synthèse sont des termes utilisés interchangeablement pour désigner des images 3D créées sur un ordinateur. Aujourd'hui, nous pouvons difficilement imaginer un monde sans ce type d'images qui sont devenues omniprésentes dans les médias et nos moyens de communication quotidiens, mais aussi d'une importance cruciale dans plusieurs domaines, allant de l'industrie comme l'aviation ou l'automobile, à la visualisation des calculs scientifiques ou des images médicales, en passant par la réalité virtuelle, les jeux vidéo, les films d'animation et les effets visuels.
Pendant longtemps, la génération de ces images passait nécessairement par l'utilisation d'outils de modélisation. Mais grâce aux progrès de l'apprentissage profond, une nouvelle discipline appelée "neural rendering" ou rendu neuronal [TFT+20] permet de générer automatiquement des images 3D photoréalistes avec un certain degré de contrôle sur les paramètres géométriques et photométriques. Cependant, comme nous allons le découvrir, ces techniques ne remplacent pas encore complètement les méthodes traditionnelles de rastérisation [GC20] et de ray tracing [HAM19] qui, à leur tour, ont profité des avancées du calcul parallèle des nouvelles cartes graphiques (GPU) pour améliorer la qualité du rendu et le temps de calcul. La figure 5.1 montre des exemples d'images obtenues par les différentes techniques.

Le principal défi de l'infographie est de créer des images photo-réalistes et à faible coût en termes de temps calcul [PJH16]. Pour cela, les différentes techniques tentent d'approcher la solution de l'équation de transport, qui modélise la radiance réfléchie par un point d'un objet de la scène vers l'observateur. La modélisation informatique exacte de cette équation reste impossible malgré la puissance de calcul dont nous disposons aujourd'hui en raison du très grand nombre de photons dont il faut simuler l'interaction avec tous les objets de la scène avant d'atteindre la caméra. Toutefois, grâce à des approximations, il est possible d'obtenir des images indiscernables des photographies réelles.
Dans ce qui suit, nous allons présenter la pipeline de rastérisation et la manière dont il approxime la lumière avec des modèles d'ombrage ambiant, diffus et spéculaire pour différents types de matériaux et d'éclairage. Nous découvrirons ensuite le pipeline et l'évolution des techniques de ray tracing, et la manière dont elles simulent les effets d'éclairage pour augmenter le niveau de photoréalisme. Concernant les méthodes fondées sur l'apprentissage profond, nous présenterons les réseaux adverses (ou antagonistes) génératifs (GAN), qui ont révolutionné les images de synthèse, et leur utilisation avec les auto-encodeurs variationnels (VAE) dans le domaine du rendu neuronal afin de générer des modèles 3D et de faire le rendu de manière photo-réaliste à partir d'un nouveau point de vue ou d'une nouvelle position d'éclairage. Une comparaison est faite selon différents critères de photo-réalisme, de contrôlabilité, de temps de calcul et de travail, d'accessibilité, de qualité des détails et d'applications. Nous allons aussi discuter de l'aspect animation et particulièrement de l'estimation de la pose humaine par caméra 3D.

FIGURE 5.1 – Rendu d'images de synthèse par rastérisation (première ligne), par Ray Tracing (deuxième ligne) et par "Neural Rendering" pour la synthèse de nouvelle vue (troisième ligne) et pour la synthèse de nouvelle position d'éclairage (quatrième ligne).

1 Équation de transport

Le rendu est le processus de projection d'un modèle 3D généré par ordinateur sur un écran 2D. La couleur de chaque pixel de l'image rendue en 2D dépend de la position, de la direction et du type d'éclairage, de la géométrie et des matériaux de la scène, ainsi que de la position et de la direction de l'observateur (caméra). L'équation de transport (ou équation de rendu) permet de calculer pour un point P d'un objet de la scène, sa radiance L_o (couleur du pixel) dans une direction w_o (vers la caméra) :

$$L_o(P, w_o) = \int_S f(P, w_o, w_i) L_i(P, w_i) |\cos(\theta_i)| dw_i, \tag{5.1}$$

avec f la fonction BRDF (Bi-directional Reflectance Distribution Function) décrivant les propriétés de surface de l'objet (matériau), w_i la direction de la lumière incidente, et θ_i l'angle entre la lumière incidente et la normale à la surface. Le terme $cos(\theta_i)$ modélise l'atténuation géométrique en fonction de l'angle.
Pour obtenir un rendu photo-réaliste, il est donc nécessaire d'intégrer les lumières incidentes provenant de toutes les sources S, de toutes les directions, vers chaque point de la scène afin d'estimer sa couleur (radiance).
En pratique, malgré la puissance de calcul dont nous disposons aujourd'hui, il est impossible de simuler sur un ordinateur le mouvement de chaque photon qui part de chaque éclairage et interagit avec plusieurs éléments de la scène avant d'atteindre l'observateur, à cause du très grand nombre de photons générés par l'ensemble des éclairages. Il est toutefois possible de créer des scènes photoréalistes à l'aide de modèles approximatifs.

2 Rastérisation

2.1 Pipeline

La rastérisation est la technique de rendu qui, pour chaque vertex du modèle 3D, visible par la caméra, calcule la couleur de son pixel à l'aide d'un modèle de Shading (Ombrage). La pipeline de rastérisation est illustrée dans la figure 5.2.

Les shaders correspondent aux étapes programmables sur GPU avec un langage spécifique (GLSL pour OpenGL ou HLSL pour DirectX).
Le vertex shader reçoit les données géométriques (position vertex) du CPU et les transforme en position relative à la caméra. Ce shader s'exécute pour chaque vertex une seule fois (souvent en parallèle).
Les shaders de tessellation et de géométrie sont facultatifs.
La tessellation, introduite plus récemment, permet de générer par combinaison de vertex un grand nombre de polygones lorsque nous avons besoin de

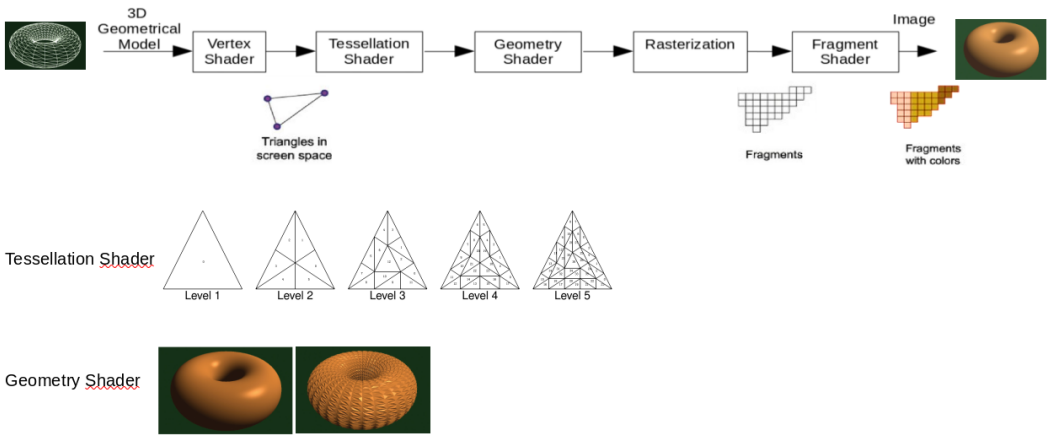

FIGURE 5.2 – Pipeline de Rastérisation

beaucoup de vertices.
Le shader de géométrie permet de manipuler les primitives (vertex, polygones et lignes) de plusieurs façons (étirement, trous, etc.) transformant ainsi des modèles simples en des modèles plus complexes. Une utilisation intéressante est d'ajouter des textures au modèle (bosses, cheveux, etc.).
Le rasterizer transforme les primitives, généralement des triangles, en fragments de pixels en les projetant sur l'image avec un modèle caméra, et permet également d'interpoler toute variable de sortie du vertex shader (position, couleur, etc.).
Le shader fragment calcule la couleur des pixels en fonction de l'éclairage, des matériaux et des textures de la scène, en utilisant un modèle de Shading.
Lors de l'affichage, un algorithme de suppression des surfaces cachées HSR (Hidden Surface Removal), non programmable, permet de n'afficher que les vertex visibles en comparant les distances de chaque vertex par rapport à la position de la caméra et en n'affichant que les plus proches et pas ceux de derrière.
L'animation des objets peut être effectuée en faisant varier les paramètres de l'objet (position, couleur, etc.) dans le temps et en appelant la fonction d'affichage en boucle.
Les objets 3D peuvent être définis dans un espace local (objet) ou un espace monde (quelconque dans la scène) et peuvent être manipulés géométriquement par de simples transformations matricielles. La projection des objets se fait soit selon une projection en perspective pour un effet réaliste ou orthographique pour des applications de CAO par exemple. Les matrices de transformations de l'objet et de la caméra sont envoyées dans la pipeline avec les vertex.

2.2 Mappage de texture

Il s'agit du moyen le plus simple d'ajouter du réalisme. Il est souvent utilisé dans les jeux vidéo et les animations pour peindre les visages, la peau et les vêtements. Les textures à appliquer aux modèles 3D sont stockées dans des images. Nous appelons texels les pixels des textures pour les différencier des pixels de l'image rendue. Pour chaque vertex (x, y, z) du modèle 3D, nous attribuons les coordonnées texels correspondantes (s, t) de la couleur à afficher. Les coordonnées texels peuvent être assignées manuellement dans le cas de formes simples (cube, pyramide), ou algorithmiquement pour les géométries courbées (sphère, tore), ou encore dans le cas de modèles complexes (humain, objets) avec les outils "UV-Mapping" de logiciels dédiés (Maya, Blender). Le mappage de texture produit souvent des artefacts de flou, de distorsion et d'aliasing, en raison de la différence de résolution et de rapport d'aspect entre le modèle et la texture [SA20]. Le mipmapping utilise des textures à différentes résolutions pour appliquer la plus proche à chaque région de l'image. Le flou et la distorsion sont corrigés à l'aide de filtres anisotropes et d'algorithmes de correction de perspective.

2.3 Lumière et matériaux

Pour donner plus de réalisme au rendu, la rastérisation utilise des modèles de Shading (ombrage ou réflexion) qui calculent la couleur des pixels en fonction de l'éclairage et des matériaux des objets. Le modèle de shading le plus couramment utilisé pour approximer la lumière (équation de transport) est le modèle ambiant, diffus et spéculaire (voir [MS18] et [HH20b]). Un reflet ambiant affecte tous les éléments de la scène de manière égale. Une réflexion diffuse reflète la lumière dans toutes les directions en fonction de l'angle d'incidence. Une réflexion spéculaire donne à l'objet une apparence brillante en fonction de l'angle de l'observateur et de l'angle d'incidence. L'éclairage et le matériau ont tous les deux des propriétés qui reflètent ces trois caractéristiques qui doivent être spécifiées. Le matériau possède en outre une caractéristique de brillance.

Du côté de l'éclairage, il existe quatre types de sources lumineuses :

— Ambiance globale : éclairage dont la source et la direction sont indéterminées. Sa contribution est égale en tout point de la scène. Il correspond en effet à la lumière qui a rebondi plusieurs fois à travers les objets de la scène. Il est modélisé par une valeur RGBA.

— Éclairage directionnel (ou distant) : éclairage dont la position est également indéterminée, mais qui possède une direction. Il correspond à un éclairage très distant (par exemple le soleil), dont les rayons sont considérés comme parallèles. Il est modélisé par quatre vecteurs : les valeurs RGBA ambiantes, diffuses et spéculaires, et le vecteur direction.

— Positionnel : éclairage dont la position est connue, mais dont la direction

va dépendre de chaque vertex de la scène. Il correspond aux sources proches (lampes, bougies). Il est modélisé par les valeurs RGBA ambient, diffuse et spéculaire et le vecteur de position. Nous pouvons également ajouter un facteur d'atténuation pour modéliser la diminution de l'éclairage en fonction de la distance $C_{att} = \frac{1}{k_c + k_l.r + k_q.r}$, avec $k_c \geq 1$, k_l et k_q sont respectivement les coefficients d'atténuation constants, linéaire et quadratique et r la distance.

— La lumière ponctuelle : lumière caractérisée à la fois par une direction et une position. Elle est modélisée par un cône de demi-angle θ compris entre 0 et 90°. Le facteur d'intensité est calculé par $cos^f(\phi)$, avec ϕ l'angle entre la direction D d'illumination et la direction V vers le vertex et f est l'exposant de Falloff (affaiblissement) qui simule la variation de l'illumination le long du cône. Le facteur d'intensité est multiplié par l'intensité de l'illumination pour simuler l'effet de cône. Nous indiquons donc pour la lumière ponctuelle, les valeurs RGBA ambiantes, diffuses et spéculaires, ainsi que la direction, la position, l'affaiblissement f et l'angle θ. L'intensité est nulle lorsque $\phi \geq \theta$.

L'éclairage interagit différemment avec chaque matériau. Nous devons donc spécifier, pour le matériau, les valeurs RGBA ambiantes, diffuses et spéculaires ainsi que la brillance. Certains matériaux qui émettent leur propre lumière peuvent avoir un paramètre d'émissivité.

2.4 Éclairage de la scène

Nous calculons la couleur de chaque pixel par :

$$I = I_{amb} + I_{diff} + I_{spec}, \tag{5.2}$$

où I_{amb} (I_{diff} ou I_{spec}) est la somme des intensités ambiantes (diffuses ou spéculaires) I^s_{amb} (I^s_{diff} ou I^s_{spec}) des différentes sources lumineuses s. Les contributions ambiante, diffuse et spéculaire des sources lumineuses et des matériaux, sont incluses selon les équations 5.3, 5.4 et 5.5. La figure 5.3 illustre les différents paramètres.

$$I^s_{amb} = L^s_{amb} M_{amb}, \tag{5.3}$$

avec L^s_{amb} et M_{amb} les coefficients d'éclairage et de matériaux ambiants (voir section précédente).

$$I^s_{diff} = L^s_{diff} M_{diff} cos(\theta), \tag{5.4}$$

avec L^s_{diff} et M_{diff} les coefficients d'éclairage diffus et de matériau,
θ l'angle entre la normale à la surface N et la lumière incidente L ,
et $cos(\theta) = max((N \bullet L), 0)$.

$$I^s_{spec} = L^s_{spec} M_{spec} max((R \bullet V)^n, 0), \tag{5.5}$$

avec L^s_{spec} et M_{spec} les coefficients spéculaires de l'éclairage et du matériau, R la direction de réflexion, V la direction d'observation, et n le coefficient de brillance qui modélise l'atténuation en fonction de l'angle d'observation.

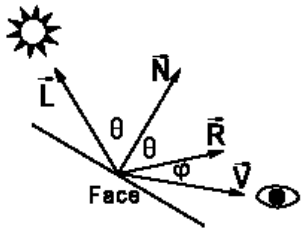

FIGURE 5.3 – Illustration de la réflexion de la lumière.

Le calcul de la normale pour chaque pixel prend du temps. Les techniques de Shading lisse (Gouraud ou Phong) calculent la couleur uniquement pour les pixels correspondant au vertex, et en déduisent par interpolation la couleur des autres pixels.

Il existe plusieurs façons de combiner la couleur I_{tex} de l'image de texture avec le modèle d'ombrage. La façon la plus simple est d'utiliser une moyenne pondérée comme dans l'équation 5.6. Il existe plusieurs variantes en fonction de l'effet souhaité.

$$I = 0,5 \cdot I_{tex} + 0,5 \cdot (I_{amb} + I_{diff} + I_{spec}) \tag{5.6}$$

2.5 Ombre

L'ajout d'ombre est important pour la perception de la profondeur et donc pour un rendu photoréaliste (voir [LP09] et [KSS16]). Les ombres dures (Hard Shadows) sont celles dont les contours sont nets. L'approche par ombre projective (Projective Shadows) consiste à utiliser une matrice de projection pour projeter l'ombre d'un objet sur un plan. L'approche par volumes d'ombre (Volume Shadows) consistent à identifier et à réduire l'intensité de la partie de la scène où l'objet projette son ombre, cela génère moins d'artefacts, mais prend du temps (supporté sur certains GPU). La cartographie des ombres (Shadow Mapping) utilise l'idée que les objets qui ne sont pas atteints par la lumière sont dans l'ombre, elle est implémentée en utilisant l'algorithme de suppression des surfaces cachées HSR, présenté précédemment, mais en positionnant la caméra à la position d'éclairage. Les ombres douces (Soft Shadows) sont des ombres aux contours flous, elles sont les plus présentes dans la réalité à cause de l'éclairage non ponctuel. Dans cette catégorie, le Percentage closer filtering (PCF) est la technique la plus utilisée, elle consiste à estimer le pourcentage qu'une région est dans l'ombre en fonction de sa distance à l'éclairage.

2.6 Mapping des bosses et des hauteurs

Dans le cas de surfaces très irrégulières, l'effet de réalisme créé par le mapping de texture n'est pertinent que si l'éclairage et la position de la caméra ne varient pas. Dans le cas contraire, il est nécessaire de modéliser géométriquement cette irrégularité, ce qui demande beaucoup de travail avec un outil de modélisation et beaucoup de vertices à traiter pour le rendu [SWJH13]. Le Mapping des bosses (Bump Mapping) consiste à simuler l'effet de l'éclairage sur des surfaces irrégulières, sans modifier la géométrie du modèle, mais en modifiant la normale aux polygones. En effet, nous avons vu dans la section 2.4 que la normale, en plus de la direction de l'éclairage, peut déterminer l'intensité réfléchie par les surfaces. Il suffit d'utiliser lors de l'éclairage une carte de normales (Normal Map) qui correspond aux irrégularités souhaitées. Pour des géométries connues telles qu'une sphère, la carte normale peut être générée par des équations. Pour les modèles complexes, nous utilisons des tables de correspondance LUT.
Une autre technique appelée Mapping des hauteurs (Height Mapping) consiste à utiliser une carte de la hauteur des irrégularités à appliquer sur le modèle pour modifier la position des sommets. Cette technique présente l'avantage de pouvoir simuler l'effet d'ombre en capturant les irrégularités des contours (le Bump Mapping ne capture que les détails à l'intérieur de l'objet). Cependant, le Bump Mapping permet de capturer plus de détails, car il est exécuté au niveau des pixels et non des sommets, mais il est possible d'augmenter le nombre de sommets avec le shader de tesselation.

3 Lancer de rayons (Ray Tracing)

La puissance du ray tracing réside dans sa simplicité. Le principe de base illustré dans la figure 5.4 consiste à envoyer un rayon à partir du pixel et à trouver le point d'intersection avec l'objet le plus proche. Pour l'ombrage, un autre rayon est envoyé du point d'intersection à l'éclairage pour déterminer si l'objet est éclairé ou dans l'ombre et calculé la couleur du pixel.

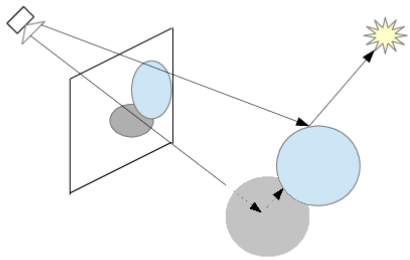

FIGURE 5.4 – Ray tracing illustration.

Un rayon est une demi-ligne 3D partant d'un point A (l'origine) vers un point

B. Il est défini par l'équation paramétrique :

$$R(t) = (1-t)A + tB, \tag{5.7}$$

Le paramètre t est utilisé pour déterminer le point d'impact. Cette équation est souvent écrite :

$$R(t) = O + t \cdot d, \tag{5.8}$$

avec O l'origine (point A) et d la direction (B-A).

Ainsi, pour l'ombrage d'un point P de la scène avec un éclairage à la position L, nous envoyons un rayon d'ombrage depuis l'origine P dans la direction $d = L - P$. Si l'intersection se produit dans l'intervalle $t \in [0, 1]$, le rayon rencontre une géométrie qui bloque la lumière. En pratique, nous considérons un intervalle $[t_{min}, t_{max}]$ en raison de l'imprécision des calculs de float, avec $t_{min} = \epsilon$ et $t_{max} = 1 - \epsilon$.

Pour prendre en compte des effets plus réalistes de l'éclairage et des matériaux, il existe différentes variantes et implémentations du ray tracing[Shi16]. L'objectif étant toujours d'obtenir des approximations de l'équation de transport. L'algorithme de Whitted [Whi05] considère des sources de lumière ponctuelles. Si la surface est diffuse, un rayon d'ombrage est envoyé à chaque source d'éclairage. Si la surface est spéculaire, un rayon de réflexion est utilisé. L'algorithme de Cook [CPC84, Coo86] (ou ray tracing stochastique) est celui qui a révolutionné le cinéma en ajoutant plusieurs effets. Les rayons sont lancés aléatoirement hors de la direction spéculaire (figure 5.5 à gauche) pour créer des effets de flou de mouvement, et aléatoirement vers l'éclairage pour simuler un éclairage non ponctuel.

En général, plusieurs rayons sont envoyés pour chaque pixel pour l'anticrénelage (antialiasing).

L'aspect force brute du lancement de plusieurs rayons par pixel, et de plusieurs rayons par point d'impact, et vers chaque source d'éclairage, rend la technique très gourmande en temps de calcul.

L'algorithme de Kajiya [Kaj86] (ou path tracing) a modifié l'algorithme de Cook, en appliquant le principe du lancement aléatoire de rayons pour les surfaces diffuses (figure 5.5 au milieu), et a introduit la notion de path tracing qui consiste à envoyer depuis chaque point d'impact un seul rayon mais dans une direction choisie par l'algorithme monte carlo. Un Path ou chemin (figure 5.5 à droite) correspond à la trajectoire suivie par le rayon depuis le pixel jusqu'à l'éclairage. Pour chaque pixel, plusieurs chemins sont générés afin d'améliorer l'estimation de la radiance.

Un compromis doit être trouvé entre le nombre de rayons et le débruitage pour accélérer le traitement et avoir un rendu photo-réaliste. Le débruitage reste important pour l'accélération du ray tracing malgré la puissance de calcul des GPU récents.

La programmation GPU du lancer de rayons est effectuée sur cinq shaders [HAM19] :

— Shader de génération de rayons : transformation des pixels en rayons.

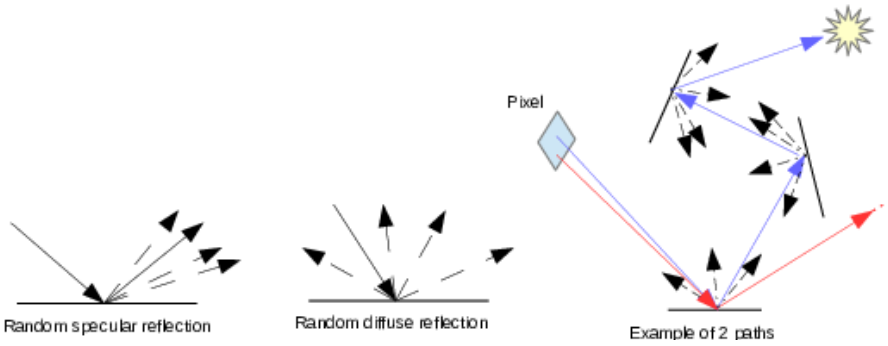

FIGURE 5.5 – Coulée de rayons aléatoires pour les surfaces spéculaires et diffuses, et génération de 2 trajectoires

.

— Intersection shader : code d'intersection des rayons avec la géométrie. Chaque géométrie (sphère, triangle, courbe de Bézier, etc.) nécessite un code différent.

— Miss shader : programmer ce qu'il faut faire si aucune intersection n'est trouvée.

— Closest Hit shader : programmer ce qu'il faut faire en cas d'intersection (modèle d'ombrage).

— Any hit shader : ajout d'effets tels que la transparence.

4 Réseaux adverses génératifs et auto-encodeurs

En apprentissage profond, nous distinguons les modèles discriminatifs qui permettent de séparer les données en classes (vus dans le chapitre 3) et les modèles génératifs qui permettent de générer des données d'une certaine classe. Étant donné une image X d'un objet et Y sa classe, un discriminateur permet de prédire la probabilité $P(Y/X)$ que X appartienne à Y. Inversement, un générateur permet de générer une image de synthèse X appartenant à une classe Y en maximisant la probabilité $P(X/Y)$. Les modèles génératifs les plus populaires sont le Variational Autoencoders (VAE) et les Generative Adversarial Networks (GAN), dont les architectures sont illustrées sur la figure 5.6.

Pendant l'apprentissage, le VAE encode l'image dans un vecteur "latent" qui alimente à son tour un décodeur pour générer une image réaliste. Pendant la phase de test, nous utilisons uniquement le décodeur en lui fournissant un vecteur aléatoire qui nous permet de générer de nouvelles images réalistes aléatoires. Les encodeurs et décodeurs sont généralement des réseaux de neurones convolutifs (CNN).

Les GANs ont révolutionné le domaine de la synthèse d'images en permettant de générer automatiquement et rapidement des images ultra-réalistes. Ils sont

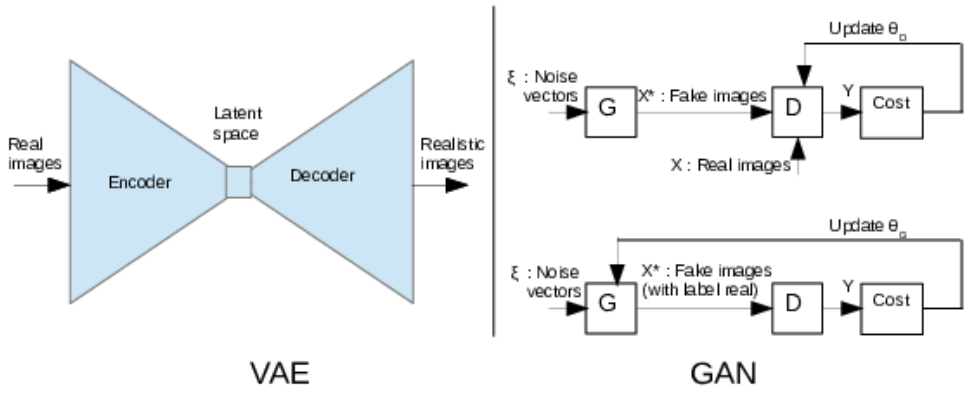

FIGURE 5.6 – Architecture VAE et formation du générateur et du discriminateur

.

utilisés dans de nombreuses applications image et vidéo telles que la création de visages [KLA+20], la traduction d'images (image translation) [PLWZ19] qui permet la création de scènes photoréalistes à partir d'une segmentation grossière, l'animation de portraits [ZSBL19], ou la génération d'objets 3D [WZX+16].
Un GAN se compose de deux modèles : un générateur G et un discriminateur D, qui sont généralement des modèles CNN. Une analogie de l'initiateur du GAN [GPAM+14] compare G à un faussaire d'oeuvres d'art et D à un inspecteur qui tente de détecter les faux tableaux. Les deux modèles, par un jeu compétitif, apprennent l'un de l'autre et améliorent leurs performances. Au bout d'un certain temps, il devient impossible de différencier la contrefaçon de l'original. Le système GAN est illustré dans la figure 5.6. G est équivalent au décodeur dans VAE, sauf qu'il n'est pas guidé par un encodeur et reçoit un bruit aléatoire à son entrée.
Nous alternons l'apprentissage de D et de G. Pour entraîner D (figure 5.6 en haut à droite), nous alimentons d'abord G avec un ensemble (batch) de vecteur de bruit ϵ pour générer de fausses images X^*. D est ensuite alimenté avec X^* ainsi que des images réelles (avec les classes correspondantes : images réelles ou fausses). Les paramètres θ_d de D sont optimisés en minimisant une fonction objective Binary Cross Entropy (BCE) :

$$J = -\frac{1}{m}\sum_{i=1}^{m}[Y^i log(h(X^i,\theta) + (1-Y^i) log(1-h(X^i,\theta)], \tag{5.9}$$

avec m le nombre d'échantillons dans le batch, X^i l'échantillon d'image et Y^i l'étiquette correspondante, et $h(X^i,\theta)$ la prédiction de X^i avec les paramètres θ.
L'apprentissage de G (figure 5.6 en bas à droite) est effectué de manière similaire, sauf que nous fournissons à D que de fausses images X^* mais avec

des étiquettes d'images réelles, car nous voulons que G apprenne à tromper D afin de générer des images réalistes. Par conséquent, les paramètres θ_g de G sont optimisés en maximisant la fonction BCE. Afin d'éviter les problèmes de "mode collapse" et de "vanishing gradient", dus au fait que D apprend plus vite que G, et donc que l'apprentissage de G ne progresse plus, nous pouvons utiliser un GAN convolutif profond (DCGAN) [RMC15] ou remplacer la BCE par une fonction objective de Wasserstein (WCGAN) [ACB17].

La génération avec les GANs peut se faire de trois manières différentes :

— Inconditionnelle : synthèse d'images aléatoires d'une classe aléatoire.

— Conditionnel : synthèse d'images aléatoires d'une classe spécifiée à l'entrée de G et de D.

— Contrôlable : spécification d'une caractéristique dans le contenu de l'image à générer (position de l'objet, direction du regard, direction de l'éclairage, etc.)

Les GAN contrôlables [LS19] sont ceux qui nous intéressent ici, afin de pouvoir contrôler les paramètres de l'objet, de la caméra et de l'éclairage de la même manière que les méthodes traditionnelles de rastérisation ou le ray tracing.
Pour modifier une caractéristique dans l'image générée, il suffit de modifier une valeur dans le vecteur latent (bruit). Mais nous devons savoir quelle valeur du vecteur latent correspond à la caractéristique que nous voulons modifier. Habituellement, un classificateur est utilisé à la sortie du générateur pour détecter la caractéristique que nous voulons contrôler et pénaliser le générateur pour chaque image ne contenant pas cette caractéristique. Pour éviter que les caractéristiques soient corrélées, un grand vecteur latent est utilisé.

5 Rendu neuronal

Les algorithmes de rendu neuronal (Neural Rendering) permettent, en combinant les techniques d'apprentissage automatique avec l'aspect physique de l'informatique graphique, de générer des rendus photoréalistes contrôlables et des modèles 3D accessibles dans des environnements virtuels. Le principe est d'entraîner des modèles à générer des images à partir de différents paramètres spécifiés d'éclairage, de géométrie, de matériau et de caméra.

5.1 Synthèse de nouvelles vues

La synthèse de vue nouvelle (Novel View Synthesis) consiste à effectuer, à partir d'une ou de plusieurs images d'une scène, un rendu photoréaliste en simulant un mouvement de caméra. Dans [ERB+18], un réseau génératif (GQN) prend en entrée des images d'une scène acquises depuis différents points de vue, et permet de générer l'image d'un nouveau point de vue. Le GQN

est composé d'un encodeur qui extrait un vecteur représentatif de la scène à partir des images observées, et d'un générateur qui prend en entrée le vecteur représentatif, ainsi que la position et l'orientation souhaitées de la caméra, et génère le point de vue correspondant. La figure 5.7 illustre le principe et l'architecture du modèle.

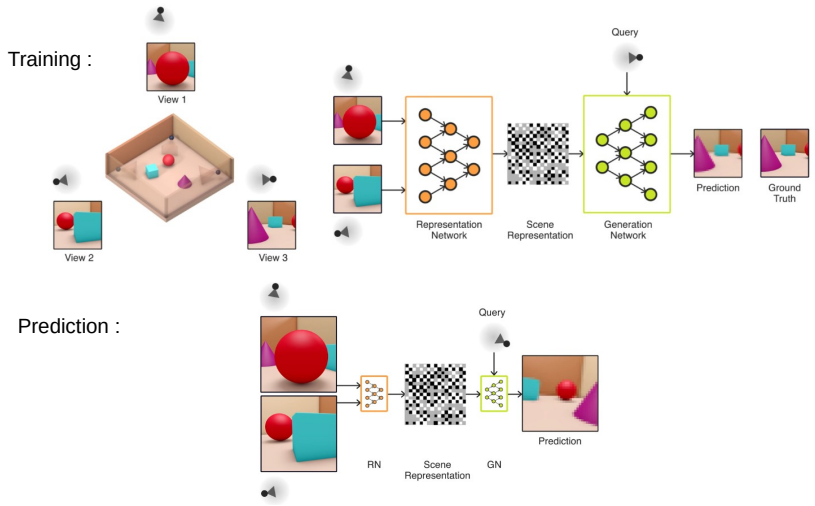

FIGURE 5.7 – Le GQN est constitué d'un encodeur qui extrait un vecteur représentatif de la scène à partir des images observées et d'un décodeur qui génère une image non observée d'un nouveau point de vue. Les paramètres des nouvelles position et orientation de la caméra sont fournies en entrée du décodeur.

Dans [LSS+19], le VAE illustré sur la figure 5.8 est entraîné à générer un modèle 3D animé à partir de vidéos de différents points de vue. Le modèle 3D est utilisé pour générer une image d'un nouveau point de vue par ray tracing à partir des paramètres caméra.

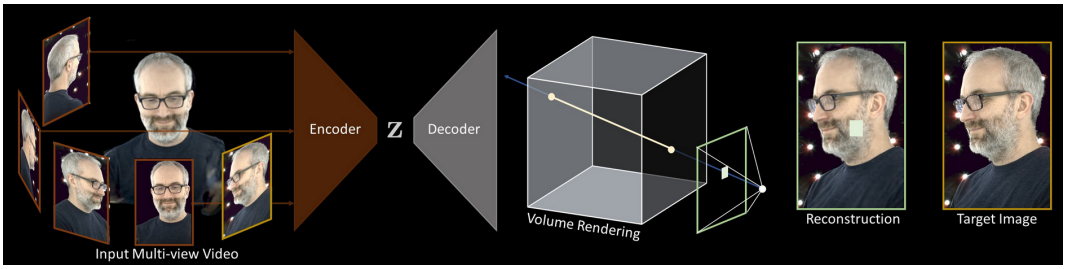

FIGURE 5.8 – Génération d'un modèle 3D à partir de différents points de vue et rendu par ray tracing pour la synthèse d'une nouvelle vue.

5.2 Relighting et matériaux

Le Relighting consiste à générer, à partir d'images acquises avec différentes positions d'éclairage, une image avec n'importe quelle position d'éclairage. Cela permettra en combinaison avec la synthèse de nouvelles vues, de contrôler à la fois l'éclairage et la position de la caméra.
L'approche la plus simple consiste à reconstruire la géométrie et les propriétés des matériaux de la scène, puis à effectuer le rendu en changeant la position de l'éclairage. Cependant, l'approche de reconstruction est difficile pour les matériaux et les géométries complexes. Ces méthodes (voir [WDT+09] et [RDL+15]) utilisent l'équation de transport 5.1, où la fonction f peut être échantillonnée en utilisant différentes positions d'éclairage. L'image éclairée depuis un nouveau point de vue peut être obtenue en interpolant f. Cette méthode permet d'obtenir des résultats très photoréalistes, mais nécessite plusieurs images en entrée (entre 10 et 100). Dans [XSHR18], un modèle permet d'estimer l'image d'une scène éclairée depuis un nouveau point de vue, à partir de seulement cinq images d'éclairages différents. Pour cela, un VAE illustré sur la figure 5.9 prend en entrée les images des différents éclairages, ensuite, le vecteur latent est modifié avec la nouvelle direction d'éclairage afin de générer l'image correspondante en sortie. Aussi, afin de ne sélectionner que les points de vue pertinents (pas très proches), un premier réseau est optimisé pour effectuer cette opération. D'autres méthodes fondées sur les VAEs permettent un ré-éclairage plus spécifique de portrait [SBT+19], de corps [KE19], ou de scène extérieure [PGZ+19].

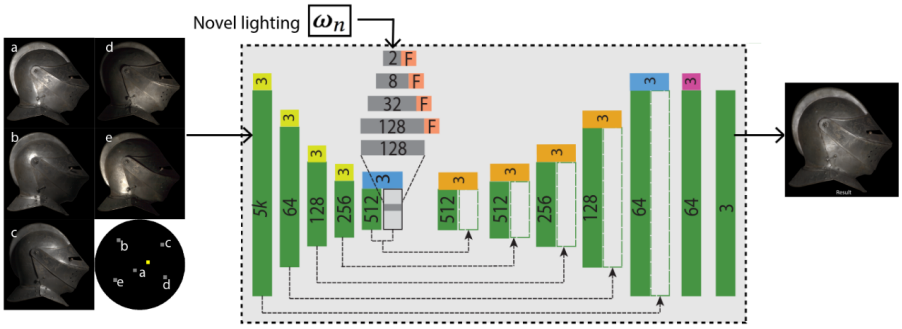

FIGURE 5.9 – Relighting avec un VAE.

5.3 Content control

Pour le contrôle de la géométrie, beaucoup de travaux ont été réalisés pour le contrôle des visages 3D. Dans [KGT+18], le modèle utilisé prend en entrée deux portraits vidéos, source et cible, et permet de transférer de la source à la cible, à la fois la pose de la tête, les expressions faciales, et le mouvement du regard. Pour cela, un modèle paramétrique est d'abord utilisé pour extraire de la source et de la cible, les paramètres à transférer ainsi

que les paramètres fixes d'illumination et d'identité. Les nouveaux paramètres permettent d'effectuer par rastérisation le rendu d'un modèle 3D de la cible. Enfin, un VAE est entraîné à générer une image photo-réaliste à partir du modèle 3D. Une technique similaire avec une nouvelle architecture est présentée dans [KDRZ20]. Dans [ZSBL19], un modèle fondé sur un GAN permet de générer de nouvelles positions du visage à partir de points de repère des traits du visage (dessin grossier du visage).

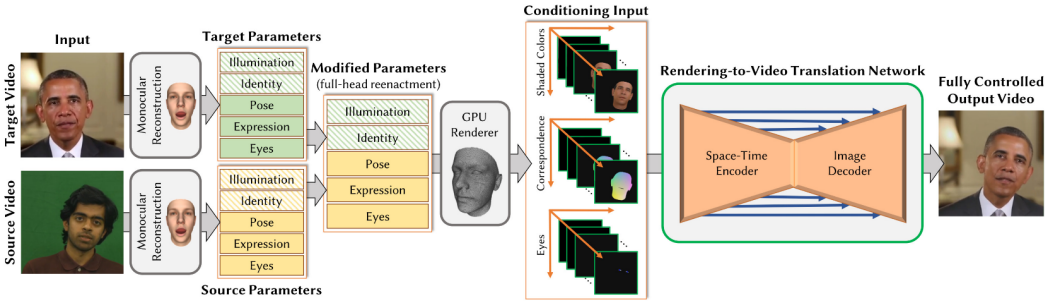

FIGURE 5.10 – Transfert de la pose de la tête, des expressions faciales et du mouvement du regard d'une vidéo source vers une vidéo cible.

6 Comparaison

La figure 5.11 montre un tableau comparatif des quatre approches selon les critères discutés ci-dessous.

La rastérisation permet une bonne qualité, mais inférieure au ray tracing à cause des approximations faites sur le modèle d'ombrage qui génèrent des artefacts et ne permettent pas de simuler tous les effets de réflexion nécessaires au photoréalisme. Cependant, la rastérisation est plus facile à implémenter en temps réel sur n'importe quel hardware et elle est souvent utilisée dans les jeux vidéo et le web, mais aussi pour certains films d'animation en raison de la rapidité du rendu.

Le ray tracing a longtemps été difficile à implémenter en temps réel, il a donc été utilisé hors ligne pour des applications telles que les films, mais aujourd'hui, c'est possible grâce à la puissance des GPU récents. Le ray tracing reste toujours la principale solution pour créer des contenus photoréalistes. Nous avons également vu que la rastérisation et le ray tracing permettent de contrôler tous les paramètres de la scène (éclairage, objet et caméra) mais nécessitent beaucoup de temps de travail (modélisation géométrique et matérielle, éclairage, texture, animation, etc.) et de temps de calcul du rendu. En revanche, les techniques GAN sont plus rapides et automatiques. Pour effectuer le rendu avec un GAN contrôlable, nous utilisons des classifieurs pour détecter les caractéristiques (âge, cheveux, etc.) que nous voulons contrôler, mais le problème est qu'il faut un classifieur pour chaque caractéristique et

	Photo-realism	Real Time CPU	Real Time GPU	Working time	Accessibility of the 3D model for manipulation	Controllability of the scene content	Quality of image details
Rasterization	Medium	Yes	Yes	Slow but tools available	Yes	Total	High
Ray tracing	Yes	No	Yes	Slow but tools available	Yes	Total	High
GAN	Yes	Yes	Yes	Fast generation but requires the development of new models for each type of object	No	Limited (not all types of content)	Medium (artifacts)
Neural rendering	Yes	Yes	Yes	Fast generation but requires the development of new models for each type of object	Yes	Limited (loss of quality when camera and lighting are moved simultaneously)	Medium (artifacts)

FIGURE 5.11 – Tableau comparatif : rastérisation, ray tracing, GAN et rendu neuronal.

qu'il doit être fiable pour ne pas induire le GAN en erreur.
Le rendu neuronal permet de générer des modèles 3D à partir d'images réelles, et d'effectuer un rendu photo-réaliste à partir de n'importe quelle position de caméra ou d'éclairage. Cependant, cette technique nécessite de très grands ensembles de données pour l'apprentissage. De plus, des pertes de qualité sont observées lors du déplacement simultané de la caméra et de l'éclairage. Par conséquent, ces méthodes ne permettent pas un contrôle total de la scène comme dans le cas du rastérisation et du ray tracing. Cependant, le rendu neuronal est très pertinent pour les applications en temps réel comme la vidéoconférence ou la réalité virtuelle, ou pour les applications hors ligne comme les effets visuels.

7 Détection de la pose humaine par caméra 3D Kinect

La Kinect est une caméra 3D qui utilise les radiations infrarouges pour capturer la carte de profondeur de la scène, qui nous renseigne sur la distance des objets par rapport à la caméra. Les avantages d'une carte de profondeur par rapport à une image RGB sont multiples : il est plus facile de différencier les pixels d'un même objet, l'image obtenue est indépendante de l'éclairage et des paramètres caméra, elle permet une reconstruction 3D de la scène, et ce type d'images est plus facile à traiter, ce qui permet de détecter et de suivre des personnes et de localiser les articulations et les différentes parties du corps. La figure 5.12 montre un exemple de carte de profondeur et de détection de la pose humaine réalisées avec la Kinect.

Les applications des caméras 3D et particulièrement la Kinect (grâce à son accessibilité) sont nombreuses comme le montre la figure 5.13 : jeux vidéo,

FIGURE 5.12 – Première ligne : deux versions de la Kinect. Deuxième ligne : image RGB, carte de profondeur et détection de la pose humaine.

interfaces gestuelles, détection de la pose humaine, détection des expressions faciales, mesure des performances sportives, visio-conférence immersives, réalité virtuelle, etc.

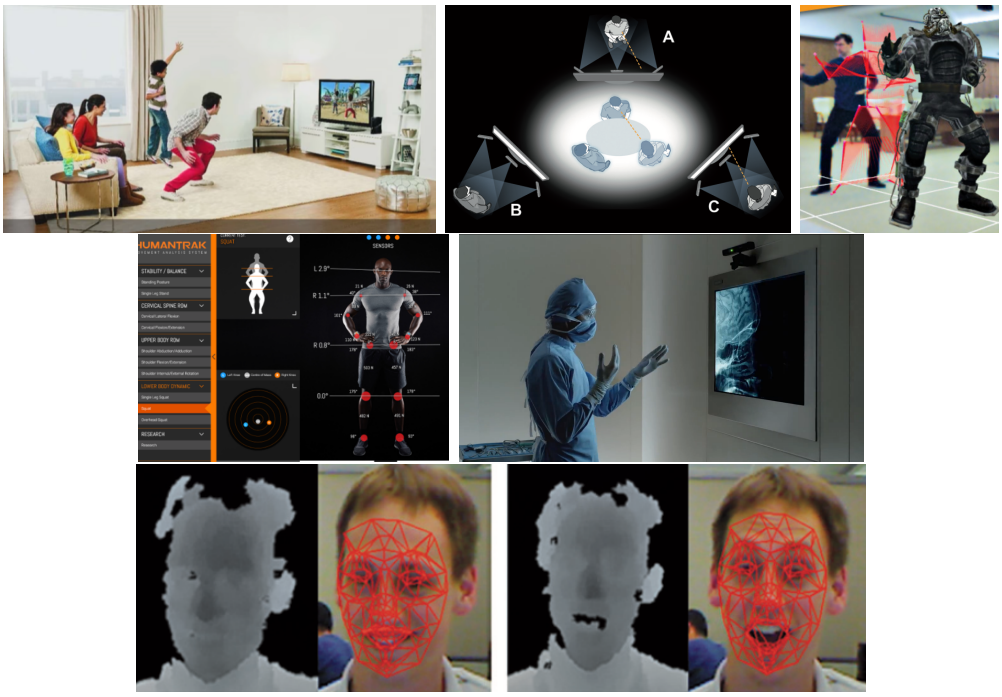

FIGURE 5.13 – Exemples d'applications de la Kinect.

Le succès de la Kinect est dû au travail de la communauté Open Source qui a rendu accessible les pilotes logiciels (drivers) "OpenKinect" et "libfreenect" quelques jours après sa commercialisation par Microsoft. Un peu plus tard différentes APIs avec des fonctionnalités avancées ont vu le jour : "OpenNI Sensor Kinect", "PrimeSense NITE", etc. Ces APIs implémentent divers algorithmes de détection de la pose humaine. Plus tard, Windows a rendu

accessible gratuitement sont SDK Kinect.
La figure 5.14 montre les différents composants de la Kinect. L'appareil contient une caméra standard, une caméra infra-rouge, un projecteur infra-rouge, un moteur (rotation verticale ±30) et 4 microphones (son quadriphonique pour la localisation dans l'espace, la surpression de bruit, etc.).

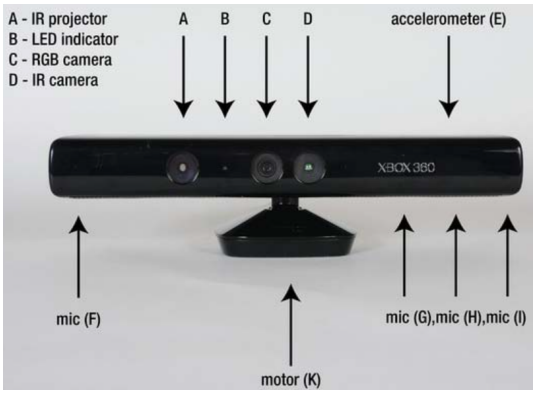

FIGURE 5.14 – Hardware Kinect.

Le principe de capture de la carte de profondeur à l'aide de deux caméras ordinaires (pas la Kinect) est illustré sur la figure 5.15. Les centres des caméras gauche et droite correspondent à C_l et C_r dans la figure, les points x_l et x_r correspondent à la projection du point 3D X dans les deux images des caméras. Nous commençons par estimer une carte de disparité de chacune des images. Pour cela, pour chaque pixel dans l'image de gauche (ou de droite), nous cherchons son correspondant dans l'autre image, à l'aide d'un algorithme de mise en correspondance (voir chapitre 2). Les deux caméras étant alignées, la zone de recherche est réduite à une seule ligne (épipolaire).

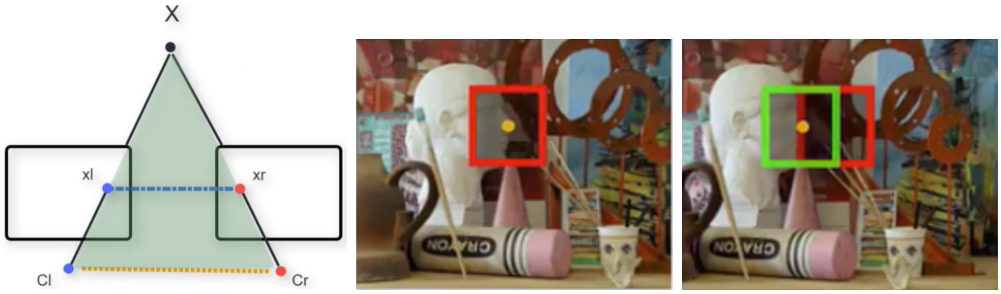

FIGURE 5.15 – Système stéréo.

La distance entre les points x_l et x_r correspond à la disparité du point X dans le système stéréo, comme illustré sur la figure 5.16.

La profondeur de X (distance Z du point X par rapport à la caméra stéréo) peut ensuite être estimée en connaissant la focale f et la distance inter-caméras B, comme le montre la figure 5.17.

La kinect utilise le principe de lumière structurée qui consiste à remplacer les deux caméras par un projecteur infra-rouge et une caméra infra-rouge. Le

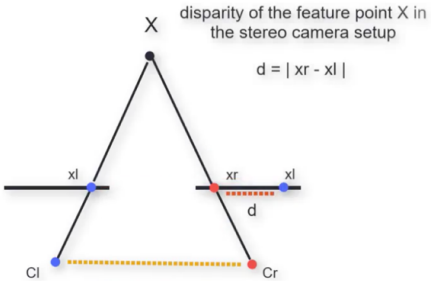

FIGURE 5.16 – Calcul de la disparité.

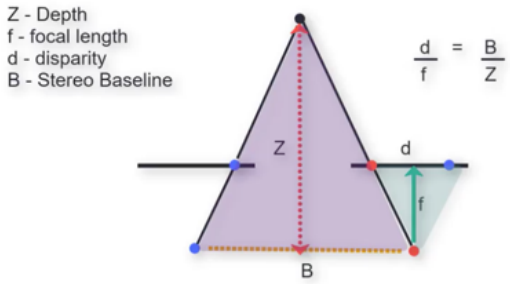

FIGURE 5.17 – Calcul de la profondeur.

principe consiste à projeter sur la scène un signal connu (forme, code barre, etc.) et d'observer le signal projeté dans l'image infra-rouge. La figure 5.18 illustre ce principe. Le calcul de la carte de la disparité et de la carte de profondeur s'effectue comme précédemment.

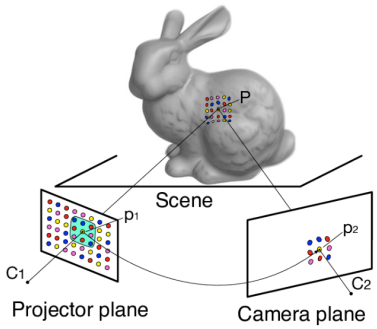

FIGURE 5.18 – Lumière structurée.

Nous constatons sur la figure 5.18 que les deux caméras peuvent ne pas être alignées, mais en connaissant la distance inter-caméras B, il est possible d'aligner parfaitement les images par des algorithmes de recalage.
Notons que la Kinect n'est pas parfaite, la distance des objets détectés est limité entre 1 et 7 mètres. Aussi, parfois certains points observés par la caméra ne sont pas atteints par le projecteur à cause de la présence d'obstacles, ce qui créé des taches noires dans la carte de profondeur.

Détection de la pose humaine

L'une des applications VFX qui nous intéresse ici est l'utilisation de la Kinect pour effectuer la détection de la pose humaine. Le but est de trouver la position 3D des différentes articulations du corps afin de créer une représentation 3D du squelette comme le montre la figure 5.19.

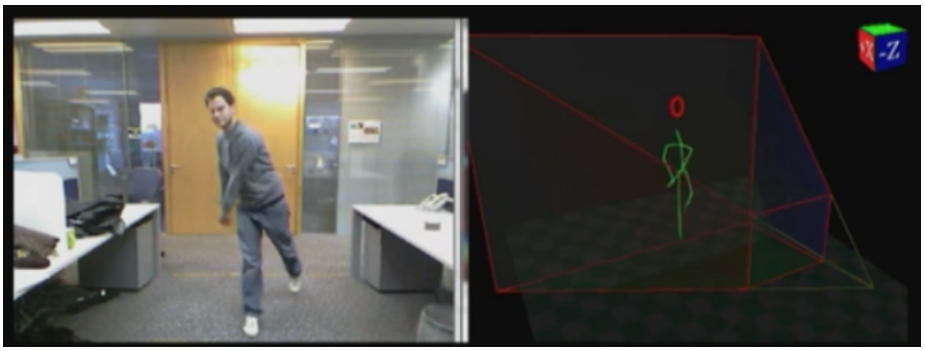

FIGURE 5.19 – Capture de mouvement avec la Kinect.

Pour effectuer la détection de la pose humaine, ou la reconnaissance des formes de manière plus générale, il faut prendre en considération plusieurs problèmes : différentes positions, occlusion, éclairage, différentes formes et dimensions, bruits, etc. La figure 5.20 montre des exemples d'images de différentes situations.

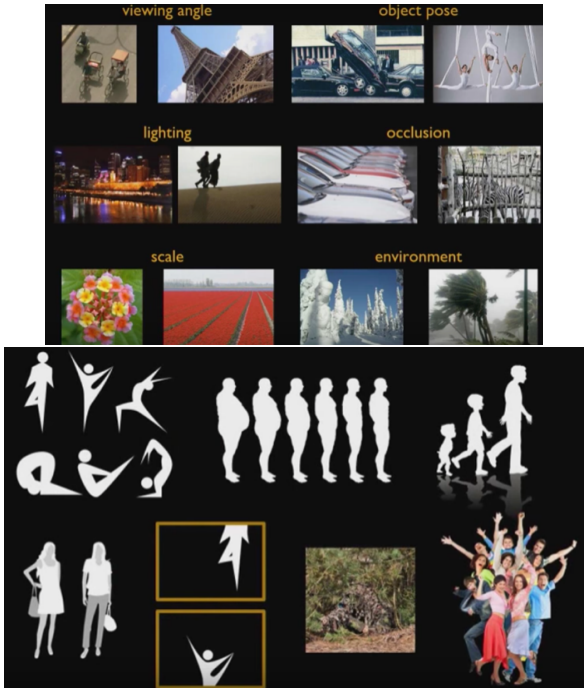

FIGURE 5.20 – Problèmes de variations dans la détection de la pose humaine et la reconnaissance des formes de manière générale.

Il existe plusieurs solutions pour résoudre ces problèmes. Dans le domaine des VFX, la technique la plus utilisée est la capture de mouvement avec de multiples caméras (entre 10 ou 20) calibrées minutieusement et placées autour de la pièce. Les acteurs portent des costumes spécifiques contenant des marqueurs. Les caméras suivent les marqueurs pour reconstruire le mouvement. Cette approche permet d'obtenir un résultat très précis et à une haute cadence d'images, mais nécessite un matériel cher, encombrant et calibré. Elle est souvent réservée à des films à grand budget.

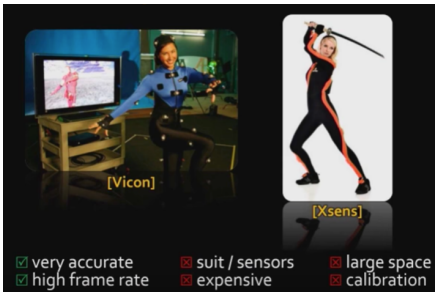

FIGURE 5.21 – Motion capture.

Il existe d'autres approches fondées sur la vision par ordinateur.
Plusieurs travaux [GWK05, ZF07, KHS10, SM10, GPKT10, PGKT10, BMB+13] ont été réalisés avec la carte de profondeur obtenue par une paire de caméras RGB (stéréo), mais aucune méthode n'a été adaptée pour qu'elle fonctionne quelque soit l'environnement ou la personne. Cela est dû aux problèmes de variations qui persistent dans les images RGB.
Un autre type d'approche est le suivi de mouvement qui consiste à prédire la position à l'instant $t+1$ à partir de la position à l'instant t et de la carte de profondeur à l'instant $t+1$. L'inconvénient de cette approche est l'initialisation de la pose et les cas de mouvements rapides du sujet.
La troisième catégorie d'approches par vision consiste à comparer la pose de départ à différentes positions possibles afin de trouver la plus proche (méthode hiérarchique) [Gav00, OS08]. Le problème est qu'il faut faire un très grand nombre de comparaisons à cause du grand nombre de possibilités de pose. Il y a aussi un risque qu'une partie du corps ne soit pas suivie si elle n'est pas initialisée. Une alternative est de faire la correspondance pour chaque partie du corps [IF01, BM09], mais il est difficile de l'effectuer de manière efficace en considérant les variations possibles des poses.
L'approche Kinect permet de classifier chaque pixel de la carte de profondeur en une partie du corps. Il s'agit d'une méthode locale et non-temporelle : chaque pixel est comparé uniquement à son voisinage et le traitement s'effectue image par image. Par conséquent, le traitement est plus rapide que les approches par vision, avec la possibilité d'implémentation parallèle sur GPU. La classification s'effectue par apprentissage automatique (voir chapitre 3). Le modèle utilisé est entrainé avec des données de carte de profondeur et les labels des différentes parties du corps. La dataset contient environ $100K$ échantillons correspondants

à différentes positions et formes pour rendre la détection invariante (voir figure 5.22).

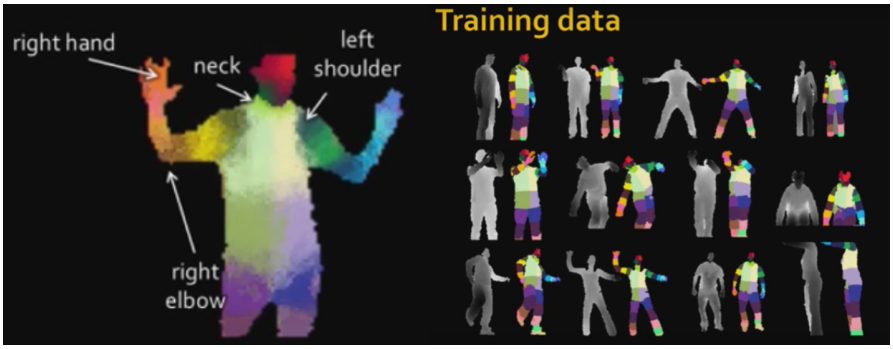

FIGURE 5.22 – Labelisation des différentes parties du corps et dataset utilisée pour l'apprentissage de la détection de la pose humaine.

8 Conclusion

Les différentes approches existantes de rendu 3D ont été présentées : la rastérisation et le Ray Tracing, ainsi que les méthodes de Deep Learning GAN et rendu neuronal. Une comparaison a été effectuée dans le but initial de nous permettre de choisir la meilleure approche pour notre étude sur les VFX de Matchmoving et sur la réalité augmentée. Les méthodes de Deep Learning ont révolutionné et changé les paradigmes dans le domaine de la synthèse d'images, grâce à la qualité ultra-réaliste et à la génération automatique d'images. Cependant, les techniques d'apprentissage profond sont encore limitées en termes de contrôle de la scène, et ne permettent pas de manipuler les objets, la caméra et l'éclairage avec la même facilité que les techniques traditionnelles. L'avantage de ces méthodes est que le temps de travail est beaucoup plus réduit que les méthodes traditionnelles qui nécessitent d'apprendre un outil de modélisation et de créer des modèles géométriques manuellement. Par conséquent, les méthodes d'apprentissage profond sont très pertinentes pour certaines applications telles que la réalité virtuelle. Dans le chapitre 7, la méthode de rastérisation a été combinée avec des méthodes de mise en correspondance, vues dans les chapitres précédents, pour proposer un système complet de Matchmoving. Nous avons aussi découvert l'utilisation d'une caméra 3D Kinect pour la détection de la pose humaine. Il s'agit d'une autre approche d'insertion de modèles 3D dans une vidéo réelle et d'animation guidée par des acteurs. Dans le chapitre suivant, nous allons présenter différents algorithmes VFX utilisés en post-production pour diverses applications et particulièrement le Matchmoving et la réalité augmentée.

Chapitre 6

Composition, retouche et restauration d'images

Sommaire

Dans ce chapitre, nous allons présenter des algorithmes de retouches, de composition et de restauration des images, les plus utilisés dans les domaines des VFX et particulièrement comme post-traitement de Matchmoving.

1 Image Matting

Le Matting image (ou alpha matting) consiste à extraire à partir d'une image I, un objet F (Foreground) et le fond B (background), dans le but d'effectuer une nouvelle composition avec l'objet ou le fond. La formulation mathématique du problème s'écrit :

$$I(x,y) = \alpha(x,y).F(x,y) + (1 - \alpha(x,y)).B(x,y), \tag{6.1}$$

avec I l'image observée et F, B et α les images recherchées. Pour simplifier, nous omettons dans l'écriture des équations les coordonnées pixels (x, y).
Le canal de transparence α (alpha matte), est une image en niveau de gris prenant une valeur de 1 pour les pixels de l'objet à extraire, une valeur de 0 pour les pixels du fond et une valeur entre 0 et 1 pour certaines zones de l'image : pixels des contours recevant à la fois la lumière de l'objet et du fond (surtout en basse résolution), les zones floues à cause du mouvement ou à cause de l'ouverture de l'objectif caméra et les objets transparents, translucides ou minces (chevelures).
Le Matting exige donc un masque alpha avec des contours lisse à la différence de la segmentation qui est binaire.
Les images couleurs I, F et B possèdent chacune 3 composantes RGB : $I = [I_r I_g I_b]^T$, $F = [F_r F_g F_b]^T$ et $B = [B_r B_g B_b]^T$.
Le problème est donc mal posé, puisque nous avons un système à 3 équations et 7 inconnues (α, F_r, F_g, F_b, B_r, B_g et B_b). Le système possède donc plusieurs solutions si nous ne faisons pas d'hypothèses supplémentaires sur la scène.
Nous distinguons les hypothèses simples dans le cas d'un fond uniforme et le cas plus complexe d'un fond texturé quelconque.

1.1 Fond uniforme

Dans un environnement contrôlé, telles que les scènes de tournage des films, nous utilisons souvent des fonds vert ou bleu, puisque ces couleurs ne contiennent pas la composante rouge, étant donné que nous cherchons le plus souvent à extraire des acteurs (à cause de la couleur rouge de la peau). Cela suppose aussi que les acteurs ne doivent pas porter des costumes contenant la couleur du fond.
En considérons un fond uniforme B connu, le système est réduit à 3 équations et 4 inconnues. Nous avons donc besoin d'une 4ème équation pour résoudre le problème. L'hypothèse de Vlahos [Vla71] permettant de déduire le canal α s'écrit :

$$\alpha = 1 - a_1.(I_b - a_2.I_g), \tag{6.2}$$

avec a_1 et a_2 des valeurs paramétrables.
Cette méthode est intuitive. L'idée est qu'en supposant un fond bleu, le but est d'avoir une valeur α qui tend vers 0 lorsque $I_b \geq I_g$.
La première ligne de la figure 6.1 montre un exemple de résultat que nous avons obtenu en appliquant cette formule sur une image avec un fond bleu. Nous constatons que le canal α contient des erreurs. L'inconvénient est que cette méthode ne fonctionne que pour certaines couleurs. Notons que la composition est effectuée avec la formule de Matting en remplaçant F avec l'image du nouveau fond.
Une autre heuristique est proposée par l'auteur pour prendre en compte d'autres couleurs :

$$\alpha = 1 - a_1.[I_b - a_2.[a_5.max(r, g) + (1 - a_5).min(r, g)]], \tag{6.3}$$

avec $r = a3. * I_r$ et $g = a_4. * I_g$. les coefficients a_1, a_2, a_3, a_4 et a_5 sont paramétrables.
La figure 6.1 montre un exemple de résultat que nous avons obtenu en appliquant cette nouvelle formule. Nous obtenons moins d'erreurs de couleurs et un canal α plus lisse au niveau des contours.

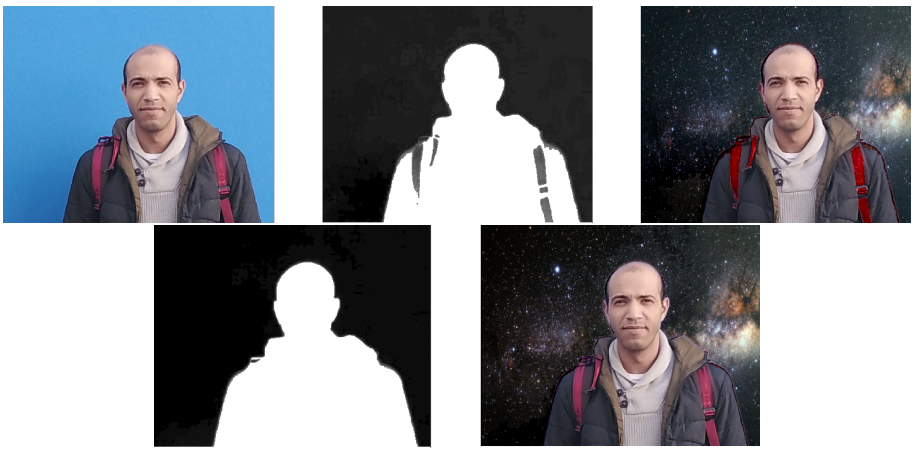

FIGURE 6.1 – Résultat d'extraction du canal de transparence α avec les formules de Vlahos et composition avec un nouveau fond.

Dans [SB96], des formulations mathématiques (pas d'heuristiques) du problème sont présentées en supposant que l'objet ne contienne pas de couleur bleue. Les auteurs proposent aussi une approche différente consistant à utiliser deux images I_1 et I_2 avec des fonds différents B_1 et B_2. Notons que les deux images B_1 et B_2 doivent être capturées sans l'objet du fond, nous utilisons donc 4 images au total. Nous obtenons donc le système suivant :

$$\begin{aligned} I_1 &= \alpha F + (1 - \alpha)B_1 \\ I_2 &= \alpha F + (1 - \alpha)B_2, \end{aligned} \tag{6.4}$$

d'où :

$$(I_1 - I_2) = (1 - \alpha)(B_1 - B_2) \tag{6.5}$$

Nous avons donc un système à 3 équations et une seule inconnue α, il s'agit donc d'un système surdéterminé. La solution par moindre carré est :

$$\begin{aligned} \alpha &= 1 - \frac{(I_1 - I_2) \bullet (B_1 - B_2)}{||B_1 - B_2||^2} \\ &= 1 - \frac{(I_1 - I_2)^T.(B_1 - B_2)}{(B_1 - B_2)^T.(B_1 - B_2)}, \end{aligned} \tag{6.6}$$

avec $\bullet$ le produit scalaire.
En pratique, il est difficile d'obtenir 4 images I_1, I_2, B_1 et B_2 parfaitement alignées et avec un éclairage stable. Cette méthode est réservée à un environnement de laboratoire très bien contrôlé.

1.2 Fond texturé

Dans un environnement quelconque, l'intervention de l'utilisateur ou une segmentation de l'image sont nécessaires pour fournir à l'algorithme de Matting des informations supplémentaires sur l'objet et le fond.
Deux types d'hypothèses, illustrées sur la figure 6.2, sont utilisées :

— Trimap : Segmentation grossière de l'image en 3 régions : objet, fond et inconnues. Il s'agit de donner des valeurs de 1 et 0 aux pixels dont nous sommes sûrs qu'ils appartiennent à l'objet ou au fond et des valeurs de $0,5$ pour les zones inconnues.

— Scribbles : L'utilisateur indique dans l'image certaines régions de l'objet et du fond, le reste de l'image est considéré comme régions inconnues. Cette méthode est plus simple pour l'utilisateur.

FIGURE 6.2 – Trimap : images du haut. Scribbles : images du bas.

1.3 Matting bayésien

L'idée de base est de maximiser la probabilité d'obtenir F, B et α, sachant I :

$$\arg\max_{F,B,\alpha} P(F,B,\alpha|I) \tag{6.7}$$

En appliquant la règle de Bayes, nous obtenons :

$$\arg\max_{F,B,\alpha} \frac{P(I|F,B,\alpha)P(F,B,\alpha)}{P(I)} \tag{6.8}$$

Nous pouvons omettre le terme $P(I)$ car il ne dépend pas des paramètres à estimer et étant donné que F, B et α sont indépendants. Nous obtenons :

$$\arg\max_{F,B,\alpha} P(I|F,B,\alpha)P(F)P(B)P(\alpha) \tag{6.9}$$

Le terme de données $P(I|F,B,\alpha)$ est modélisé par une distribution gaussienne telle que I, F, B et α soient cohérents avec l'équation de Matting 6.1 :

$$P(I|F,B,\alpha) = \exp^{\frac{-1}{\sigma^2}||I-(\alpha F+(1-\alpha)B)||_2^2}, \tag{6.10}$$

l'écart type σ est configurable et permet de modéliser de combien nous nous éloignons de l'équation de Matting.
Les termes des probabilités a priori de F et B sont modélisés par des distributions gaussiennes :

$$\begin{aligned} P(B) &= \frac{1}{(2\pi)^{3/2}|\Sigma_B|^{1/2}} \exp^{-\frac{1}{2}(B-\mu_B)^T\Sigma_B(B-\mu_B)} \\ P(F) &= \frac{1}{(2\pi)^{3/2}|\Sigma_F|^{1/2}} \exp^{-\frac{1}{2}(F-\mu_F)^T\Sigma_F(F-\mu_F)} \end{aligned} \tag{6.11}$$

Les moyennes μ_B et μ_F, et les covariances Σ_B et Σ_F sont estimées en utilisant les images Trimap ou Scribbles fournit par l'utilisateur :

$$\begin{aligned} \mu &= \frac{1}{N}\sum_{i=1}^{N} I_i \\ \Sigma &= \frac{1}{N}\sum_{i=1}^{N}(I_i-\mu)(I_i-\mu)^T, \end{aligned} \tag{6.12}$$

avec N le nombre d'échantillons de F ou de B dans la carte Trimap/Scribbles et I_i la valeur dans l'image I de ces échantillons.
Le terme a priori $P(\alpha)$ est considéré constant pour simplifier les calculs. Dans [AF04], une distribution beta est utilisée.
Nous introduisons ces termes dans l'équation 6.9 et nous appliquons le log pour simplifier les calculs :

$$\arg\max_{F,B,\alpha} \frac{-1}{\sigma^2}||I-(\alpha F+(1-\alpha)B)||_2^2 - \frac{1}{2}(B-\mu_B)^T\Sigma_B(B-\mu_B) - \frac{1}{2}(F-\mu_F)^T\Sigma_F(F-\mu_F) \tag{6.13}$$

Les termes constants ont été omis, car ils ne dépendent pas des termes à estimer.
Cette équation est maximisée en cherchant F, B et α qui annulent les dérivées par rapport à ces paramètres. Nous obtenons :

$$\begin{vmatrix} \Sigma_F^{-1} + \frac{\alpha^2}{\sigma^2}\mathbf{I}_{3\times3} & \frac{\alpha(1-\alpha)}{\sigma^2}\mathbf{I}_{3\times3} \\ \frac{\alpha(1-\alpha)}{\sigma^2}\mathbf{I}_{3\times3} & \Sigma_B^{-1} + \frac{(1-\alpha)^2}{\sigma^2}\mathbf{I}_{3\times3} \end{vmatrix} \begin{vmatrix} F \\ B \end{vmatrix} = \begin{vmatrix} \Sigma_F^{-1}\mu_F + \frac{\alpha}{\sigma^2}I \\ \Sigma_B^{-1}\mu_B + \frac{1-\alpha}{\sigma^2}I \end{vmatrix} \tag{6.14}$$

$$\alpha = \frac{(I-B)\bullet(F-B)}{||F-B||^2} \tag{6.15}$$

avec $\mathbf{I}_{3\times3}$ la matrice identité.
Le Bayesian Matting [CCSS01] est un algorithme itératif qui consiste à initialiser α avec la Trimap/Scribble et d'alterner entre l'estimation de F et B à partir de l'équation 6.14 et l'estimation de α avec l'équation 6.15 jusqu'à la convergence.
Pour éviter le problème de chevauchement entre les distributions gaussiennes F et B, nous estimons ces distributions comme des mixtures de gaussiennes (GMM) et nous appliquons l'algorithme précédent pour chaque composante GMM. La solution retenue est celle qui maximise l'équation 6.9.
Dans le cas des images complexes, il est possible d'appliquer la méthode localement sur des fenêtres centrées sur le pixel à estimer.

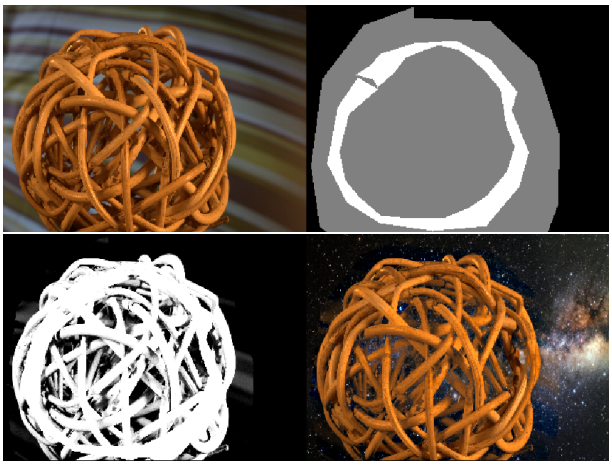

FIGURE 6.3 – première ligne : image originale et trimap dessiné par l'utilisateur (blanc : objet, noir : fond, gris : inconnu). Deuxième ligne : estimations du canal α par Matting Bayesien local et composition.

Nous avons implémenté l'algorithme de Matting bayésien de manière locale. L'utilisateur dessine à main levée une Trimap en indiquant une région du fond et une région de l'objet à extraire et le reste de l'image est considéré comme inconnu. Notons aussi que le traitement a été effectué de manière locale pour améliorer le résultat dans le cas de structures complexes. La figure 6.3

montre un exemple de résultat obtenu. Nous constatons que la méthode permet d'extraire tous les détails de l'objet et du fond pour une zone inconnue occupant plus de la moitié de l'image.

2 Composition et retouche

La composition est le problème inverse de l'alpha matting. Cette technique consiste en la combinaison de régions images S et T issues de sources multiples dans une seule image I de manière convaincante.
S et T peuvent être extraits par Alpha Matting, mais cela requiert beaucoup de travail de la part de l'utilisateur. Il est donc plus facile de commencer par une sélection grossière de S.
Le but est de trouver un masque M, spécifiant les zones occupées par les images S et T dans I, comme illustré sur la figure 6.4.

FIGURE 6.4 – Illustration de la composition de régions d'images S et T à l'aide d'un masque M.

L'équation de Matting est utilisée pour créer une composition convaincante I dans laquelle l'image S est superposée au-dessus de l'image T :

$$I = M.S + (1 - M).T \tag{6.16}$$

L'objectif est d'obtenir un masque M qui permet d'obtenir un résultat de composition avec des contours invisibles dans la région de transition entre S et T. la figure 6.5 montre un exemple réalisé avec un logiciel de retouche.

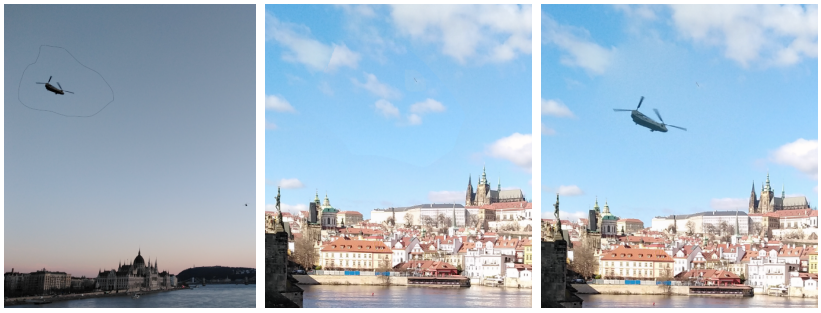

FIGURE 6.5 – Exemple de composition de deux régions images.

Une solution simple est de calculer I dans la région de transition comme une moyenne pondérée de T et S en fonction de la distance. La figure 6.6 montre un exemple de transition linéaire.

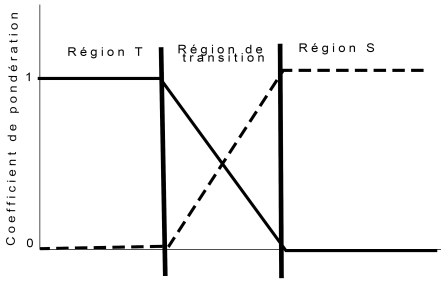

FIGURE 6.6 – Transition linéaire entre les régions S et T.

L'inconvénient de cette méthode est le choix de la largeur de transition. Si la transition est étroite, les contours de la transition seront visibles et si la transition est large, les détails seront perdus.
Dans [BA83], les auteurs proposent de travailler dans le domaine fréquentiel et d'appliquer des transitions larges pour les basses fréquences et des transitions étroites pour les hautes fréquences. Ce principe peut être appliqué en convoluant les images avec des gaussiennes de différentes tailles et en calculant les différences des gaussiennes (Laplacienne). Chaque image obtenue peut être vue comme une représentation fréquentielle à une certaine échelle : plus la taille des gaussiennes est grande, plus l'image est floue (basse fréquence).

3 Restauration et Inpainting

La restauration et l'Inpainting d'images et de la vidéo font partie des outils les plus utilisés dans l'industrie cinématographie et les effets visuels.

3.1 Restauration

Le but de cette section est de connaitre le modèle de dégradation des images et comment ajouter des informations à priori afin d'améliorer les images.
La restauration d'images utilise le modèle de dégradation de la figure 6.7.

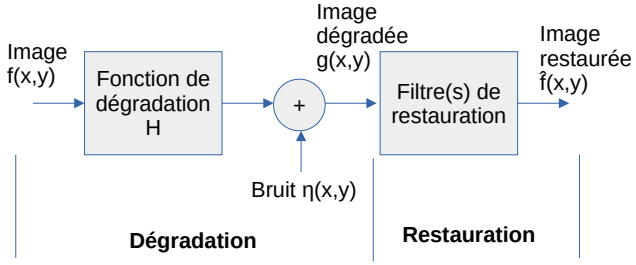

FIGURE 6.7 – Modèle de dégradation et de restauration

avec : f : image avant dégradation,

H : filtre de dégradation (exemples : flou de mise au point ou flou de mouvement),
n : bruit qui s'ajoute à l'image après la dégradation (exemples : bruits capteur caméra),
g : image obtenue après dégradation et bruitage.
La restauration consiste à obtenir, à partir de g, une image $\hat{f}$ qui soit la plus proche possible de f. L'idée est d'inverser le processus de bruitage et de dégradation.
Notons que nous distinguons la restauration de l'amélioration des images (image enhancement) qui ne cherche pas une image qui soit la plus proche de l'originale, mais seulement une image visuellement meilleure (netteté, clarté, etc.).
Formulation du modèle : La dégradation est modélisée par la formule :

$$g(x, y) = f(x, y) * h(x, y) + \eta(x, y) \tag{6.17}$$

avec $*$ le produit de convolution.
Bruits additifs et multiplicatifs : Dans le modèle présenté, le bruit est additif. Il ne s'agit pas du seul bruit possible, car dans certaines caméras le bruit est multiplicatif. Il existe d'autres types de bruits, mais les plus courants sont les bruits additifs et multiplicatifs. Une des particularités du bruit multiplicatif est que la quantité de bruit ajoutée dépend du signal f : si la valeur pixel est faible, le bruit ajouté sera faible comparé au cas où la valeur pixel est grande. Il existe un moyen de transformer le bruit multiplicatif en bruit additif. Pour cela, il suffit d'appliquer la fonction log ($log(a.b) = log(a) + log(b)$).
Nous travaillons donc avec une version logarithmique de l'image. L'image finale après traitement est récupérée en appliquant une fonction exponentielle. À cause de cette astuce, la plupart des études sur la restauration considèrent un bruit additif. L'inconvénient de cette méthode est que si le bruit n'est pas complètement éliminé, l'exponentiel risque de l'intensifier encore plus.
Nous découvrons dans ce qui suit les différents types de bruit.

Types de bruits

Pour connaitre le type de bruit à partir de l'image observée, nous calculons une probabilité de bruit.

Bruit gaussien :
la formule de distributions du bruit gaussien, affichée sur la figure 6.8 est la suivante :

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(z-\overline{z})^2}{2\sigma^2}}, \tag{6.18}$$

avec σ l'écart type du bruit et $\overline{z}$ la moyenne du bruit.
En pratique, il n'existe pas un système physique qui produit un bruit gaussien.

Toutefois, il s'agit du modèle le plus utilisé dans les systèmes de traitement d'images et de la vidéo. La raison est que d'un côté, il permet des bonnes approximations des autres types de bruit, particulièrement lorsque nous observons l'image localement sur des petites régions, et d'un autre côté, il permet de simplifier les calculs.

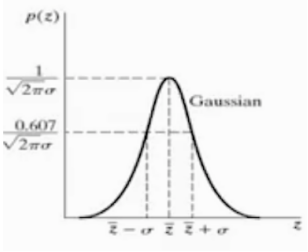

FIGURE 6.8 – Distribution gaussienne

Bruit de Rayleigh :
À la différence du bruit gaussien, le bruit de Rayleigh existe dans le monde réel dans des applications telles que l'imagerie sous-marine ou l'imagerie par résonance magnétique.
la formule de distributions du bruit de Rayleigh, affichée sur la figure 6.9 est la suivante :

$$p(z) = \begin{cases} \frac{2}{b}(z-a)e^{-\frac{(z-a)^2}{b}}, & \text{si } z \geq a. \\ 0, & \text{sinon.} \end{cases} \tag{6.19}$$

La moyenne et l'écart type s'écrivent :

$$\begin{aligned} \overline{z} &= a + \sqrt{\frac{\pi b}{4}} \\ \sigma^2 &= \frac{b(4-\pi)}{4} \end{aligned} \tag{6.20}$$

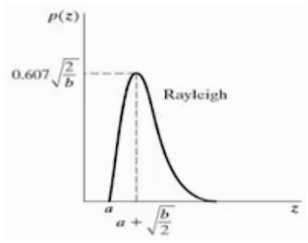

FIGURE 6.9 – Distribution Rayleigh

Bruit exponentiel :
Ce type de bruit peut être dû à un appareil (caméra) ou à un traitement

(quantification). Sa distribution, affichée sur la figure 6.10, s'écrit :

$$p(z) = \begin{cases} ae^{az}, & \text{si } z \geq 0. \\ 0, & \text{sinon.} \end{cases} \tag{6.21}$$

La moyenne et l'écart type s'écrivent :

$$\begin{aligned} \overline{z} &= \frac{1}{a} \\ \sigma^2 &= \frac{1}{a^2} \end{aligned} \tag{6.22}$$

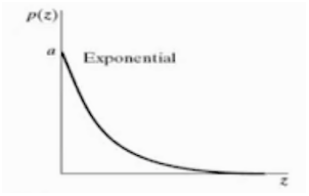

FIGURE 6.10 – Distribution exponentielle

Bruit uniforme :
Il s'agit d'un bruit ayant une valeur constante sur un certain intervalle et nulle ailleurs. Sa distribution, affichée sur la figure 6.11, s'écrit :

$$p(z) = \begin{cases} \dfrac{1}{b-a}, & \text{si } a \leq z \leq b. \\ 0, & \text{sinon.} \end{cases} \tag{6.23}$$

avec $\overline{z} = \dfrac{a+b}{2}$.

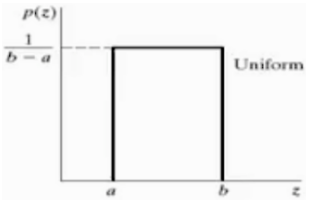

FIGURE 6.11 – Distribution uniforme

Les bruits exponentiel et uniforme sont utilisés pour modéliser le bruit de quantification (différence entre la vraie valeur dans un intervalle et la valeur représentative).

Bruit impulsionnel (poivre et sel) :
L'idée est que, avec une certaine probabilité, le pixel change complètement de valeur (par exemple, blanc) et avec une autre probabilité, il change à une autre valeur (par exemple le noir). À la différence du bruit gaussien qui affecte tous

les pixels, le bruit impulsionnel va affecter certains pixels, mais pas d'autres. Mais les pixels affectés le seront tous de la même façon (blanc ou noir). Ce bruit peut être utilisé pour modéliser les pixels défectueux de la caméra avec une probabilité faible : bruit poivre (noir) correspond aux pixels brulés.

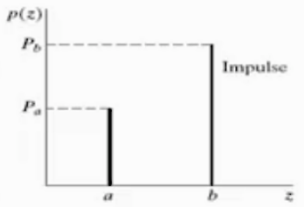

FIGURE 6.12 – Distribution uniforme

Nous présentons dans la suite comment estimer ces bruits à partir de l'image.

Estimation du bruit

Pour illustrer l'impact de chaque type de bruit, considérons une image très simple et l'histogramme correspondant :

FIGURE 6.13 – Histogramme d'image simple

L'histogramme est constitué de trois fonctions deltas : une pour chaque région image (blanc, gris, noir).
La figure suivante montre l'effet des bruits gaussien et Rayleigh sur l'histogramme :

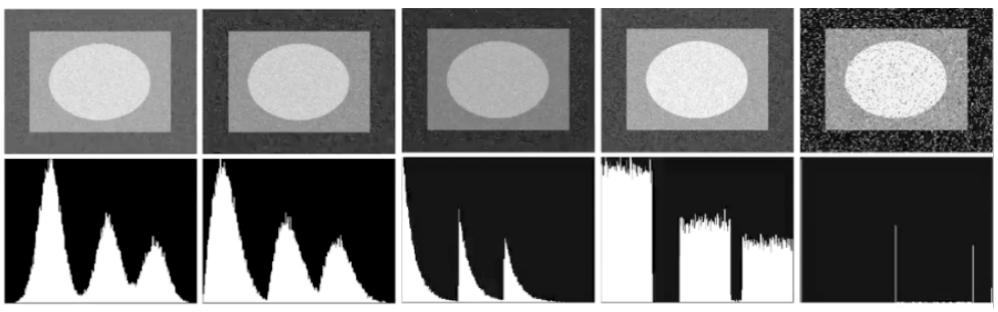

FIGURE 6.14 – Illustration des bruits gaussiens, Rayleigh, exponentiel, uniforme et impulsionnel

Nous constatons que la forme de la distribution du bruit se reflète dans l'histogramme. Cela est très utile pour estimer le type de bruit. En effet, il est très difficile de connaitre le type de bruit en observant simplement l'image, d'où l'utilisation de l'histogramme. Notons que les formes des distributions ne sont pas parfaites puisqu'il s'agit uniquement de probabilités.
Pour estimer le bruit, nous faisons dans un premier temps l'hypothèse simple qu'une petite région de l'image est uniforme et donc que tous les pixels de l'image originale sans bruit ont la même valeur.
L'histogramme d'une petite région nous permet de détecter visuellement le type de bruit :

FIGURE 6.15 – Détection visuelle du bruit à partir d'une région locale

Pour détecter le type bruit automatiquement (sans observer l'histogramme), il suffit de tester un modèle gaussien et d'ajuster ses paramètres à la distribution avec des outils d'ajustement de la courbe, afin qu'il colle de la meilleure façon. Il faut faire la même chose avec les distributions Rayleigh, uniforme, etc. Ensuite, nous sélectionnons le modèle avec la plus faible erreur par rapport à la distribution. Il ne s'agit peut-être pas de la distribution exacte, mais cela nous donne une approximation suffisante.
Notons que l'opération doit être effectuée pour chaque région locale de l'image, sauf si nous sommes certains que l'image entière est affectée par le même type de bruit, dans ce cas, il suffit de sélectionner quelques régions et de faire la moyenne des paramètres de distribution de toutes les estimations.
Généralement, nous utilisons des régions locales de très petite taille pour augmenter la probabilité que l'hypothèse de région uniforme soit vérifiée. L'inconvénient est que nous n'obtenons pas assez de pixels pour faire des statistiques correctes (moyenne, écart type, etc.).
Il est donc préférable d'utiliser des régions locales plus larges, qui peuvent contenir des régions différentes, et d'utiliser des techniques permettant de détecter des multiples distributions dans l'histogramme (mixture de gaussienne, espérance-maximisation, clustering, etc.).

Suppression du bruit

La connaissance des paramètres et du type de bruit peut nous aider à choisir le filtre adéquat. Par exemple, pour un bruit gaussien de moyenne nulle ou un bruit uniforme, nous pouvons utiliser un filtre moyen, dans le cas d'un bruit impulsionnel, nous utilisons un filtre médian, etc.

Le filtrage peut être amélioré en utilisant l'estimation pour vérifier la qualité de la restauration.
Considérons que l'image est affectée uniquement par le bruit et omettons le terme de dégradation :

$$g(x, y) = f(x, y) + \eta(x, y) \tag{6.24}$$

Nous cherchons une estimation $\hat{f}$ qui soit la plus proche possible de l'image originale f.
L'idée est que l'erreur entre l'estimation $\hat{f}$ et l'image observée g doit correspondre au bruit estimé. Si le type de bruit estimé est différent de la distribution de l'erreur, alors soit l'estimation du bruit est erronée, soit le débruitage n'est pas correcte.
L'erreur utilisée dépend du type de bruit estimé : erreur quadratique moyenne (MSE) pour un bruit gaussien, différence absolue pour un bruit exponentiel, etc.
Dans ce qui suit, nous allons inclure la fonction de dégradation.

Fonction de dégradation

Nous allons voir maintenant comment estimer la fonction de dégradation h, appelée aussi « blurring function ». La tâche est beaucoup plus complexe que l'estimation de bruit. Commençons par considérer que nous n'avons pas de bruit, le modèle de dégradation s'écrit :

$$g(x, y) = f(x, y) * h(x, y) \tag{6.25}$$

Appliquons la transformée de Fourier à ce modèle :

$$G(u, v) = F(u, v).H(u, v) \tag{6.26}$$

Rappelons que dans le domaine de Fourier, la convolution se transforme en multiplication.
Filtrage inverse : La connaissance de H nous permet donc de calculer F :

$$F(u, v) = \frac{G(u, v)}{H(u, v)} \tag{6.27}$$

Il suffit ensuite d'effectuer la transformée de Fourier inverse (filtrage inverse). Le problème est que rien ne garantit que $H(u, v)$ est toujours différent de zéro. Nous allons découvrir les types de fonction de dégradation H les plus communes et comment les estimer.
Flou : Le flou peut être modélisé par la convolution de l'image originale par une fonction gaussienne. Supposons que h est une gaussienne $G(0, \sigma)$ de moyenne nulle et d'écart type σ. Nous avons donc :

$$g(x, y) = f(x, y) * G(x, y) \tag{6.28}$$

Si nous savons que notre caméra floute l'image, il suffit d'utiliser en entrée une image de fonction $\delta(x, y)$ afin d'estimer le flou G :

$$G = \delta(x, y) * G \tag{6.29}$$

Il s'agit d'une simple calibration de la caméra.
La fonction gaussienne peut aussi être utilisée pour d'autres types de dégradations, telle que la turbulence des images aériennes.
Flou de mouvement : C'est un type de flou très fréquent qui se produit lorsqu'un objet (ou la caméra) se déplace rapidement. Étant donné que la caméra effectue une intégration de la lumière incidente de l'objet sur une certaine durée T, l'image observée s'écrit :

$$g(x, y) = \int_0^T f(x - x(t), y - y(t))dt, \tag{6.30}$$

avec $x(t)$ et $y(t)$ le déplacement du pixel pendant le mouvement. C'est équivalent à additionner plusieurs images de l'objet pendant son déplacement. Nous cherchons donc à estimer une seule image $f(x, y)$ à partir de plusieurs images inconnues. Il existe des méthodes avancées de "Sparse modeling", mais nous allons présenter ici le filtre de Wiener qui est une méthode plus simple pour résoudre ce problème en inversant le filtre et le bruit.
Filtre de Wiener : Il s'agit d'un filtre de restauration qui donne de bons résultats sous certaines conditions. L'idée de base est que nous cherchons à minimiser l'erreur quadratique moyenne (MSE : Mean squared error) entre la reconstruction et le signal original inconnu : $E[(f(x, y) - \hat{f}(x, y))^2]$. Étant donné que nous utilisons l'espérance E, nous n'avons donc pas besoin de connaitre le signal original f mais uniquement des statistiques sur f.
Après application de la dérivée pour effectuer la minimisation et passage dans le domaine de Fourier, nous obtenons :

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{H^2(u, v) + \dfrac{S_\eta}{S_F}} \right] G(u, v) \tag{6.31}$$

Le premier terme entre crochets correspond au filtre de Wiener que nous multiplions par l'observation G pour obtenir l'estimation $\hat{F}$ de l'image originale dans le domaine de Fourier. H est le filtre de dégradation estimé (flou gaussien, flou de mouvement, etc.), H^* est le conjugué complexe de H, S_η le spectre de puissance du bruit et S_F le spectre de puissance du signal. Rappelons que le spectre de puissance d'une fonction est la magnitude de la transformée de Fourier de sa fonction de corrélation. Le terme S_η / S_F est difficile à estimer et le plus souvent nous le remplaçons par une constante K :

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{H^2(u, v) + K} \right] G(u, v) \tag{6.32}$$

Applications :
Nous allons appliquer et comparer le filtre inverse et le filtre de Wiener dans les cas de flou gaussien et de flou de mouvement.
Nous considérons l'image originale et l'image floue de la figure 6.16.

FIGURE 6.16 – Image originale et flou gaussien

En supposant le filtre de flou gaussien H connu, nous pouvons appliquer le filtrage inverse pour estimer F à partir de l'image observée G :

$$F = \frac{G}{H} \tag{6.33}$$

Nous obtenons l'image de la figure 6.17.

FIGURE 6.17 – Filtrage inverse du flou gaussien

La mauvaise qualité des détails de l'image est due au fait que le filtre gaussien à des valeurs très proches de zéro.
L'application du filtre de Wiener donne un résultat largement meilleur comme le montre la figure 6.18.

FIGURE 6.18 – Filtrage de Wiener du flou gaussien

Notons que la valeur de K a été choisie empiriquement.
Considérons une image dégradée par un flou de mouvement et un bruit additif. Nous supposons le filtre de flou de mouvement H connu pour illustrer la

FIGURE 6.19 – Filtrage inverse du flou de mouvement

supériorité du filtre de Wiener par rapport au filtrage inverse.
Dans le cas d'un faible bruit, nous obtenons le résultat de la figure 6.19.

Nous constatons que le filtre de Wiener donne un résultat parfait, étant donné que H est connu et que nous sélectionnant la meilleure constante K. De l'autre côté, le filtrage inverse donne un résultat de faible qualité malgré la connaissance de H à cause du bruit.
Pour une quantité de bruit plus importante, nous obtenons le résultat suivant de la figure 6.20.

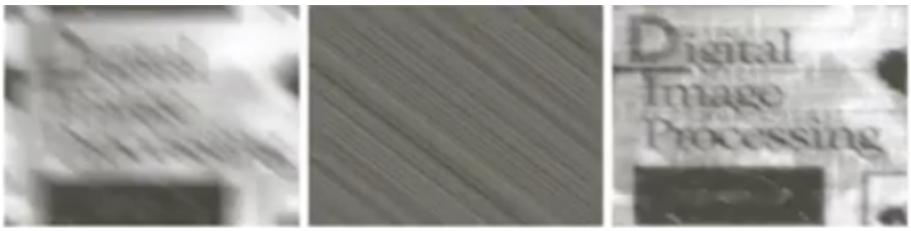

FIGURE 6.20 – Filtrage de Wiener du flou de mouvement

Le filtrage inverse donne un résultat qui dégrade tous les détails de l'image observée. En effet, il restaure uniquement le mouvement (H connu), mais pas les détails à cause de la quantité de bruit importante. De l'autre côté, avec le filtre de Wiener, la visibilité a été améliorée.
Le filtre de Wiener malgré sa simplicité reste un outil très performant.

3.2 Inpainting

L'Inpainting d'image est l'art de modifier une image de façon non détectable. La figure 6.21 montre une image dégradée et la restauration effectuée manuellement (pas avec un logiciel) par des professionnels :

Pour un observateur quelconque dans un musée, il s'agit de l'image originale. Certains experts peuvent toutefois détecter les régions retouchées.
Le but de l'Inpainting est donc de faire croire à l'observateur que l'image ou la vidéo modifiée est l'originale.
Le principe consiste à remplir les régions détériorées avec des informations provenant de régions voisines ou lointaines de l'image, voire à partir de base de données d'images similaires.

FIGURE 6.21 – Exemple de restauration manuelle

Il existe différents types de dégradations dans le cas de photographie : rayures, plies, taches, etc.
Parfois le but de l'inpainting est de supprimer des objets de l'image. Toutefois, il n'est pas toujours possible de remplacer des zones image de manière intelligente et une intervention de l'utilisateur est nécessaire. L'automatisation de cette tâche est possible dans certains cas avec des techniques d'apprentissage profond si nous fournissons à l'ordinateur beaucoup d'informations (dataset large). Mais généralement l'Inpainting n'a pas un niveau de connaissance élevé pour savoir avec quel objet remplacer la zone manquante dans l'image.
Les algorithmes d'Inpainting sont fondés sur des équations aux dérivées partielles EDP.
L'algorithme utilise à son entrée l'image à retoucher et un masque des zones à remplir.
Les algorithmes par EDP s'inspirent des méthodes de restauration utilisées par les conservateurs dans les instituts de l'art. Le principe est illustré dans la figure 6.22. Les artistes commencent par dessiner les contours manquants. Ensuite, ils propagent les couleurs à l'intérieur des régions. Enfin, ils ajoutent du bruit pour que l'image apparaisse plus naturelle.

FIGURE 6.22 – Principe de restauration utilisé par les artistes.

L'idée de base est qu'il existe une continuité des contours et des couleurs.

Soit L l'information que nous voulons propager et $\vec{N}$ la direction de propagation, que nous allons définir dans la suite.
Le gradient ∇L est projeté sur la direction de propagation N comme le montre la figure 6.23 et nous voulons qu'à la fin du processus d'Inpainting, ∇L soit perpendiculaire à N afin que l'information ne change pas de direction, ce qui se traduit par l'équation :

$$\nabla L \bullet \vec{N} = 0, \tag{6.34}$$

FIGURE 6.23 – Direction de propagation de l'extérieur de la région manquante

La solution par EDP est :

$$\nabla L \bullet \vec{N} = \frac{\partial I}{\partial t}, \tag{6.35}$$

L'image I est modifiée de façon itérative. Lorsque la variation est très faible, nous arrêtons le processus.
Nous voulons que l'information soit lisse, c'est-à-dire, sans grand saut dans l'intensité image qui sera détectable par l'observateur. L'opérateur le plus simple est le Laplacien (dérivée seconde) :

$$L = \Delta I \tag{6.36}$$

Nous voulons que les contours restent continus, nous pouvons choisir comme direction de propagation la perpendiculaire aux contours :

$$\vec{N} = \nabla^T I \tag{6.37}$$

Nous obtenons donc l'équation d'Inpainting suivante :

$$\frac{\partial I}{\partial t} = \nabla(\Delta I) \bullet \vec{\nabla^T} I \tag{6.38}$$

D'autres modèles d'information L et de direction de propagation N plus avancées, ainsi que d'autres types d'approches par synthèse de texture ou vidéo, sont présentés dans [Hal12].

4 Conclusion

Dans ce chapitre, nous avons présenté divers algorithmes très utilisés en post-production des VFX : séparation d'objet de l'arrière-plan, composition des images et restauration des images. L'intérêt de ces algorithmes pour notre étude sur le matchmoving et la réalité augmentée, est qu'après les traitements automatiques de suivi, d'insertion, d'animation et de rendu, l'intervention de l'utilisateur est souvent nécessaire pour améliorer et corriger l'image. Les outils présentés ici, constituent les types de manipulations les plus communes. Dans le chapitre suivant, nous allons présenter les nouvelles méthodes proposées.

Chapitre 7

Méthodes proposées

Sommaire

1 Suivi de mouvement et applications VFX

Dans [HH20a], nous avons proposé un système de suivi de mouvement, combinant différents algorithmes de vision par ordinateur, dans le but d'améliorer le processus des VFX et particulièrement la réalité augmentée 2D et 3D et la stabilisation vidéo.
Le suivi de mouvement consiste à faire la mise en correspondre des régions similaires d'images consécutives d'une vidéo acquise par une caméra en mouvement. Il existe deux types de suivi : le suivi d'objets et le suivi de points d'intérêt. Le suivi d'objets consiste à détecter et à suivre des objets connus tels que des personnes ou des voitures. Le suivi de points d'intérêt qui nous intéresse ici (voir la figure 7.1), consiste à détecter et à suivre des structures telles que des coins, des lignes ou des régions connectées similaires (blobs).

FIGURE 7.1 – Exemple de suivi des points d'intérêt dans une séquence vidéo.

Les étapes du système proposé sont illustrées sur la figure 7.2.

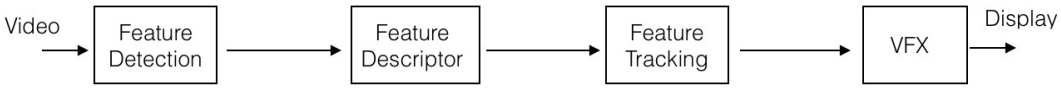

FIGURE 7.2 – Étapes de la méthode proposée.

Le système consiste à détecter, décrire et suivre les points d'intérêt dans une vidéo.
Nous avons testé les différents algorithmes de détection multi-échelles Harris-Laplace, Hessian-Laplace et Laplacien de Gaussienne, présentés dans les sections 1.2 et 1.3 du chapitre 2.
La description et l'orientation des points d'intérêt ont été réalisés par la méthode décrite dans la section 2.1 du chapitre 2.
Le suivi est réalisé par la méthode de mise en correspondance décrite dans la section 2.1 du chapitre 2.
La figure 7.3 montre quelques résultats obtenus par notre implémentation Matlab (voir l'article [HH20a] pour plus de détails). Notez que seules les Features (points d'intérêt) visibles sur l'ensemble de la séquence sont suivis et affichés. La figure 7.3 montre les Features détectés dans la première image et les Features restants à la fin de la séquence (d'une durée de 18 secondes).

Plus la séquence est longue et plus le mouvement de la caméra est important, et moins il reste de Features à la fin. Nous constatons également qu'avec les détecteurs Hessian et LoG, nous pouvons suivre plus d'éléments en raison de leur meilleure réponse aux blobs.

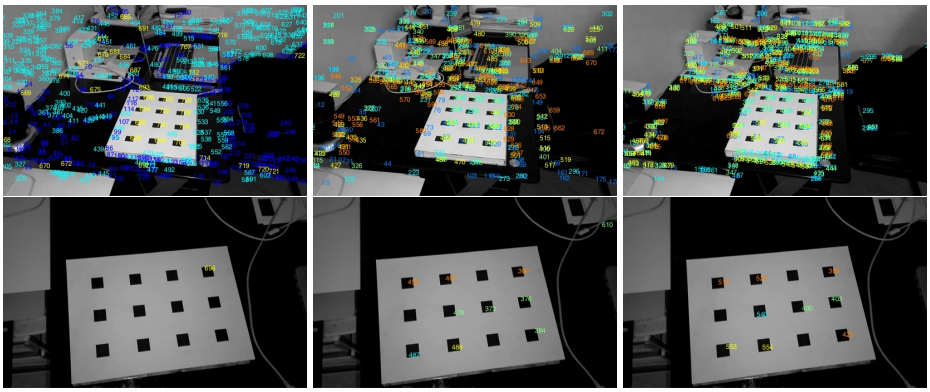

FIGURE 7.3 – Suivi de Features entre la première image (première ligne) et la dernière image (deuxième ligne) d'une séquence vidéo de 18 secondes, à l'aide des détecteurs Harris Laplace (à gauche), Hessian Laplace (au milieu) et LoG (à droite).

Une expérience a été menée pour évaluer le suivi quantitativement. Ensuite, nous avons réalisé une évaluation sur des applications d'effets visuels. Pour l'application de réalité augmentée, le résultat du suivi et la transformation de perspective sont utilisés pour insérer une image sur une surface plane d'une scène filmée par une caméra en mouvement. Nous avons également évalué le suivi pour l'insertion d'objets 3D dans une application de Matchmoving. Pour la stabilisation vidéo, le résultat de suivi a été utilisé pour estimer la transformation géométrique entre les images.

1.1 Évaluation quantitative

Nous avons procédé à une évaluation expérimentale pour mesurer la stabilité du suivi. Nous avons utilisé un téléphone mobile monté sur un trépied pour capturer une vidéo d'une scène fixe (sans aucun mouvement caméra), afin d'obtenir une vérité de terrain.
Étant donné que la caméra et la scène sont statiques, un Feature devrait idéalement être détecté dans la même position.
Nous avons évalué trois types de séquences : aucun changement d'éclairage, changement progressif d'éclairage et changement instantané d'éclairage.
Pour évaluer l'algorithme de suivi, nous calculons un score de répétabilité [TS04, HAA16] entre chaque image de la séquence et la première image, considérée comme référence :

$$S = \frac{R}{N_f}, \tag{7.1}$$

avec R le nombre de détections répétables, qui correspond au nombre de points dont la position n'a pas changé par rapport à l'image de référence. Nous pouvons introduire un intervalle de tolérance du déplacement des points. Nf est le nombre d'éléments visibles sur l'ensemble de la séquence.
La figure 7.4 montre quelques images d'une séquence utilisée pour l'expérience. Pour la première expérience sans changement d'éclairage, le seul facteur influençant la position des points entre deux images consécutives est le bruit du capteur de la caméra. Les courbes de la figure 7.5 comparent les scores de répétabilité des trois détecteurs, en utilisant des tolérances de 0 et de 2 pixels. Les figures 7.6 et 7.7 montrent les résultats des expériences avec un changement graduel de l'éclairage et avec un changement instantané de l'éclairage.

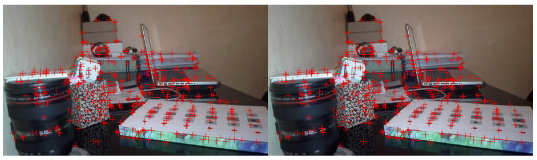

FIGURE 7.4 – Exemples d'images de l'expérience : résultat du suivi entre la première et la dernière image de la séquence.

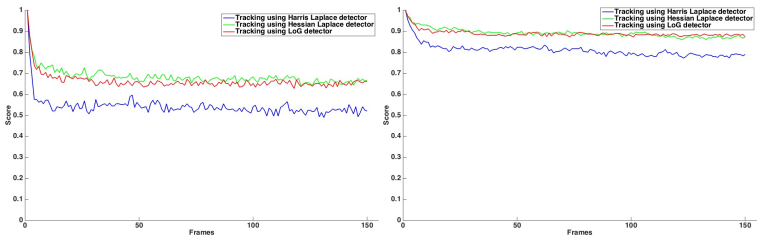

FIGURE 7.5 – Le score de répétabilité de la séquence sans changement d'éclairage, en utilisant une tolérance de position des Features de 0 pixel (à gauche) et 2 pixels (à droite).

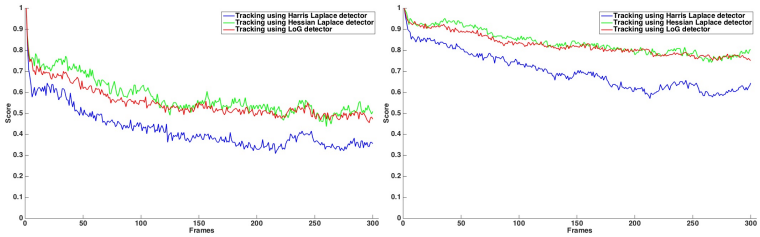

FIGURE 7.6 – Le score de répétabilité de la séquence avec un changement progressif de l'éclairage, en utilisant une tolérance de position des Features de 0 pixel (à gauche) et de 2 pixels (à droite).

Les détecteurs Hessian-Laplace et LoG présentent des résultats quasi-équivalents qui surpassent le détecteur Harris-Laplace (des résultats similaires

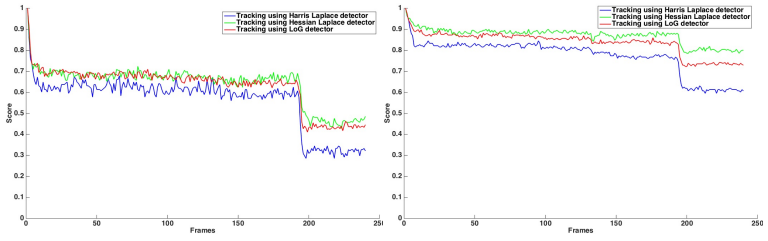

FIGURE 7.7 – Le score de répétabilité de la séquence avec changement instantané de l'éclairage, en utilisant une tolérance de position des caractéristiques de 0 (à gauche) et 2 pixels (à droite).

ont été trouvés dans [BS11]).
Dans la première expérience, l'éclairage est constant pendant toute la séquence. Le score se stabilise après un certain temps (voir explication ci-dessous). En fixant un seuil de tolérance nul (nous considérons pour le calcul du score uniquement les points dont la position n'a pas changé), nous obtenons un score allant jusqu'à environ $0,7$. Le score s'améliore au fur et à mesure que le seuil de tolérance augmente. Un score allant jusqu'à environ $0,9$ est obtenu pour une tolérance de ± 2 pixels.
Dans la deuxième expérience, l'éclairage varie de manière inhomogène dans la scène. La courbe montre que le score de détection diminue au cours du temps en raison de la variation de l'éclairage.
Enfin, dans la dernière expérience, l'éclairage varie instantanément. Les résultats de l'évaluation montrent que le score diminue instantanément au moment du changement d'éclairage.
Nous notons également, dans toutes les courbes d'évaluation, une baisse du score au début de la séquence. Ceci est dû à la sensibilité du détecteur aux Features à faible contraste. En effet, certaines Features à faible contraste vont apparaître dans les premières images de la séquence, mais pas dans les suivantes. Il est possible de réduire la valeur de cette baisse du score en ajustant les paramètres de sensibilité du détecteur, nous obtiendrons un meilleur score, mais aussi moins de Features, ce qui n'est pas pertinent pour la plupart des applications de suivi.
Les résultats de l'évaluation nous donnent une idée de la performance du suivi. Dans la suite de cet article, nous allons évaluer le suivi en le combinant avec des algorithmes VFX.

1.2 Applications VFX

Dans cette section, nous proposons d'utiliser le résultat de la détection et du suivi dans une vidéo acquise avec une caméra en mouvement pour des applications VFX (voir figure 7.2). Un avantage de la méthode proposée est son aspect modulaire. Chaque étape de détection, d'extraction des descripteurs, et de suivi, peut être remplacée par un algorithme différent afin de proposer la

solution la plus appropriée pour chaque application VFX.

Réalité augmentée

La réalité augmentée consiste à ajouter ou à améliorer des éléments dans une vidéo d'une scène réelle. Ici, nous proposons d'insérer une image 2D dans une surface plane sélectionnée par l'utilisateur. L'apparence de l'image insérée doit correspondre aux perspectives de la scène lors du déplacement de la caméra. Pour cela, le résultat de la position des Features, appartenant à la surface plane, est utilisé pour calculer la transformation de perspective (homographie) entre chaque deux images consécutives de la vidéo.
Nous commençons par détecter les Features en utilisant le détecteur Laplace Hessien multi-échelle qui, comme nous l'avons mentionné précédemment, donne le meilleur compromis entre la détection des blobs et le rejet des contours, en plus de son invariance par translation et échelle. Les descripteurs sont extraits, comme expliqué précédemment, afin d'obtenir des Features invariants par rotation et par changement de luminance. Ensuite, nous effectuons le suivi entre des images consécutives en faisant correspondre leurs descripteurs à l'aide d'une mesure SSD.
Les résultats du suivi de la première et de la dernière image de la vidéo sont affichés à l'utilisateur pour vérifier qu'il n'y a pas d'erreurs de mise en correspondance, souvent dues à des variations géométriques et photométriques importantes. Les erreurs de suivi peuvent être considérées comme des valeurs aberrantes et détectées par un algorithme tel que Ransac [FB81], qui n'est pas pertinent dans notre cas, car nous devons avoir un modèle de données, alors que nous considérons des mouvements de caméra aléatoires et inconnus. Pour un meilleur résultat de mise en correspondance, les scènes VFX sont souvent filmées dans un environnement contrôlé en ajoutant des marqueurs. Pour notre étude, nous ajoutons également des marqueurs à la surface plane. Il est à noter qu'il existe des détecteurs non génériques [RRMSMC18, GJMSMCMJ14, BAC+16, WO16] qui ne fonctionnent qu'avec des marqueurs artificiels.
Comme l'illustre l'exemple de la figure 7.8 d'une séquence vidéo d'une durée de 18 secondes, l'utilisateur dessine sur la première image un rectangle sur la surface plane, où il souhaite insérer l'image, puis sélectionne les caractéristiques appartenant à la surface plane et qui ont été bien suivies (sans erreur de correspondance).

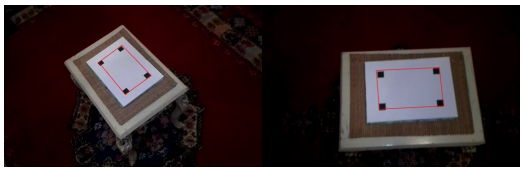

FIGURE 7.8 – La surface plane sélectionnée par l'utilisateur (à gauche) et la transformation de perspective correspondante (à droite) après l'estimation et l'application de l'homographie en utilisant le résultat du suivi.

L'étape suivante est l'estimation de l'homographie entre les images consécutives.

Homographie : algorithme DLT

Nous considérons $\boldsymbol{x_i}$ la position d'un Feature i appartenant à la surface plane et $\boldsymbol{x_i}'$ la position du même Feature dans l'image suivante. La transformation en perspective de $\boldsymbol{x_i}$ en $\boldsymbol{x'_i}$ correspond à la matrice H, 3×3, telle que :

$$\boldsymbol{x_i} = H.\boldsymbol{x'_i} \tag{7.2}$$

où :

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}, \tag{7.3}$$

et les coordonnées homogènes $\boldsymbol{x_i}$:

$$\boldsymbol{x_i} = \begin{bmatrix} x_i & y_i & w_i \end{bmatrix}^T \tag{7.4}$$

Pour $w_i = 1$, $\begin{bmatrix} x_i & y_i \end{bmatrix}^T$ correspond aux coordonnées de l'image. Par conséquent, l'équation est définie à un facteur près. C'est-à-dire qu'en multipliant H par un facteur, l'égalité reste valable. Cela réduit le degré de liberté de H de 9 à 8. Nous estimons H avec l'algorithme de transformation linéaire directe DLT [HZ03] que nous allons présenter dans ce qui suit. Notez qu'il existe aussi des méthodes d'estimation de l'homographie utilisant l'apprentissage profond [DMR16].
Nous commençons par écrire l'équation d'homographie sous la forme d'un produit scalaire :

$$\boldsymbol{x'_i} \times H\boldsymbol{x_i} = 0 \tag{7.5}$$

Nous notons les lignes de H :

$$\boldsymbol{h}^1 = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}^T, \tag{7.6}$$

$$\boldsymbol{h}^2 = \begin{bmatrix} h_4 & h_5 & h_6 \end{bmatrix}^T, \tag{7.7}$$

et :

$$\boldsymbol{h}^3 = \begin{bmatrix} h_7 & h_8 & h_9 \end{bmatrix}^T \tag{7.8}$$

L'équation précédente devient :

$$\boldsymbol{x'_i} \times H\boldsymbol{x_i} = \begin{bmatrix} y'_i \boldsymbol{h}^{3T}\boldsymbol{x_i} - w'_i \boldsymbol{h}^{2T}\boldsymbol{x_i} \\ w'_i \boldsymbol{h}^{1T}\boldsymbol{x_i} - x'_i \boldsymbol{h}^{3T}\boldsymbol{x_i} \\ x'_i \boldsymbol{h}^{2T}\boldsymbol{x_i} - y'_i \boldsymbol{h}^{1T}\boldsymbol{x_i} \end{bmatrix} \tag{7.9}$$

Et comme :

$$\boldsymbol{h}^{jT}\boldsymbol{x_i} = \boldsymbol{x_i}^T \boldsymbol{h}^j, \tag{7.10}$$

nous obtenons :

$$\begin{bmatrix} \mathbf{0}^T & -w'_i\boldsymbol{x_i}^T & y'_i\boldsymbol{x_i}^T \\ w'_i\boldsymbol{x_i}^T & \mathbf{0}^T & -x'_i\boldsymbol{x_i}^T \\ -y'_i\boldsymbol{x_i}^T & x'_i\boldsymbol{x_i}^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{h}^1 \\ \boldsymbol{h}^2 \\ \boldsymbol{h}^3 \end{bmatrix} = 0 \tag{7.11}$$

Nous écrivons cette équation sous la forme :

$$A_i\boldsymbol{h} = 0 \tag{7.12}$$

où :

$$h = \begin{bmatrix} \boldsymbol{h}^1 \\ \boldsymbol{h}^2 \\ \boldsymbol{h}^3 \end{bmatrix} \tag{7.13}$$

et

$$A_i = \begin{bmatrix} \mathbf{0}^T & -w'_i\boldsymbol{x_i}^T & y'_i\boldsymbol{x_i}^T \\ w'_i\boldsymbol{x_i}^T & \mathbf{0}^T & -x'_i\boldsymbol{x_i}^T \\ -y'_i\boldsymbol{x_i}^T & x'_i\boldsymbol{x_i}^T & \mathbf{0}^T \end{bmatrix} \tag{7.14}$$

Nous pouvons remarquer que la troisième ligne de A_i peut être obtenue, à une échelle près, à partir de la somme de la première ligne multipliée par x'_i et de la deuxième ligne multipliée par y'_i. La matrice A_i est donc de rang 2. L'équation est donc équivalente à :

$$\begin{bmatrix} \mathbf{0}^T & -w'_i\boldsymbol{x_i}^T & y'_i\boldsymbol{x_i}^T \\ w'_i\boldsymbol{x_i}^T & \mathbf{0}^T & -x'_i\boldsymbol{x_i}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{h}^1 \\ \boldsymbol{h}^2 \\ \boldsymbol{h}^3 \end{bmatrix} = 0 \tag{7.15}$$

Par conséquent, nous considérons la nouvelle matrice A_i de dimensions 2×9 :

$$A_i = \begin{bmatrix} \mathbf{0}^T & -w'_i\boldsymbol{x_i}^T & -y'_i\boldsymbol{x_i}^T \\ w'_i\boldsymbol{x_i}^T & \mathbf{0}^T & -x'_i\boldsymbol{x_i}^T \end{bmatrix} \tag{7.16}$$

Chaque paire de Features similaires $\boldsymbol{x_i}$ et $\boldsymbol{x'_i}$ nous donne deux équations. Par conséquent, pour résoudre le problème de l'homographie, nous utilisons quatre paires de Feature qui nous fournissent un ensemble d'équations :

$$A\boldsymbol{h} = 0 \tag{7.17}$$

La matrice A est construite à partir des lignes de la matrice A_i de chaque paire d'éléments.
Comme la position des Features est sujette au bruit, nous utilisons plus de quatre Features, et nous cherchons une solution approximative correspondant à la meilleure transformation projective H, qui minimise la fonction de coût suivante :

$$min_{\boldsymbol{h},||\boldsymbol{h}||=1} = ||A\boldsymbol{h}|| \tag{7.18}$$

Nous utilisons la contrainte $||\boldsymbol{h}|| = 1$ afin d'éviter la solution triviale $\boldsymbol{h} = 0$. La solution est obtenue à l'aide de la méthode des moindres carrés en calculant la SVD de A :

$$A = UDV^T, \tag{7.19}$$

et en déduisant $\boldsymbol{h}$ comme la dernière colonne de V.
Les coordonnées de l'image (x_i, y_i) ont des valeurs qui varient de 1 à des centaines ou des milliers, tandis que $w_i = 1$. Il est donc nécessaire de normaliser les valeurs de A pour avoir des ordres de grandeur similaires [CB03] avant d'appliquer l'algorithme DLT. Nous commençons par calculer pour chaque ensemble de Features x_i (et x'_i) la transformation T (et T'), consistant en une translation et une mise à l'échelle de x_i en x^*_i, de sorte que le centroïde de l'ensemble des Features x^*_i soit $(0, 0)$ et que la distance moyenne à l'origine, soit $\sqrt{2}$:

$$T = \begin{bmatrix} s^{-1} & 0 & 0 \\ 0 & s^{-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\overline{m}_1 \\ 0 & 1 & -\overline{m}_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.20}$$

où $\overline{m}_1$ et $\overline{m}_2$ sont les coordonnées du centroïde de $\boldsymbol{x_i}$:

$$\overline{m}_1 = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{7.21}$$

$$\overline{m}_2 = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{7.22}$$

et

$$s = (\frac{1}{2n} \sum_{i} (x_i - \overline{m}_1)^2 + (y_i - \overline{m}_2)^2)^{0.5} \tag{7.23}$$

La transformation T' de x'_i en x'^*_i est calculée de la même façon. Ensuite, nous appliquons l'algorithme DLT précédent pour trouver H^*. Enfin, nous dénormalisons H^* pour trouver H :

$$H = T'^{-1} H^* T \tag{7.24}$$

Dans notre application, nous estimons L'homographie H pour chaque deux images consécutives de la vidéo. Nous appliquons ensuite ces homographies au masque d'insertion choisi par l'utilisateur afin de suivre le masque tout au long de la séquence.
Enfin, nous calculons l'homographie entre l'image rectangulaire à insérer et le masque suivi et l'appliquons afin de déformer l'image rectangulaire à la position du masque. La figure 7.9 montre un exemple de résultats.

Comme nous l'avons souligné au début de cette section, plus la durée de la vidéo est longue et plus le mouvement de la caméra est important, et moins les Features suivis seront nombreux à la fin. Cela affecte également la stabilité du suivi : la position des Features suivis sera moins précise en raison de l'accumulation d'erreurs. Notre défi était de réussir à suivre des éléments sur une plus longue durée. Les paramètres permettant d'avoir plus de Features sont le seuil de détection, le seuil SSD de la mise en correspondance et la taille de la zone de recherche. Les valeurs de ces paramètres dépendent de chaque vidéo. Les figures 7.10 montrent un exemple de résultat sur une vidéo relativement

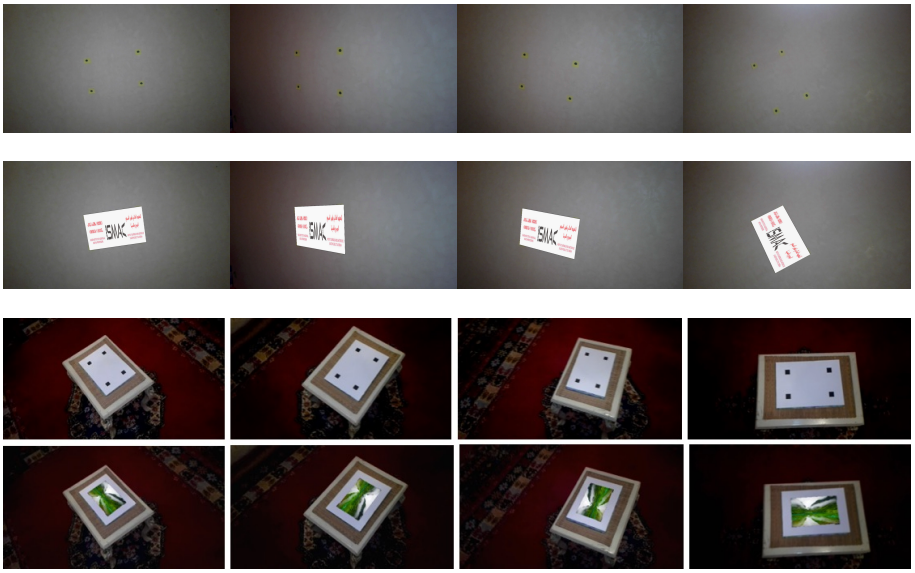

FIGURE 7.9 – Images de la vidéo originale (première et troisième lignes) et le résultat de l'insertion des images (deuxième et quatrième lignes). Les durées de la vidéo sont respectivement de 20 et 13 secondes.

longue d'une durée de 18 secondes avec un changement de perspective important. Nous constatons que de petites erreurs de position peuvent être introduites. Notez également qu'afin d'obtenir un résultat d'homographie plus stable, nous avons utilisé plus de marqueurs : plus que le nombre minimal de quatre points nécessaires à l'homographie. Un autre exemple est présenté sur la figure 7.11.

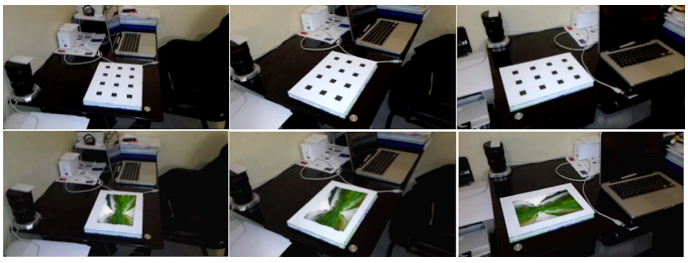

FIGURE 7.10 – Exemple d'insertion d'une image dans une vidéo de 18 secondes avec un changement de perspective important.

Matchmoving

Le matchmoving consiste à insérer des objets 3D dans la vidéo d'une scène réelle. Pour cela, il faut déterminer pour chaque image de la vidéo, la position et l'orientation de la caméra dans un référentiel de la scène. La première étape de matchmoving est le suivi des Features. Nous avons utilisé le résultat du notre système de suivi afin de le fournir à un logiciel de matchmoving permettant

FIGURE 7.11 – Exemple d'insertion d'une image dans une vidéo de 20 secondes.

l'estimation du mouvement de la caméra et l'insertion des objets 3D. La figure 7.12 montre un exemple de résultat. Nous constatons que la perspective des objets 3D correspond au mouvement de la caméra. Une étude plus approfondie du Matchmoving sera présentée dans la suite de ce chapitre.

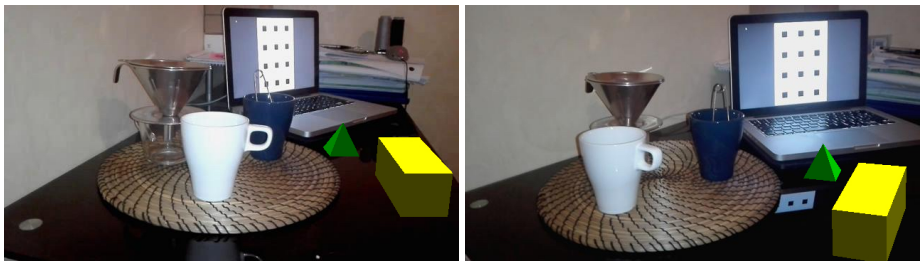

FIGURE 7.12 – Matchmoving : insertion d'objets 3D dans une vidéo en utilisant le résultat du suivi.

Stabilisation de la vidéo

Nous proposons ici une autre application du suivi de Features qui consiste à stabiliser une vidéo acquise par une caméra portable afin de corriger les vibrations de l'image ou d'ajouter certains effets artistiques à la vidéo. Nous supposons qu'entre deux images successives de la séquence, l'image subit une rotation, une mise à l'échelle et une translation. La transformation géométrique [BGPS07] s'écrit comme suit :

$$x_{i+1} = x_i.s.cos(\theta) - y_i.s.sin(\theta) + T_x \tag{7.25}$$

$$y_{i+1} = x_i.s.sin(\theta) + y_i.s.cos(\theta) + T_y \tag{7.26}$$

avec (x_i, y_i) la position de l'élément dans l'image i et (x_{i+1}, y_{i+1}) celle de son correspondant dans l'image suivante. s, θ et (T_x, T_y) correspondent respectivement aux coefficients d'échelle, de rotation et de translation. L'ensemble des Features suivis est utilisé pour estimer les différents paramètres de la transformation avec la méthode des moindres carrés.
Enfin, afin de stabiliser la vidéo, nous appliquons à chaque image la

transformation inverse : échelle $1/s$, rotation $-\theta$ et translation $(-T_x, -T_y)$. La figure 7.13 montre un exemple des résultats obtenus. Nous constatons que l'algorithme de stabilisation compense le mouvement de la caméra.

FIGURE 7.13 – Images d'une vidéo avant la stabilisation (première ligne) et après la stabilisation (deuxième ligne).

Note sur l'implémentation

Tous les algorithmes ont été implémentés en partant de zéro à l'aide de Matlab, à l'exception de l'algorithme d'homographie [Hom]. Toutes les vidéos ont été acquises avec un téléphone portable. La résolution vidéo est égale à 640×320. Empiriquement, nous trouvons que les paramètres (voir chapitre 2) qui donnent le meilleur résultat pour presque toutes les vidéos, et que nous avons utilisé pour obtenir les résultats présentés ici, sont : $\sigma_0 = 1,5$; $b = 1.4$; $k = 0.04$; $a = 1.3$; Seuil de Harris de 10^{-9} ; Seuil de Hessian Laplace de 10^{-3} ; Seuil de LoG de 10^{-1} ; Cinq échelles : $[\sigma_0.b, \sigma_0.b^2, \sigma_0.b^3, \sigma_0.b^4, \sigma_0.b^5]$; Taille du voisinage pour la correspondance de 20 pixels ; Taille des descripteurs de 81×81 ; Seuil SSD de 2 ; Taille de la région de support de $6 \times W$; Les valeurs d'entrée de l'image sont normalisées entre 0 et 1.

1.3 Conclusion

l'objectif de notre étude était de développer des approches applicables aux vidéos acquises avec une caméra mobile. L'évaluation qualitative et quantitative sur des vidéos de scènes réelles atteste de l'efficacité des algorithmes proposés. Cependant, la méthode présente des limites telles que des erreurs de mise en correspondance dues au bruit. L'intervention de l'utilisateur peut permettre d'éliminer les mauvais résultats. Cette intervention peut être évitée en utilisant un algorithme d'élimination des aberrations tel que RANSAC, sauf qu'il sera nécessaire d'ajuster ses paramètres manuellement en fonction de chaque séquence. Le suivi des Features peut être amélioré en utilisant un filtre de Kalman pour estimer la position des Features non visibles

sur l'ensemble de la séquence et qui dans notre cas ont été écartées. L'une des limites de notre algorithme de réalité augmentée est qu'il ne gère pas le cas où les objets de premier plan occultent le plan d'insertion. Enfin, notons que l'aspect modulaire de notre méthode la rend plus flexible

2 Pré-visualisation du Matchmoving

Dans [HH20b], nous avons proposé une méthode d'insertion d'un objet de synthèse 3D dans une vidéo de scène réelle. L'originalité de la méthode proposée réside dans la combinaison et l'application aux effets visuels de différents algorithmes de vision par ordinateur et d'informatique graphique. Tout d'abord, les paramètres intrinsèques et les coefficients de distorsion de la caméra sont estimés en utilisant une mire plane de calibration en damier, à l'aide de l'algorithme de Zhang. Ensuite, le dictionnaire de marqueurs AruCo et l'algorithme de détection des Features correspondant sont utilisés pour détecter les quatre coins d'un seul marqueur artificiel ajouté à la scène. Une méthode de perspective à 4 points est utilisée pour estimer la rotation et la translation de la caméra par rapport à un système de référence 3D fixé au marqueur. Le modèle de perspective de la caméra est ensuite utilisé pour projeter l'objet 3D sur le plan de l'image, tout en respectant les variations de perspective lorsque la caméra est en mouvement. L'objet 3D est éclairé avec des modèles d'ombrage (Shading) diffus et spéculaire, afin de faire correspondre l'objet à l'éclairage de la scène. Enfin, nous avons mené une expérience pour évaluer quantitativement et qualitativement la stabilité de la méthode.

Le matchmoving ou camera tracking est une méthode de réalité augmentée utilisée en effets visuels [Rad13], et qui consiste à insérer des objets de synthèse 3D dans une vidéo de scène réelle, de telle sorte que l'objet coexiste de manière cohérente avec les autres éléments, tout en respectant la géométrie et l'éclairage de la scène, comme le montre la figure 7.14.

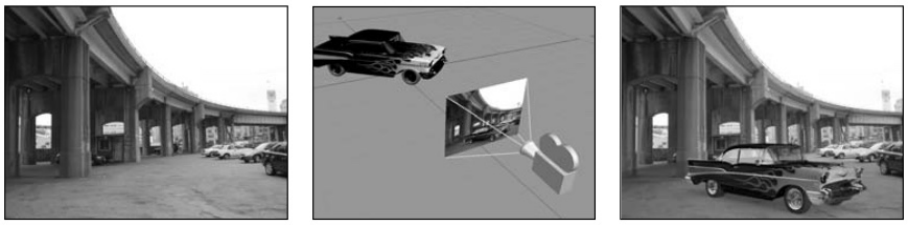

FIGURE 7.14 – Illustration du Matchmoving.

Habituellement, dans le domaine cinématographique, cet effet est obtenu en post-production. Cependant, en cas de problème lors du tournage, qu'il soit technique (éclairage ou suivi) ou artistique (position de l'objet), il est difficile

de réaliser le matchmoving en post-production sans passer beaucoup de temps à traiter la vidéo image par image ou sans refaire le tournage de la scène. Cela implique des pertes de temps et d'argent importantes. Dans ce chapitre, nous proposons une méthode permettant de faire une prévisualisation du résultat en temps réel. Cela permet de détecter et de corriger les problèmes pendant le tournage.
Le problème du matchmoving comporte des aspects géométriques et photométriques, qui sont souvent traités séparément dans la littérature.
L'estimation de la pose de la caméra (rotation et translation de la caméra) est au coeur de l'aspect géométrique. Dans [RTPS16, KK17, LHK+19], des capteurs externes (capteur inertiel, Wifi ou casque Hololens) sont utilisés à cette fin. Dans [KGC15, MYKR17, ELJ18] l'estimation est réalisée par apprentissage profond, cela résout le problème du flou de mouvement, mais nécessite la présence de textures dans l'image. Les méthodes traditionnelles sont fondées sur la détection de Features, qui sont des motifs tels que des coins ou des régions connectées similaires (blobs) qui ont la particularité d'être fiables pour le suivi (voir chapitres 2 et 4). Certains détecteurs sont invariants à l'échelle, aux transformations affines et à l'orientation [MS04]. Des marqueurs artificiels peuvent être ajoutés dans la scène afin de gérer le problème des zones non texturées et pour une détection plus robuste, grâce au code binaire "QR" unique inclus dans les marqueurs.
Pour cette étude, nous utilisons l'algorithme de détection ArUco [GJMSMCMC16] et le dictionnaire de marqueurs correspondant en raison de ses performances de détection par rapport aux autres marqueurs artificiels [RRMSMC18] et pour la rapidité de calcul. En localisant les quatre coins d'un seul marqueur, celui-ci peut être utilisé comme système de référence 3D pour l'estimation de la pose de la caméra, grâce à une méthode de perspective 4 points [Zha00].
Concernant l'aspect éclairage, les deux méthodes les plus utilisées pour le rendu sont la rastérisation [GC18] et le ray tracing [WM19], que nous avons présenté dans le chapitre 5. Nous utilisons la rastérisation en raison de la vitesse de traitement sur CPU et de la qualité de rendu suffisante pour la prévisualisation. Enfin, au lieu d'utiliser une méthode d'estimation automatique de l'illumination [HGSH+17], nous choisissons manuellement la position d'éclairage de l'objet 3D lors de l'ajustement de son aspect géométrique. Cette intervention manuelle est nécessaire puisqu'elle dépend du choix de l'utilisateur.
L'originalité de la méthode est de proposer une solution de prévisualisation du Matchmoving, en combinant tous les aspects géométriques et d'ombrage dans un seul système. D'autre part, la méthode est accessible par l'utilisation d'une seule caméra, d'un marqueur artificiel et d'algorithmes rapides d'estimation de pose et de rendu de la caméra.
Les étapes du système proposé sont illustrées sur la figure 7.4.

Tout d'abord, la caméra est calibrée afin d'estimer ses paramètres intrinsèques. Ensuite, nous procédons à la détection des coins des marqueurs artificiels afin

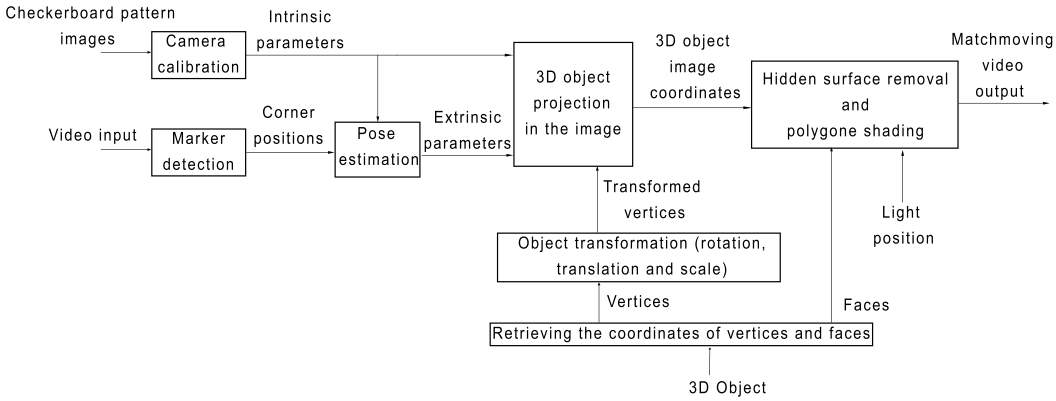

FIGURE 7.15 – Étapes de la méthode proposée.

d'estimer la pose de la caméra. Après avoir ajusté l'apparence géométrique désirée de l'objet 3D, nous le projetons sur l'image en utilisant le modèle de perspective de la caméra et les paramètres estimés de la caméra. Les faces visibles de l'objet sont calculées par un algorithme de suppression des surfaces cachées. Enfin, nous attribuons à chaque face une couleur en utilisant un modèle d'ombrage diffus et spéculaire, selon une position d'éclairage choisie par l'utilisateur.

2.1 Modèle de perspective caméra

Matrice de calibration

L'objectif est d'établir la relation entre les coordonnées pixels d'un point dans l'image et les coordonnées en mètres du point 3D correspondant dans l'espace monde [Rad13]. Nous considérons que la caméra suit un modèle sténopé. Les coordonnées 3D sont spécifiées dans un système de référence caméra $(X\ Y\ Z)$ comme indiqué sur la figure 7.16.

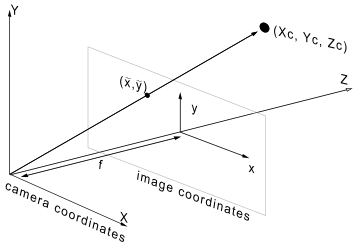

FIGURE 7.16 – Modèle sténopé et systèmes de référence caméra et image.

Nous considérons un point de la scène avec des coordonnées $\begin{bmatrix} X_c & Y_c & Z_c \end{bmatrix}^T$. La projection $\begin{bmatrix} \tilde{x} & \tilde{y} \end{bmatrix}^T$ dans l'image en fonction de la distance focale f est :

$$\tilde{x} = f\frac{X_c}{Z_c} \qquad \tilde{y} = f\frac{Y_c}{Z_c} \tag{7.27}$$

Les valeurs $\begin{bmatrix}\tilde{x} & \tilde{y}\end{bmatrix}^T$ sont des mesures physiques en mètres. Les valeurs $\begin{bmatrix}x & y\end{bmatrix}^T$ en pixels s'écrivent en fonction de la largeur d_x et de la hauteur d_y d'un pixel et du centre de l'image $\begin{bmatrix}x_0 & y_0\end{bmatrix}^T$ qui correspond à la projection de l'origine du système de référence de la caméra dans le plan de l'image :

$$x = \frac{\tilde{x}}{d_x} + x_0 \qquad y = \frac{\tilde{y}}{d_y} + y_0 \tag{7.28}$$

Nous obtenons donc :

$$x = \frac{fX_c}{d_x Z_c} + x_0 \qquad y = \frac{fY_c}{d_y Z_c} + y_0 \tag{7.29}$$

L'équation 7.29 peut être écrite sous forme matricielle :

$$\begin{bmatrix}x & y & 1\end{bmatrix}^T \sim K \begin{bmatrix}X_c & Y_c & Z_c\end{bmatrix}^T \tag{7.30}$$

avec

$$K = \begin{bmatrix}\alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1\end{bmatrix}, \tag{7.31}$$

$$\alpha_x = \frac{f}{d_x}, \tag{7.32}$$

et

$$\alpha_y = \frac{f}{d_y} \tag{7.33}$$

K est appelée la matrice de calibration et le symbole $\sim$ signifie que l'égalité est obtenue à un facteur près (en divisant le terme de droite par Z_c).

Distorsions

Les caméras utilisées dans la vie réelle sont plus complexes qu'un simple modèle sténopé. L'image souffre généralement de distorsions radiales causées par la forme sphérique de l'objectif [Zha00]. La relation entre les coordonnées $\begin{bmatrix}\tilde{x} & \tilde{y}\end{bmatrix}^T$ de l'image idéale (sans distorsion) et les coordonnées $\begin{bmatrix}\tilde{x}_{dist} & \tilde{y}_{dist}\end{bmatrix}^T$ de l'image observée (avec distorsion) est la suivante :

$$\begin{aligned}\tilde{x}_{dist} &= (1 + \kappa_1(\tilde{x}^2 + \tilde{y}^2) + \kappa_2(\tilde{x}^2 + \tilde{y}^2)^2)\tilde{x} \\ \tilde{y}_{dist} &= (1 + \kappa_1(\tilde{x}^2 + \tilde{y}^2) + \kappa_2(\tilde{x}^2 + \tilde{y}^2)^2)\tilde{y}\end{aligned} \tag{7.34}$$

Les coefficients κ_1 et κ_2 contrôlent la quantité de distorsion. Dans le cas de distorsions importantes (objectif grand angle), un troisième coefficient κ_3 peut être ajouté comme troisième ordre dans la formule polynomiale. Ce modèle de distorsion est combiné avec l'équation 7.28 afin d'obtenir un modèle en fonction des coordonnés pixels $\begin{bmatrix}x_{dist} & y_{dist}\end{bmatrix}^T$:

$$\begin{aligned}x_{dist} &= (1 + \kappa_1(\tilde{x}^2 + \tilde{y}^2) + \kappa_2(\tilde{x}^2 + \tilde{y}^2)^2)(x - x_0) + x_0 \\ y_{dist} &= (1 + \kappa_1(\tilde{x}^2 + \tilde{y}^2) + \kappa_2(\tilde{x}^2 + \tilde{y}^2)^2)(y - y_0) + y_0\end{aligned} \tag{7.35}$$

Matrice caméra

En supposant que les distorsions sont compensées (voir la section suivante), la formation de l'image peut être modélisée par l'équation 7.30. Ce modèle permet d'exprimer les coordonnées d'un point dans le système de référence de la caméra. Pour le problème du Matchmoving, les coordonnées de l'objet 3D peuvent être définies par l'utilisateur dans n'importe quel système de référence de la scène. Nous devons donc, avant d'appliquer le modèle de projection, transformer les coordonnées $\begin{bmatrix} X & Y & Z \end{bmatrix}^T$ exprimées dans n'importe quel système de référence en coordonnées $\begin{bmatrix} X_c & Y_c & Z_c \end{bmatrix}^T$ exprimées dans le système de référence de la caméra. La formule de transformation est la suivante :

$$\begin{bmatrix} X_c & Y_c & Z_c \end{bmatrix}^T = R \begin{bmatrix} X & Y & Z \end{bmatrix}^T + t \tag{7.36}$$

R et t sont les paramètres extrinsèques. R est une matrice de rotation 3×3 définie par 3 angles et t est un vecteur de translation. Ainsi, l'équation 7.30 devient :

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \sim P \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T \tag{7.37}$$

avec $P = K \begin{bmatrix} R & | & t \end{bmatrix}$. P est appelée matrice caméra. Le problème du matchmoving revient à calculer P pour chaque image de la vidéo.

2.2 Méthode proposée

Estimation de la matrice de calibration

En supposant que la distance focale de la caméra ne change pas pendant l'acquisition vidéo, nous devons estimer la matrice de calibration une seule fois au début. Pour ce faire, nous utilisons plusieurs images d'une mire en damier plane de dimensions connues, capturées depuis différents points de vue [Zha00]. La figure 7.17 montre la configuration de la calibration. Nous avons utilisé une caméra de téléphone mobile montée sur un trépied et contrôlée à distance pour une plus grande stabilité d'acquisition afin de réduire l'erreur de calibration.

En considérant que $Z = 0$ correspond au plan de la mire, l'équation 7.37 est transformée en une homographie :

$$\begin{bmatrix} x & y & 1 \end{bmatrix}^T = \lambda K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X & Y & 0 & 1 \end{bmatrix}^T \tag{7.38}$$

où r_1, r_2 et r_3 sont les colonnes de R et λ un coefficient arbitraire.
Ainsi, nous avons :

$$\begin{bmatrix} x & y & 1 \end{bmatrix}^T = \lambda K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X & Y & 1 \end{bmatrix}^T \tag{7.39}$$

Par conséquent, la formule d'homographie est la suivante :

$$\begin{bmatrix} x & y & 1 \end{bmatrix}^T = H \begin{bmatrix} X & Y & 1 \end{bmatrix}^T \tag{7.40}$$

FIGURE 7.17 – (a) : Configuration de la calibration de la caméra. (b) : Résultat de la projection d'un objet 3D dans une vidéo acquise avec une caméra mobile

.

Nous notons H la matrice d'homographie 3×3 :

$$H = \lambda K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \tag{7.41}$$

H est estimée en utilisant l'algorithme DLT (présenté précédemment) [HZ03]. Nous commençons par détecter les coins des carrés du damier à l'aide d'un algorithme de détection de Feature (Harris ou autre). Chaque coin $\begin{bmatrix} x & y \end{bmatrix}^T$ nous donne deux équations à partir de 7.40. Le vecteur $\begin{bmatrix} X & Y & 1 \end{bmatrix}^T$ correspondant est connu, puisque nous avons la distance physique entre les coins. Notez que H à un degré de liberté de 8 puisque l'équation de l'homographie est définie à un facteur près. Ainsi, nous avons besoin d'un minimum de quatre coins pour trouver H. Comme la position des coins est sujette au bruit, nous utilisons plus de quatre coins. Ensuite, la méthode des moindres carrés est utilisée pour trouver une solution approximative, par une décomposition en valeurs singulières, correspondant à la meilleure homographie

H qui minimise une fonction de coût. Une fois que nous avons H pour chaque image, nous pouvons déduire K de l'équation 7.41 : en ajoutant des contraintes sur K du fait que $r1$ et $r2$ sont orthonormaux [Zha00], l'équation 7.41 est simplifiée en une équation linéaire et l'estimation de K est effectuée à nouveau avec l'algorithme DLT.

Estimation des coefficients de distorsion

Une fois que les paramètres intrinsèques $(x_0, y_0, \alpha_x, \alpha_y)$ sont connus, les images de la mire sont à nouveau utilisées. Les positions des points idéaux (sans distorsion) sont connues puisque les dimensions du damier sont également connues. Les points déformés correspondants dans l'image sont détectés à l'aide d'un algorithme de détection des Features. Ensuite, le modèle de distorsion (équation 7.35) est résolu par une méthode des moindres carrés afin d'estimer les coefficients de distorsion κ_1 et κ_2.

Détection de marqueurs

Comme la caméra est en mouvement, cette étape et toutes celles qui suivent doivent être effectuées pour chaque image. Comme mentionné précédemment, nous utilisons des marqueurs artificiels ArUco [GJMSMCMC16]. Un seul marqueur est ajouté à la scène afin d'être utilisé comme référence 3D (voir figure 7.17). L'algorithme de détection est le suivant :

1. Détection des bords et seuillage.
2. Approximation polygonale du rectangle concave à quatre coins.
3. Projection en perspective pour obtenir un rectangle frontal.
4. Identification du code interne en le comparant aux marqueurs du dictionnaire.

Pour plus de détails, voir [GJMSMCMC16].

Estimation de la pose caméra

Une fois les coins du marqueur détectés, nous utilisons leurs positions pour estimer R et t [Zha00]. Tout d'abord, les quatre coins sont utilisés pour estimer l'homographie entre le marqueur et l'image correspondante (nous considérons le marqueur dans le plan $Z = 0$). Comme nous l'avons mentionné précédemment, un minimum de quatre points coplanaires est nécessaire pour estimer une homographie avec l'algorithme DLT [HZ03]. Enfin, connaissant K et H, le calcul des paramètres externes est effectué à l'aide de l'équation 7.41, ce qui donne :

$$\begin{aligned} r_1 &= K^{-1}h_1 \\ r_2 &= K^{-1}h_2 \\ t &= K^{-1}h_3 \end{aligned} \tag{7.42}$$

où h_1, h_2 et h_3 sont les colonnes de H.
Nous déduisons r_3 en utilisant un produit vectoriel :

$$r_3 = r_1 \times r_2 \tag{7.43}$$

Projection de l'objet 3D dans l'image 2D

Après avoir ajusté la taille et la position souhaitées de l'objet par le biais de transformations géométriques, nous projetons les vertex de l'objet 3D dans l'image comme suit :

1. Transformation des coordonnées de l'objet 3D dans le système de référence de la caméra en utilisant les paramètres extrinsèques R et t.
2. Projection des points 3D dans l'image en utilisant la matrice de calibration K.
3. Application du modèle de distorsion.

Suppression des surfaces cachées

Lors de la projection d'objets 3D, nous voulons, comme dans la vie réelle, voir uniquement la face visible des objets et non leur face cachée. Ce processus est appelé HSR (Hidden Surface Removal). Pour cela, nous utilisons l'algorithme HSR Z-buffering. Deux buffers sont utilisés, un pour la couleur et un pour la profondeur (Z-buffer). Nous commençons par calculer pour chaque polygone (face formée par trois sommets) la distance d_{poly} de son centre $\begin{bmatrix} X_{poly} & Y_{poly} & Z_{poly} \end{bmatrix}^T$ par rapport à la caméra en utilisant le vecteur de translation $t = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^T$:

$$d_{poly} = \sqrt{(t_x - X_{poly})^2 + (t_y - Y_{poly})^2 + (t_z - Z_{poly})^2} \tag{7.44}$$

Pour chaque pixel de l'objet 3D projeté, si la distance du polygone correspondant est inférieure à celle stockée dans le buffer de profondeur, la distance dans le buffer de profondeur ainsi que la couleur dans le buffer de couleur sont remplacées par celles du polygone correspondant. Dans ce qui suit, nous allons voir comment calculer la couleur du polygone.

Rendu par rastérisation

Afin d'éclairer l'objet 3D projeté, nous utilisons les modèles ambiant, diffus et spéculaire, présentés dans le chapitre 5. La réflexion ambiante simule un éclairage qui affecte également tous les objets, sa contribution est une valeur constante. La réflexion diffuse disperse la lumière dans toutes les directions, sa contribution est de :

$$I_{diffuse} = C_d max((N \bullet L), 0) \tag{7.45}$$

avec L la direction de la lumière, N le vecteur normal et C_d la couleur diffuse. La position de l'éclairage est sélectionnée par l'utilisateur pour correspondre au mieux à celle de la scène réelle. La couleur C_d dépend du type d'éclairage et du matériau de l'objet, mais nous ne couvrons pas cet aspect ici, nous utilisons simplement une valeur constante de la couleur. Enfin, la réflexion spéculaire simule la brillance d'un objet, sa contribution est :

$$I_{spculaire} = C_s max(0, (R \bullet V)^n) \tag{7.46}$$

avec C_s la couleur spéculaire, V la direction de vue, n est un facteur permettant de contrôler la largeur de la tache brillante, et R est la direction de réflexion. La figure 7.18 montre un exemple de projection utilisant un éclairage diffus et spéculaire.

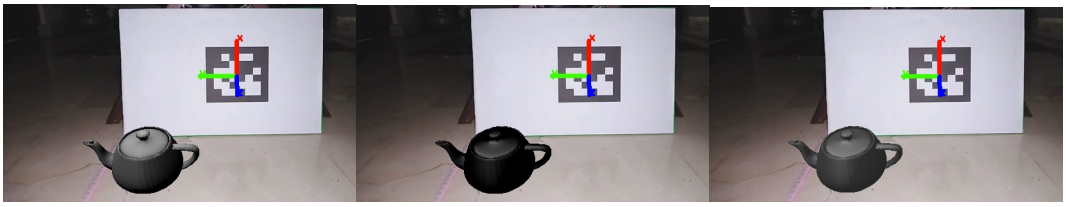

FIGURE 7.18 – De gauche à droite : projection et éclairage d'un objet 3D en utilisant : le modèle diffus, le modèle spéculaire et une combinaison des deux

.

Pour accélérer le processus dans le CPU, nous attribuons la même couleur à l'ensemble du polygone (flat shading). Pour une meilleure qualité, une valeur obtenue par interpolation à partir des trois sommets du polygone est attribuée à chaque pixel. De plus, le CPU ne permet pas un rendu rapide pour des milliers de sommets, et une implémentation GPU est nécessaire.
Notons que toutes les étapes, y compris la partie rendu, ont été implémentées sur le CPU en utilisant C++, OpenCV et la bibliothèque ArUco.

2.3 Évaluations quantitatives et qualitatives

Nous avons réalisé une expérience pour mesurer la stabilité de la méthode. Nous avons utilisé un téléphone mobile monté sur un trépied pour capturer des vidéos d'une scène fixe (sans aucun mouvement de caméra), afin d'avoir une vérité de terrain. Étant donné que la caméra et la scène sont statiques, un vertex de l'objet 3D devrait idéalement se trouver à la même position dans l'image. Nous calculons un score de répétabilité entre chaque image et la première image de la séquence considérée comme référence :

$$S = \frac{R}{N_f} \tag{7.47}$$

avec R le nombre de détections répétables, qui correspond au nombre de vertex projetés, dont la position n'a pas changé par rapport à l'image de référence. N_f est le nombre total de vertex.
Nous avons évalué trois types de séquences : éclairage fort, éclairage faible et éclairage variable, et pour chaque type de séquence nous avons placé la caméra à différentes distances. L'objectif est de mesurer l'impact de l'éclairage et de la taille des marqueurs sur la stabilité du matchmoving. La figure 7.19 montre les résultats de l'évaluation.

La première ligne de la figure 7.19 correspond aux séquences avec un fort éclairage. Nous pouvons voir que lorsque la taille du marqueur est assez grande, le score est d'environ $0,9$, le Matchmoving est assez stable. Lorsque la taille du marqueur diminue, le score tombe à $0,8$.
La deuxième ligne de la figure 7.19 correspond aux cas où l'éclairage est faible. Nous observons la même chose que précédemment, mais avec une légère baisse du score (environ $0,7$) lorsque la taille du marqueur est petite.
La troisième ligne de la figure 7.19 montre le cas d'un éclairage variable. Nous pouvons voir que lorsque nous faisons varier l'éclairage (après 500 images), en introduisant des ombres sur le marqueur et en faisant varier son contraste, le score chute.
Afin de mesurer de combien de pixels les vertex projetés se sont déplacés, nous avons recalculé le score des séquences où la taille du marqueur est petite (plus mauvais résultats), en introduisant une tolérance de déplacement de ± 1 pixel. La quatrième ligne de la figure 7.19 montre le résultat obtenu. Nous pouvons voir que le score reste toujours égal à 1, ce qui signifie que les erreurs que nous avons obtenues dans les expériences précédentes sont dues à des déplacements de vertex de ± 1 pixel, ce qui est généralement peu visible.
En plus de la taille des marqueurs et de l'éclairage, un autre facteur qui a un impact sur la stabilité du matchmoving est la distance de l'objet projeté par rapport au marqueur. En effet, l'erreur de projection est plus importante lorsque l'objet est éloigné du marqueur, en raison de l'erreur de calibration. Nous avons répété la dernière expérience avec une petite taille de marqueur pour un éclairage variable (les pires résultats) en choisissant une tolérance de ± 1, et en plaçant l'objet à différentes distances. La cinquième ligne de la figure 7.19 montre le résultat obtenu. Nous pouvons voir que lorsque l'éclairage varie, plus l'objet s'éloigne du marqueur, plus le score diminue. Dans la majorité des images, sauf lorsque l'objet est très éloigné du marqueur, le score reste assez élevé.
Une solution pour corriger le problème de stabilité consiste à traiter manuellement les quelques images à faible score en post-production.

D'après ces expériences, nous pouvons déduire que pour obtenir un résultat de projection stable, il est nécessaire d'utiliser un grand marqueur, un éclairage élevé et homogène, et de placer l'objet près du marqueur. Il est également important de réduire l'erreur de calibration en s'assurant que la mire du

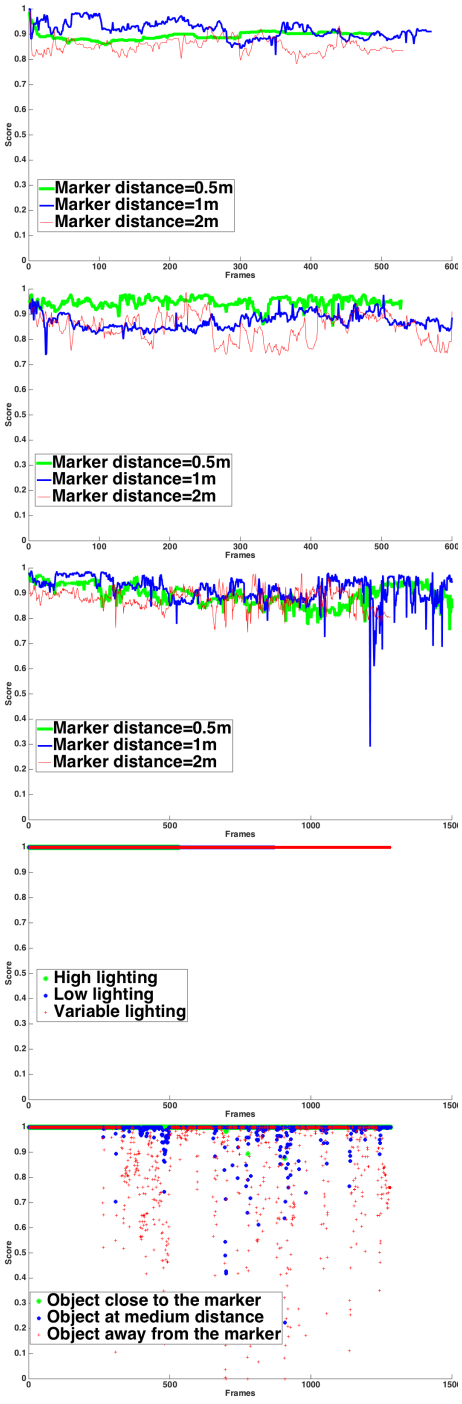

FIGURE 7.19 – Résultats de l'évaluation quantitative.

damier est parfaitement plane et en utilisant une caméra professionnelle.

Pour évaluer qualitativement le résultat, nous avons projeté un objet 3D dans des séquences vidéos capturées avec la caméra mobile d'un téléphone portable. La figure 7.20 montre des images acquises à partir de différents points de vue.

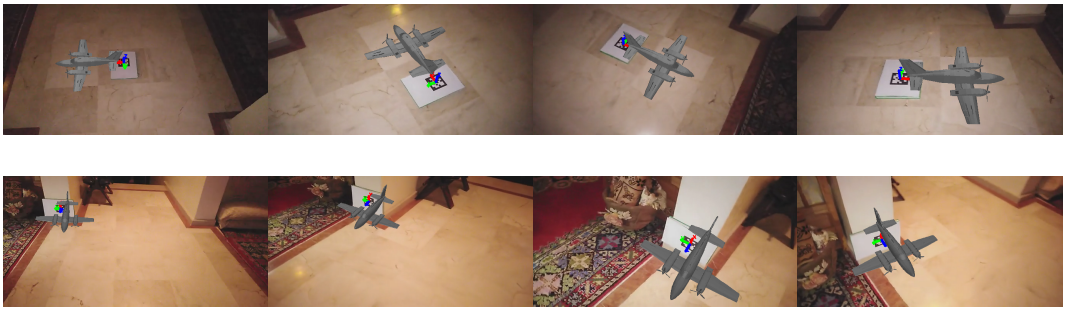

FIGURE 7.20 – Exemples de résultats pour une évaluation qualitative.

Nous pouvons voir que la géométrie de l'objet 3D correspond aux perspectives de la scène pendant le mouvement de la caméra. L'objet 3D est immobile pour faciliter l'évaluation du résultat : pour un résultat précis, l'objet doit rester immobile pendant le mouvement de la caméra. Nous avons constaté que la détection des marqueurs souffre d'instabilité et de non-détection lorsque la caméra se déplace rapidement. Parmi nos perspectives, nous proposons de résoudre ce problème avec un filtre de Kalman. Enfin, en Matchmoving, le marqueur ne doit pas être visible dans la vidéo finale. Dans le cas d'un objet immobile, il peut être placé au-dessus du marqueur. Dans le cas d'un objet mobile, une solution consiste à placer le marqueur dans une zone homogène, puis à le supprimer à l'aide d'un algorithme d'inpainting présenté dans le chapitre 6.

2.4 Conclusion

Nous avons présenté une méthode de prévisualisation de matchmoving en estimant la pose de la caméra, à l'aide d'un seul marqueur artificiel utilisé comme référence 3D. Les différents modules sont indépendants, cela permet de remplacer chacun d'entre eux par le plus approprié en fonction de l'application souhaitée. Dans notre cas, nous avons choisi les méthodes qui nous permettent de traiter les images de manière précise et rapide. Les évaluations quantitatives et qualitatives sur des vidéos de scènes réelles attestent de l'efficacité des algorithmes proposés, mais nous avons constaté que la méthode souffre d'instabilité sur certaines images lorsque le marqueur à une petite taille d'image, lorsque l'éclairage est faible ou inhomogène, lorsque les objets projetés sont loin du marqueur et lorsque la caméra se déplace rapidement. Tous les algorithmes ont été implémentés sur CPU, ce qui permet d'une part une migration plus facile vers n'importe quelle plateforme, mais pour obtenir une qualité de rendu plus réaliste, il est nécessaire d'utiliser une implémentation parallèle sur GPU.

3 CNN à fenêtre glissante et dataset de synthèse pour la détection rapide des points d'intérêt

Dans [HH21], nous proposons un nouveau modèle CNN à fenêtre glissante pour la détection rapide de Features, ainsi qu'une nouvelle dataset d'images de synthèse de Features utilisée pour entraîner le modèle, afin d'éviter l'utilisation de détecteurs traditionnels.

Dans les méthodes présentées dans le chapitre 4, les ensembles de données utilisés pour l'apprentissage des détecteurs, des descripteurs ou pour la mise en correspondance sont constitués de patchs de Features obtenus par les détecteurs traditionels discutés dans le chapitre 2. Cela est dû à la difficulté d'étiqueter manuellement les Features d'une image. Par exemple, une seule image peut contenir des centaines à des milliers de Features, qu'il faut localiser avec précision et dont il faut aussi déterminer les échelles et les orientations correspondantes. Cela rend la création manuelle d'un grand ensemble de données presque impossible. Cependant, les méthodes basées sur l'apprentissage fournissent de meilleurs descripteurs et une meilleure mise en correspondance, mais les détecteurs traditionnels restent largement utilisés et préférés aujourd'hui.
Nous partons donc du constat que les modèles de détection récents sont entrainés à partir des résultats des méthodes traditionnelles.
La première contribution est donc de créer un nouvel ensemble de données de Features de synthèse pour entraîner un modèle CNN de détection. Cet ensemble de données permet d'éviter l'utilisation de détecteurs traditionnels pour l'apprentissage et donc de ne pas entraîner le modèle avec certains Features peu fiables.
Une autre limite des détecteurs existants est l'utilisation d'une fenêtre glissante pour analyser l'image à la recherche de Features. Cela ralentit considérablement le traitement, en particulier pour les images à haute résolution.
La deuxième contribution est de proposer une nouvelle architecture CNN, basée sur une technique de fenêtre glissante convolutive [SEZ+13], pour éviter l'utilisation d'une fenêtre glissante et effectuer la détection avec un gain significatif en temps de traitement.
Le modèle proposé est évalué sur la dataset de synthèse et comparé à différentes architectures en termes de performances et de temps de traitement.
Enfin, nous proposons également une méthode pour extraire un descripteur de Features directement à partir du modèle de détection. L'invariance du descripteur par rotation, translation et échelle est testée par mise en correspondance d'images.

L'idée initiale est inspirée par le détecteur traditionnel de Harris [HS+88]. Ce

détecteur peut être vu comme une classification de l'image en trois régions : coins, uniformes et lignes. Nous avons donc créé un ensemble de données de patchs images de synthèse de ces trois structures, que nous avons augmenté par diverses transformations géométriques et photométriques et par l'ajout de bruit et de flou. L'ensemble de données est utilisé pour entraîner un CNN qui prend une image en entrée, et permet d'obtenir à sa sortie une carte de Features utilisée pour la détection, ainsi que pour extraire un descripteur.

3.1 Datasets

Ensemble de données de synthèse

Nous avons créé un ensemble de données de synthèse contenant deux classes de patchs de Features (coins) et de non Features (lignes et régions uniformes), en niveaux de gris et en dimensions 14×14. La figure 7.21 montre quelques échantillons aléatoires de l'ensemble de données des patchs, simulant différentes variations d'angle, de rotation, de luminance, de contraste, de bruit et de flou. Pour rendre la détection invariante par des transformations géométriques et photométriques, nous avons augmenté les patchs en faisant varier différents paramètres.

Pour les Features des coins, nous simulons, à pas constants : 13 angles des coins variant de 90° à 130° (pour éviter de les confondre avec des contours ou des lignes), 120 rotations de 0 à 360°, 18 contrastes supérieurs à 0.6 avec des coins à la fois plus sombres et plus clairs que le fond, un bruit gaussien $\mathcal{N}(0, (0.001)^2)$ (avec et sans), et un flou gaussien avec un noyau de taille 2×2 et un écart-type $\sigma = 2/3$ (avec et sans). Nous obtenons un nombre total de Features de 112320 ($13 \times 120 \times 18 \times 2 \times 2$).

Pour les non Features des lignes, nous avons simulé 90 rotations entre 0 et 180° (lignes symétriques), 20 luminances et contrastes, 6 bruits gaussiens avec σ entre 0 et 0.002, et 3 flous gaussiens de taille de noyau entre 1 et 3 avec $\sigma = 1$.

Pour les non Features des demi-lignes, nous avons simulé 180 rotations entre 0 et 360°, 14 luminances et contrastes, 4 bruits gaussiens avec un écart type entre 0 et $0,002$, et 3 flous gaussiens avec une taille de noyau entre 1 et 3 avec un $\sigma = 1$.

Pour les non Features des régions uniformes, nous avons simulé 205 niveaux de gris entre 0 et 1, 34 bruits gaussiens avec des sigmas entre 0 et $0,01$, et 9 flous gaussiens de taille de noyau entre 1 et 9 avec $\sigma = 3$.

Nous obtenons un total de 125370 non Features.

Les différentes valeurs ont été choisies, empiriquement, après plusieurs essais, à la fois pour avoir un maximum de variabilités, des structures coins et lignes visuellement perceptibles, et pour obtenir des nombres équilibrés d'échantillons entre les classes de Features et de non Features.

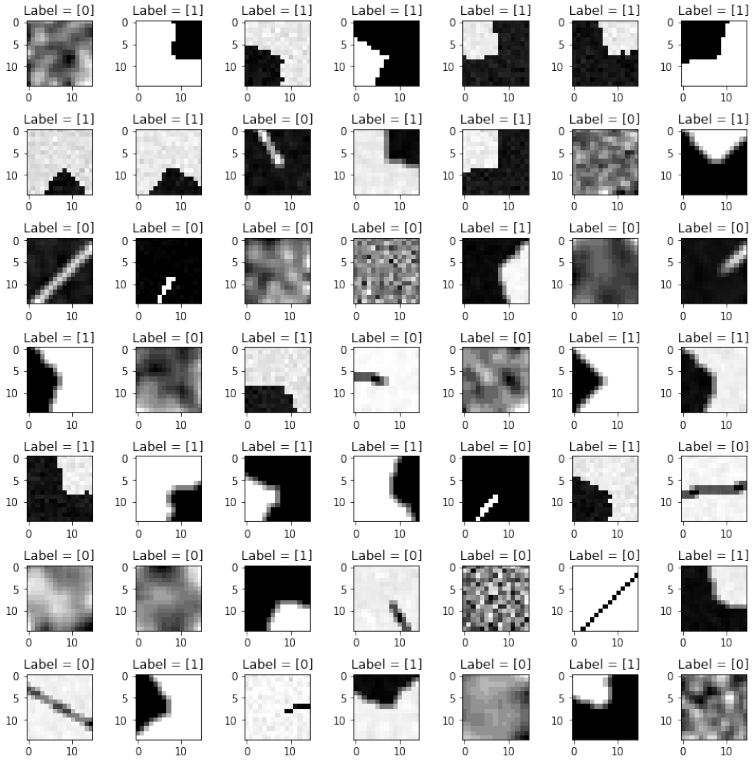

FIGURE 7.21 – Échantillons aléatoires de l'ensemble de données de patchs avec différentes transformations géométriques et photométriques. Les étiquettes 1 et 0 correspondent respectivement aux Features (coins) et aux non Features (lignes et régions uniformes)..

Les données ont été mélangées de manière aléatoire avant d'être stockées au format HDF5 dans deux fichiers d'apprentissage et de test.
Les images sont en niveau de gris et codées sur 8 bits (valeurs comprises entre 0 et 255), il est donc nécessaire de les normaliser et de les convertir en flottant avant l'apprentissage.
Au total, nous avons généré environ 237000 patchs avec des classes équilibrées ($112K$ Features et $125K$ non Features). La dataset a été rendue publique et mise à la disposition de la communauté [Datb].

Ensemble de données des images mosaïques

À partir de ces patchs et des étiquettes correspondantes (1 pour les Features et 0 sinon), nous avons généré un nouvel ensemble de données d'images mosaïques de taille 560×420, que nous utilisons en entrée du modèle, ainsi que les images de la carte des Features Y de taille 274×204, et que nous utilisons en sortie pour la localisation des Features. En effet, le modèle utilisé permet de détecter des patchs de taille 14×14 en balayant l'image avec un pas de 2 pixels, et donc le nombre de blocs balayés horizontalement dans l'image est : $274 = (560 - 14)/2 + 1$. En général, nous avons une taille de sortie de

$[(l-w)/p]+1$, avec l le nombre de lignes (ou colonnes), w la taille du patch et p le pas. Chaque image contient donc 1200 patchs, ce qui donne un ensemble de données en mosaïque d'environ 200 images (237000/1200).
Nous attribuons à chaque Feature des images Y une valeur de 1, et son voisinage reçoit des valeurs inférieures à 1, mais pas 0, selon une forme gaussienne. En effet, les pixels dans le voisinage d'un Feature contiennent une partie de celui-ci et doivent être distingués des non Features qui ont une valeur de 0. La figure 7.22 montre un exemple d'images.

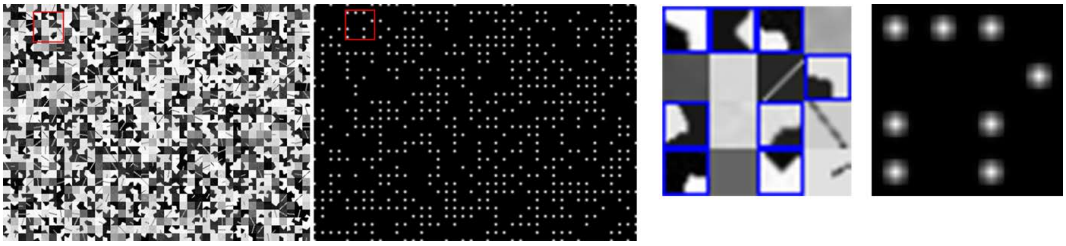

FIGURE 7.22 – Image en mosaïque et carte de Features correspondante.

En concaténant les patchs dans une image en mosaïque, nous générons de nouveaux coins avec un angle de 90° au point d'intersection de chaque 4 patchs voisins. Il est important de considérer ces coins comme des Features lorsqu'un des 4 patchs présente un contraste élevé avec ses deux voisins horizontaux et verticaux. La figure 7.23 illustre cette technique. Nous considérons comme Featues l'intersection de 4 patchs, si l'un des coins Pi, avec $(i \in [1,4])$, vérifie la condition sur le contraste suivante :

$$(\frac{min(Mi,Mj)}{max(Mi,Mj)}. \geq th) \wedge (\frac{min(Mi,Mk)}{max(Mi,Mk)} \geq th), \tag{7.48}$$

tels que j et k sont les indices des deux voisins horizontaux et verticaux du coin i, $th = 0,6$ comme pour le seuil de contraste des patchs générés, et $Mi = median(Pi)$.

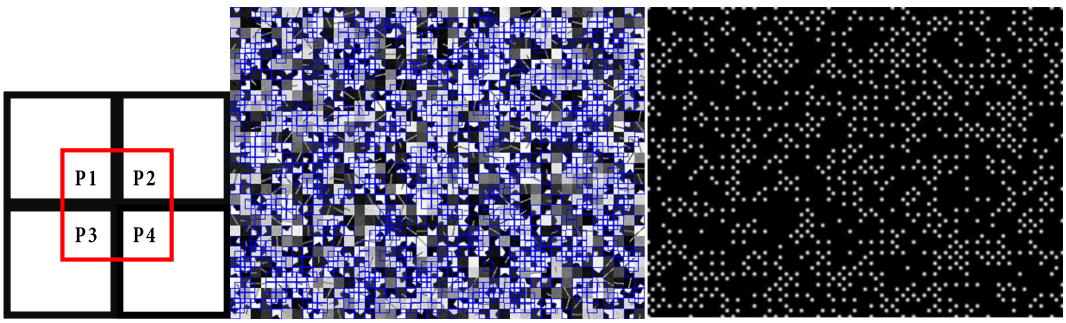

FIGURE 7.23 – De gauche à droite : Coin potentiel à l'intersection de quatre patchs, image en mosaïque et carte de Features après l'ajout de ces coins

.

3.2 Architecture

Comme il s'agit d'un nouvel ensemble de données, nous avons évalué l'apprentissage de différentes architectures de classification, peu profondes et profondes, de type ANN et CNN, que nous avons améliorées avec une nouvelle architecture convolutive à fenêtre glissante.

Approche de classification

Dans la première approche, nous avons utilisé l'ensemble de données des patchs. Il s'agit d'une approche de classification. Nous avons testé différentes architectures à profondeur croissante, tant sur les dimensions des couches que sur le nombre de filtres : trois modèles ANN à 2 couches (régression logistique), 4 couches et 12 couches, ainsi que deux modèles CNN à 6 et 12 couches. Le nombre de paramètres entraînables des architectures varie entre $29K$ et $32M$. Comme il s'agit d'une approche de classification, nous avons utilisé des fonctions d'activation ReLU, une sortie Sigmoïde (fonction logistique), et une fonction de perte BCE (entropie croisée binaire). Cette approche nécessite un temps de traitement important du fait de la nécessité de scanner l'image par bloc et de faire appel au modèle pour chaque bloc. Nous avons donc développé un nouveau modèle qui prédit la position de toutes les caractéristiques directement à sa sortie.

Approche de localisation

Nous utilisons une méthode de fenêtre glissante convolutive, consistant en un modèle CNN qui exploite le fait que les blocs traversés par une fenêtre glissante ont des régions en commun, et donc qu'une partie de l'information calculée pour un bloc peut être utilisée par les blocs voisins, comme le montre la figure 7.24.

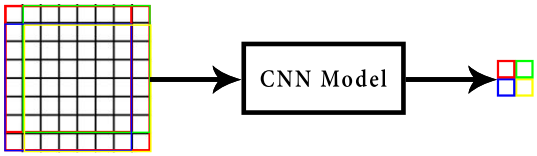

FIGURE 7.24 – Image 8×8 parcourue par blocs 7×7 avec un pas de 1. Le CNN renvoie une image 2×2 contenant le résultat de chaque bloc.

Nous utilisons donc un modèle qui prend en entrée des images de la même taille 560×420 que celles de l'ensemble de données mosaïque, et renvoie en sortie une image de la taille des cartes correspondantes de Features 274×204, ce qui revient à détecter des Features de taille 14×14 avec un pas de 2.
L'architecture du modèle suit un schéma quelque peu similaire à celui du VGG16 [SZ14] en ce qui concerne la taille des filtres et la séquence des

opérations, mais en adaptant l'architecture à notre problème de localisation (format de sortie, fonction d'activation et fonction de perte). Nous utilisons également moins de couches et avons donc moins de paramètres (seulement 2 M par rapport aux 138 M du VGG16).
La figure 7.25 montre l'architecture utilisée (image du haut). Chaque bloc correspond à une séquence de couches différentes : 'Conv' correspond à la couche de convolution, 'BN' (Batch normalization) à la couche de normalisation par lots (mise à l'échelle de chaque couche), 'ReLu' à la fonction d'activation ReLU et 'Linear' à une fonction d'activation linéaire. Chaque bloc est répété 2 ou 3 fois comme indiqué par le premier chiffre du bloc. La variable f correspond à la taille du filtre convolutif et n au nombre de filtres. Le bloc max pooling et le padding valide permettent de réduire la taille de l'image pour la conformer à la taille de la sortie souhaitée.

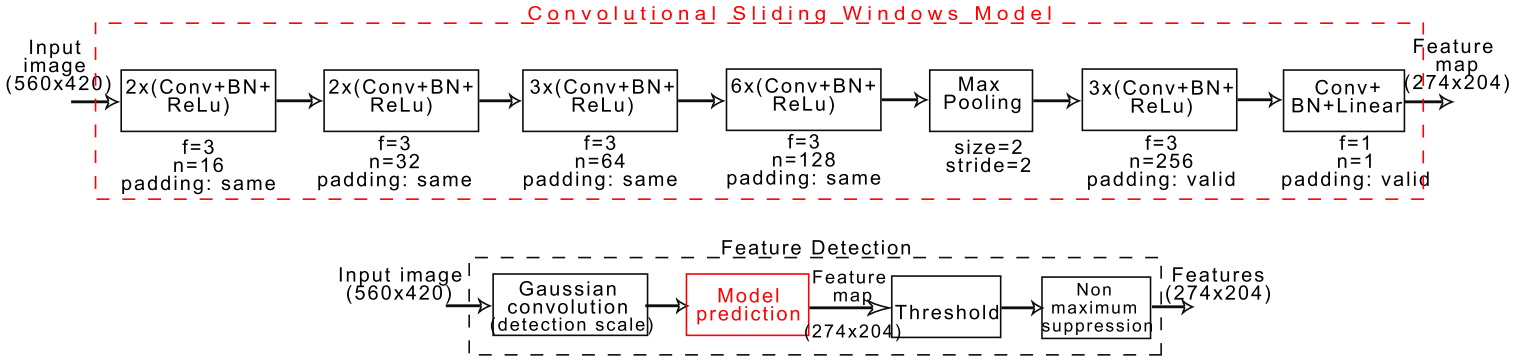

FIGURE 7.25 – Modèle CNN utilisé pour l'apprentissage des cartes de Features (en haut) et algorithme de détection des Features (en bas).

Étant donné le format et le type d'image de sortie du modèle, il ne s'agit plus d'un problème de classification, mais plutôt d'un problème de localisation de Features par régression. Nous utilisons donc une fonction d'activation linéaire en sortie et une fonction de perte MSE (Mean Squared Error).
Nous avons également créé une version régularisée du modèle par Dropout pour éviter l'overfitting. Le Dropout correspond à la suppression aléatoire des unités d'activation pour chaque échantillon pendant l'apprentissage. Après chaque fonction d'activation (sauf pour la sortie linéaire), nous utilisons un dropout de $0,7$ (30% de chance d'éliminer l'activation). Notez que pour la prédiction sur l'ensemble de données de test, nous n'utilisons pas le dropout.

Apprentissage

Le modèle est entraîné sur GPU avec l'ensemble des données mosaïques par l'algorithme Adam optimizer. En effet, nous avons testé 3 algorithmes d'optimisation : Adam, Gradient Descent Momentum et RMS, qui ont la particularité de bien se généraliser à tous les types de problèmes. Les résultats

obtenus sont équivalents. Nous avons donc opté pour Adam pour sa rapidité. Les paramètres d'optimisation sont les suivants. Pour les paramètres Adam : $\beta_1 = 0.9$, $\beta_2 = 0.999$, et $\epsilon = 1e-07$. Nous utilisons un nombre d'itérations de 50, et un taux d'apprentissage décroissant entre $0,1$ et $0,00001$ en fonction du nombre d'itérations. Les poids du modèle sont initialisés par l'initialisateur uniforme de Xavier. Nous avons utilisé 80% des données pour l'apprentissage, 10% pour la validation et 10% pour le test. La figure 7.26 montre la variation de l'erreur MSE pendant l'optimisation des modèles avec et sans régularisation.

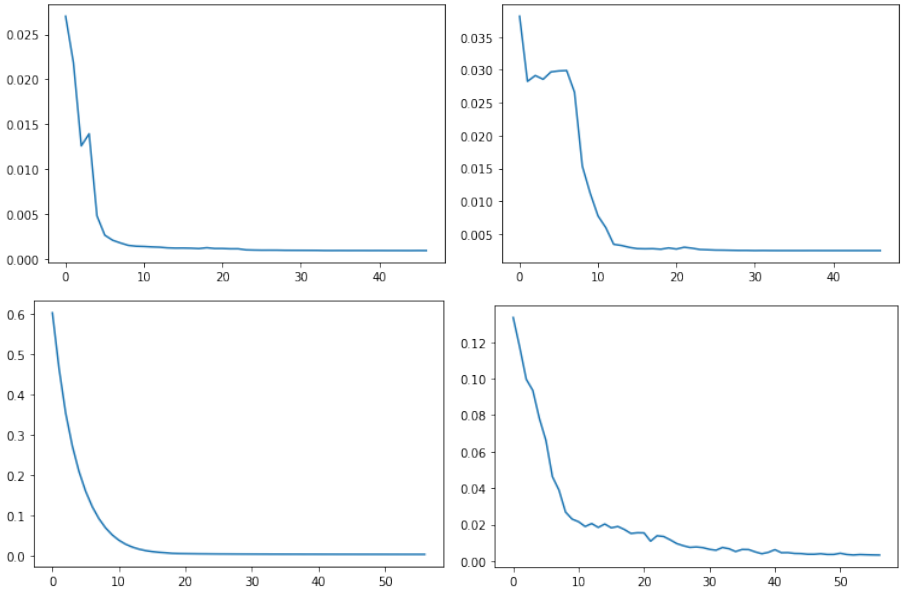

FIGURE 7.26 – MSE en fonction du nombre d'itérations, respectivement, lors de l'optimisation par les ensembles de données d'apprentissage et de validation, du modèle sans régularisation (deux premières images) et avec régularisation (deux dernières images).

Détection des Features

Une fois le modèle entraîné, il est utilisé par un algorithme de détection, comme le montre la figure 7.25 (image du bas). Un pré-traitement appliqué à l'image en entrée, consiste en une convolution gaussienne pour éliminer les Features plus petits que l'échelle de détection souhaitée. En effet, cela revient à réduire la résolution intrinsèque de l'image (sans modifier les dimensions de l'image), tout en préservant les structures de l'image, contrairement au sous-échantillonnage. Cela permet une détection multi-échelle des caractéristiques, non limitée à la taille 14×14 des patchs de l'ensemble des données. L'ensemble d'espace d'échelles utilisé est $W = [14, 18, 22, 28, 34, 42]$. Le noyau de convolution est le suivant : $W_i/2 + 1$ et $\sigma_i = W_i/6$.
Après avoir appliqué le modèle convolutionnel à fenêtre glissante, la détection est effectuée par seuillage suivi d'une opération de suppression des non-maxima

locaux. Le seuillage permet d'éliminer les non Features ainsi que le bruit, nous utilisons un seuil égal à $0,3$ (après normalisation de la carte des Features entre 0 et 1). La suppression des non maxima locaux permet de détecter le centre des Features. En effet, autour d'un même Feature, il y en a plusieurs autres qui correspondent à la même structure, nous ne retenons donc que celle qui est au centre en détectant le maximum local sur une fenêtre de 11×11, mais selon l'application, il peut être préférable d'utiliser une fenêtre plus petite pour plus de détections. La figure 7.28 montre les résultats de détection à différentes échelles appliquées au matching.

Description des Features

Un autre intérêt du modèle proposé est qu'il permet d'extraire un descripteur qui peut être utilisé pour la mise en correspondance. Pour cela, nous utilisons la sortie de la première couche comme descripteur de l'image, en appliquant les poids obtenus lors de l'apprentissage. La figure 7.27 montre les étapes de l'extraction du descripteur de caractéristiques.

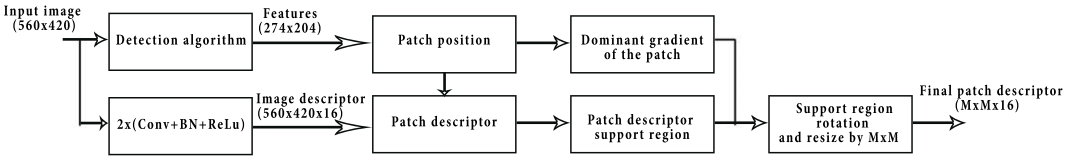

FIGURE 7.27 – Feature Descriptor Extraction Algorithm.

Nous utilisons la première couche pour extraire le descripteur, car elle permet de capturer des textures de bas niveau plus intéressantes. En parallèle, nous appliquons à l'image l'algorithme de détection. La position des Features est utilisée à la fois pour extraire le descripteur, et pour détecter l'orientation du gradient dominant (calculée sur un histogramme de 32 bins). Nous extrayons une région de support autour du patch descripteur de taille 3 fois plus grande que le patch, qui définira les pixels qui contribueront au calcul du descripteur. Nous appliquons à cette région de support une rotation afin que le gradient dominant soit horizontal. Cela permet de comparer des Features qui ont été détectées avec des orientations différentes. Enfin, le patch support est redimensionné afin de pouvoir comparer les Features détectées à différentes échelles. Le résultat final est un descripteur de taille $M \times M \times 16$, avec $M = 14$ dans le cas de notre étude.

3.3 Résultats

Évaluation de la détection des Features

Nous utilisons les ensembles de données générés (patch et mosaïque) pour évaluer les performances des différentes approches présentées dans la section

précédente. Pour la première approche de classification, nous utilisons une métrique Accuracy. Le tableau 7.1 montre les résultats obtenus pour les 3 architectures. Nous pouvons voir que tous les modèles, du peu profond au plus profond, donnent une précision de 100% sur les 3 sous-ensembles de données d'apprentissage, de validation et de test. Une simple régression logistique permet de classifier parfaitement les exemples des ensembles de données. On remarque également que plus le modèle est profond, plus la fonction de coût diminue. Cependant, le temps de traitement est d'environ 160 secondes par image sur CPU (utilisé à cause de la lenteur des boucles sur GPU).

TABLE 7.1 – Résultats de l'évaluation des approches de classification sur l'ensemble de données des patchs.

Architecture	Loss (BCE)			Accuracy		
	Train	Validation	Test	Train	Validation	Test
ANN 2 layers	4.5735e-05	4.2213e-05	4.2313e-05	100%	100%	100%
ANN 4 layers	2.6905e-08	2.5658e-08	2.3847e-08	100%	100%	100%
ANN 12 layers	1.2389e-09	1.1162e-08	4.7500e-08	100%	100%	100%
CNN 6 layers	3.0576e-05	2.0190e-05	2.0169e-05	100%	100%	100%
CNN 12 layers	2.4927e-06	1.8133e-06	1.8440e-06	100%	100%	100%

La deuxième approche par localisation a été évaluée en utilisant une mesure MSE puisqu'il ne s'agit plus d'un problème de classification, mais d'un problème de régression. Le tableau 7.2 montre les résultats obtenus. Nous constatons sur la première ligne que le modèle sans régularisation à un léger problème de sur-apprentissage dû à la variance entre les ensembles de test et d'apprentissage. Ce problème est résolu avec le modèle régularisé comme le montre la deuxième ligne du tableau. Le temps de traitement est d'environ $200ms$ par image sur GPU, nous avons donc un gain de 800 fois par rapport à la première approche pour des performances de détection similaires (très faible perte).

TABLE 7.2 – Résultats de l'évaluation de l'approche de localisation sur l'ensemble de données des images mosaïques.

Architecture	Loss (MSE)		
	Train	Validation	Test
Our model	9.5854e-04	0.0025	0.0151
Our model+Regularization	0.0029	0.0035	0.0025

Descripteur et mise en correspondance

Nous utilisons l'algorithme de détection combiné à l'algorithme d'extraction de descripteur pour effectuer la mise en correspondance entre deux images et tester l'invariance du descripteur. Nous appliquons à une image différentes transformations géométriques de translation, rotation et échelle, et comparons les descripteurs des Features détectées dans l'image avant et après transformations. Le descripteur de chaque Feature est comparé à ceux de son voisinage dans l'autre image, et la taille du voisinage utilisé est de 100 pixels. Nous utilisons une distance MSE pour comparer les descripteurs, le seuil de détection utilisé est de $0,1$. Pour plusieurs Features de la première image, nous pouvons trouver la même correspondance dans la seconde image vérifiant la condition de correspondance, il est donc important de ne garder que celle qui a l'erreur la plus faible.
La figure 7.28 montre des exemples de résultats obtenus sur une image. Pour une rotation de 5°, nous avons 95 correspondances trouvées sur environ 900 Features détectées, et 100% des correspondances sont correctes. Par contre, plus nous augmentons l'angle de rotation, moins nous avons de correspondances. La méthode n'est donc pas assez robuste pour les applications de correspondance à large variation géométriques, mais le résultat est suffisant pour les applications de suivi dans des séquences vidéo.
Nous augmentons ensuite l'échelle de l'image par un coefficient d'environ 1,5. Comme nous connaissons le coefficient d'échelle, nous avons appliqué à chaque image une détection à l'échelle correspondante. En général, nous devons appliquer une détection multi-échelle pour chaque image et effectuer la correspondance entre chaque échelle dans une image et toutes les autres échelles dans la seconde image, et ne retenir que le Feature candidat qui minimise la distance de correspondance. Nous obtenons ainsi 52 correspondances et une seule correspondance incorrecte, et donc une précision de 98%.
Enfin, nous avons appliqué à la fois une translation de 10 pixels, une rotation de 10°, et une échelle de 1,5. Nous obtenons 14 correspondances avec 100% de correspondance correcte. Notez que le nombre de correspondances diminue, car en raison de la taille de l'échelle, environ 3/4 de la première image n'est plus visible dans la seconde, et aussi en raison de l'angle de rotation plus important que dans l'exemple précédent.

3.4 Conclusion

Nous avons présenté une nouvelle méthode de détection des Features et d'extraction des descripteurs à l'aide d'un modèle CNN convolutif à fenêtre glissante. Le modèle est entraîné avec un nouvel ensemble de données d'images de synthèse. La technique de la fenêtre glissante convolutionnelle accélère le traitement local nécessaire à la détection des Features. Un algorithme d'extraction de descripteurs invariants est également proposé. Des couches de bas niveau sont utilisées pour extraire les descripteurs des Features détectés. Ainsi, le même modèle peut être utilisé pour la détection et la description des Features. Les résultats de l'évaluation du détecteur sur un ensemble de

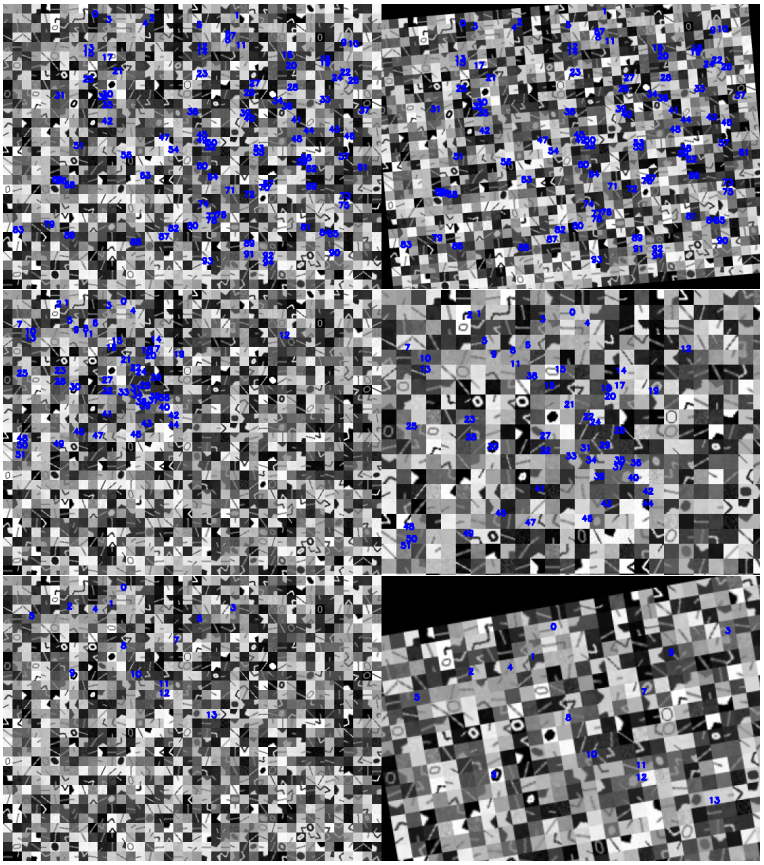

FIGURE 7.28 – Mise en correspondance par comparaison des descripteurs des Features détectées. Nous appliquons différentes transformations à l'image : rotation (première ligne), échelle (deuxième ligne) et une combinaison de translation, rotation et échelle (troisième ligne).

données de synthèse, que nous avons créé, montrent que le modèle proposé permet d'obtenir des performances de détection similaires à celles d'autres architectures de classification avec une erreur très faible, tout en permettant une amélioration très significative du temps de traitement. Le test d'invariance des descripteurs par mise en correspondance atteste de leur efficacité pour des applications telles que le suivi vidéo.

4 CNN Siamois et dataset de paires de Feature de synthèse pour la mise en correspondance

Nous avons utilisé les mêmes patchs de synthèse présentés dans la section précédente, afin de créer une nouvelle dataset large de paires de Features, similaires et différents, selon divers critères de transformations géométriques et photométriques, afin d'effectuer, grâce à un modèle convolutif siamois, la description et la mise en correspondance des points d'intérêt détectés par le

modèle convolutif à fenêtre glissante précédent. Nous complétons ainsi toute la pipeline de mise en correspondance par des modèles CNN. Les datasets proposées permettent donc une nouvelle approche pour entraîner les différents modules de mise en correspondance sans utiliser les méthodes traditionnelles. A notre connaissance, il s'agit des premiers datasets de Features de synthèse pour la mise en correspondance.

Les différentes datasets ont été rendues publiques [Datb]. Les ensembles de données sont aux formats HDF5 et SAV. L'ensemble de données de patchs est divisé en deux fichiers, l'ensemble d'apprentissage et l'ensemble de test, au format HDF5. L'ensemble de données mosaïque est divisé en quatre fichiers au format SAV : images mosaiques des ensembles d'apprentissage et de test, et cartes de Features des ensembles d'apprentissage et de test. L'ensemble de données des paires de Features est stocké dans un seul fichier au format HDF5. Chaque ensemble de données est accompagné d'un code notebook python IPYNB permettant de lire et d'afficher les données ainsi que d'autres informations (format, type, nombre d'échantillons), afin de faciliter son utilisation.

4.1 Variants du modèle convolutif à fenêtre glissante et de la dataset mosaïque

Le modèle convolutif à fenêtre glissante représenté sur la figure 7.29 prend en entrée une image de dimensions 600×450 pixels et renvoie une carte de Features qui permet de localiser les Features.

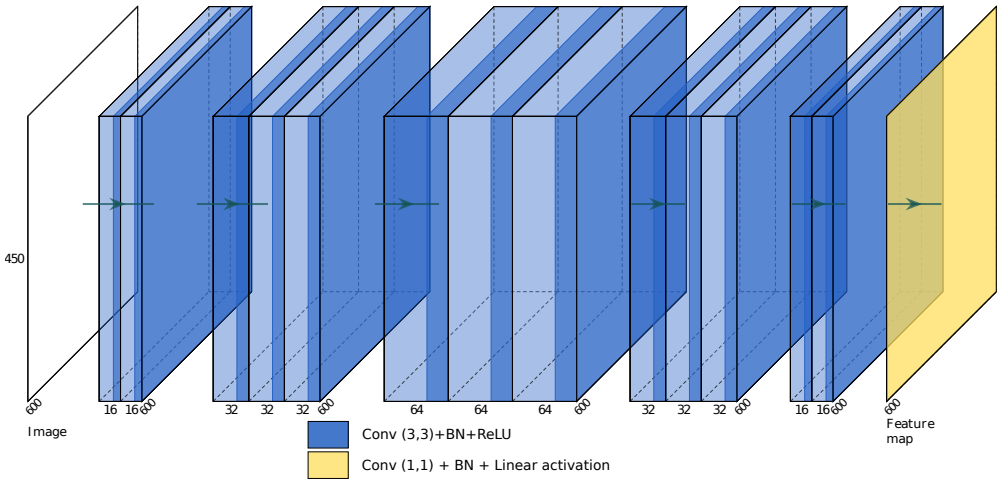

FIGURE 7.29 – Nouvelle version du modèle de fenêtre glissante convolutionnelle pour la détection rapide de Feature.

Notons que le modèle présenté ici permet d'obtenir une carte de Feature plus dense que dans sa première version présentée dans la section 3.2. En effet,

nous constatons sur la figure 7.29 que les dimensions de l'image de sortie sont les mêmes que celles de l'image en entrée, et non inférieures comme dans la première version. Cela revient à parcourir l'image avec un pas de 1 avec la fenetre glissante, d'où une carte de Features plus dense.
Par conséquent, l'ensemble de données d'images mosaïques est également légèrement différent du précédent.
Pour entraîner ce modèle, nous avons généré, à partir de l'ensemble de données de patchs, des images en mosaïque ainsi que les cartes de Features correspondantes. Les images en mosaïque sont obtenues en concaténant de manière aléatoire les patchs Features et non Features dans des images 600×450. Chaque image contient donc 1200 patches (40×30 patches de dimensions 15×15). Nous obtenons un total de 198 images à partir des 27690 patchs. Les patchs restants de la dernière image sont remplis de zéros. Les cartes de Features sont obtenues en attribuant aux Features des formes gaussiennes de taille 15×15 et $\sigma = 15/9$, normalisées entre 0 et 1. Le centre du Feature est donc étiqueté avec une valeur de 1 et son voisinage avec des valeurs inférieures à 1 mais non nulles (car elles contiennent également une partie du Feature), ce qui nous permet de les distinguer des patchs non Features auxquels nous attribuons une valeur de 0.
Nous avons aussi amélioré la condition de sélection des Features générés à l'intersection.
En concaténant les patchs, nous générons à leur intersection de nouveaux Features coins, lorsque l'un d'entre eux présente un contraste important par rapport à tous ses voisins. Nous considérons que le patch de dimension 15×15 situé à l'intersection de 4 patchs est un Feature, si l'un des coins P_i de l'intersection (figure 7.23), satisfait la condition suivante :

$$(Ctr(M_i, M_j) \geq Th) \wedge (Ctr(M_i, M_k) \geq Th) \wedge (Ctr(M_i, M_l) \geq Th), \quad (7.49)$$

avec $Ctr(A, B)$ le contraste entre deux valeurs A et B :

$$Ctr(A, B) = \frac{max(A, B) - min(A, B)}{max(A, B) + min(A, B)} \quad (7.50)$$

M_i est la valeur médiane du coin P_i de l'intersection, avec $i \in \{1, 2, 3, 4\}$.
j, k et l sont les indices des voisins P_j, P_k et P_l de P_i, avec $(i, j, k, l) \in \{1, 2, 3, 4\}$.
$Th = 0,6$ est le seuil de contraste utilisé.
Les mosaïques et les cartes de Features sont stockées au format SAV, de type float avec des valeurs comprises entre 0 et 1, et séparées en 179 images (90%) pour l'ensemble d'apprentissage et 20 images (10%) pour l'ensemble de test.
Nous avons effectué l'apprentissage du modèle avec les paramètres suivants : ensemble d'apprentissage de 80%, ensemble de validation de 10%, ensemble de test de 10%, nombre d'itérations de 1200, optimiseur Adam, taux d'apprentissage décroissant entre $0,01$ et $0,00001$, fonction de coût MSE, métrique de similarité en cosinus.

L'apprentissage a pris environ 7 heures sur GPU (Tesla P100-PCIE-16GB). La figure 7.30 montre les courbes d'apprentissage et le tableau 7.3 montre les valeurs MSE et de similarité en cosinus obtenues à la fin de l'apprentissage.

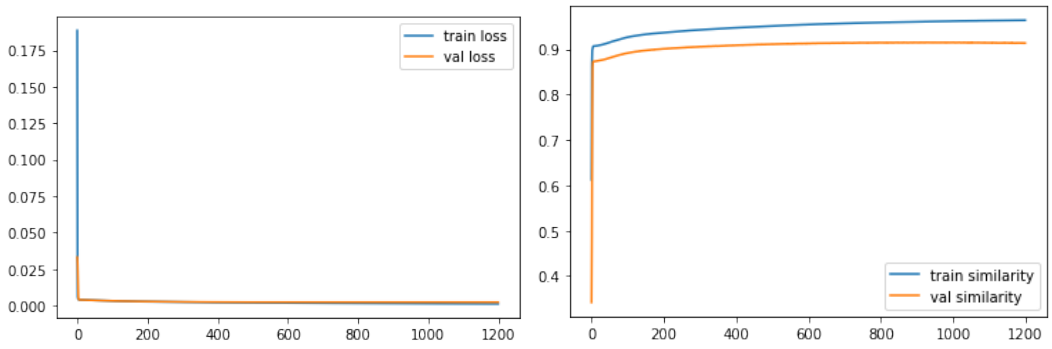

FIGURE 7.30 – Apprentissage du modèle convolutionnel à fenêtre glissante sur l'ensemble de données mosaïque : Fonction de coût MSE et métrique de similarité cosinus, en fonction du nombre d'itérations, pour l'ensemble d'apprentissage et l'ensemble de validation

.

TABLE 7.3 – Valeurs de MSE et de la similarité cosinus après l'apprentissage du modèle de détection sur l'ensemble de données mosaïques.

Fonction de coût (MSE)			Similarité cosinus		
Apprentissage	Validation	Test	Apprentissage	Validation	Test
0.0011	0.0022	0.0023	0.964	0.9137	0.9395

4.2 Ensemble de données de paires de Features pour la mise en correspondance

Nous présentons ici l'ensemble de données de paires de Features utilisé pour l'apprentissage d'un modèle convolutif siamois de description et de mise en correspondance de Features. Pour construire cet ensemble de données, nous n'utilisons que les patchs de coins puisque le module de mise en correspondance doit trouver la correspondance entre les Features détectées (coins). Notez que nous n'utilisons pas tous les patchs de coin à cause des limites imposées par la mémoire RAM pendant le processus d'apprentissage, comme nous l'expliquerons plus en détail plus loin. Nous ne retenons que 15600 coins qui nous permettent de générer l'ensemble de données de paires de Features. Pour cela, nous considérons 13 angles de coins comme dans l'ensemble de données des patchs, mais seulement 10 contrastes, 30 rotations, 2 valeurs de bruit et 2 valeurs de flou ($15600 = 13 * 10 * 30 * 2 * 2$).
Nous considérons que deux coins sont similaires s'ils ont le même angle et

le même contraste, mais peuvent avoir des rotations, un bruit ou un flou différents. Nous construisons ainsi 130 sous-ensembles de Features similaires, chacun contenant 120 Feature. En effet, le nombre 120 correspond à $30 \times 2 \times 2$ (30 rotations, 2 bruits et 2 flous). Le nombre 130 correspond à 13×10 (13 angles et 10 contrastes).
Nous notons x_j^i le j-ème patch du sous-ensemble i de patchs similaires, avec $i \in [1; 130]$ et $j \in [1; 120]$.
Pour générer les paires de Features, nous organisons les sous-ensembles X^i de patchs similaires, avec $i \in [1; 130]$, comme suit :

$$\{X^1, X^2, ..., X^{130}\} = \{\{x_1^1, ..., x_{120}^1\}, \{x_1^2, ..., x_{120}^2\}, ..., \{x_1^{130}, ..., x_{120}^{130}\}\} \quad (7.51)$$

À partir de ces données, nous générons un ensemble de paires de Features positives et négatives. Nous notons P l'ensemble des paires positives (patchs similaires) et N l'ensemble des paires négatives (patchs différents).
L'ensemble P est constitué par toutes les combinaisons de paires dans chaque sous-ensemble X^i, y compris les paires constituées par le même Feature :

$$P = \{(x_m^k, x_n^k) | k \in [1; 130], (m, n) \in [1; 120]\} \quad (7.52)$$

Ainsi, nous avons un nombre d'échantillons positifs de :

$$Card(P) = (130 \times C_{120}^2) + (130 \times 120) = 943800 \quad (7.53)$$

Nous utilisons l'algorithme 1 pour la génération des échantillons de la classe P.

Algorithme 1 : Algorithme de génération des échantillons de P.

```
For k = 1 ; k ≤ 130 ; k++
    For m = 1 ; m ≤ 120 ; m++
        For n = m ; n ≤ 120 ; n = ++
            Échantillon positif (x_m^k, x_n^k)
```

L'ensemble N de paires négatives correspond à toutes les combinaisons de chaque Feature d'un sous-ensemble X^i avec tous les Features des autres sous-ensembles X^j tel que $i \neq j$:

$$N = \{(x_p^m, x_q^n) | m \neq n, (m, n) \in [1; 130], (p, q) \in [1; 120]\} \quad (7.54)$$

Nous avons donc un nombre d'échantillons négatifs de :

$$Card(N) = C_{130 \times 120}^2 - (130 \times C_{12algpseudocode0}^2) = 120744000 \quad (7.55)$$

Le très grand nombre de paires négatives soulève deux problèmes. D'une part, les deux classes P et N sont largement déséquilibrées. D'autre part, la quantité de mémoire disque et surtout de mémoire vive nécessaires pour stocker et traiter l'ensemble N est de l'ordre de 200 Gio pour des paires de patchs 15×15

codées sur $32bits$.
Pour résoudre ce problème, nous considérons l'ensemble N^* de paires négatives que nous obtenons en prenant un Feature sur 3 du sous-ensemble X^i et un Feature sur 40 du sous-ensemble X^j. Les pas 3 et 40 ont été choisis de manière à obtenir des classes P et N^* équilibrées. Nous avons utilisé l'algorithme 2 pour la génération des échantillons de la classe N^*.
Nous obtenons $Card(N^*) = 1006200$. Les deux classes P et N^* sont donc bien équilibrées et le nombre d'échantillons ne pose pas de problème de stockage.
Le nombre total d'échantillons de l'ensemble des données est donc de $1,95\ M$ d'images de paires de Features. La figure 7.31 montre quelques échantillons aléatoires.

Algorithme 2 : Algorithme pour générer les échantillons de N^*

For $m = 1\,;\ m \leq 130\,;\ m++$
  For $n = m + 1\,;\ n \leq 130\,;\ n++$
    For $p = 1\,;\ p \leq 120\,;\ p = p + 3$
      For $q = 1\,;\ m \leq 120\,;\ q = q + 40$
        Échantillon négatif (x_p^m, x_q^n)

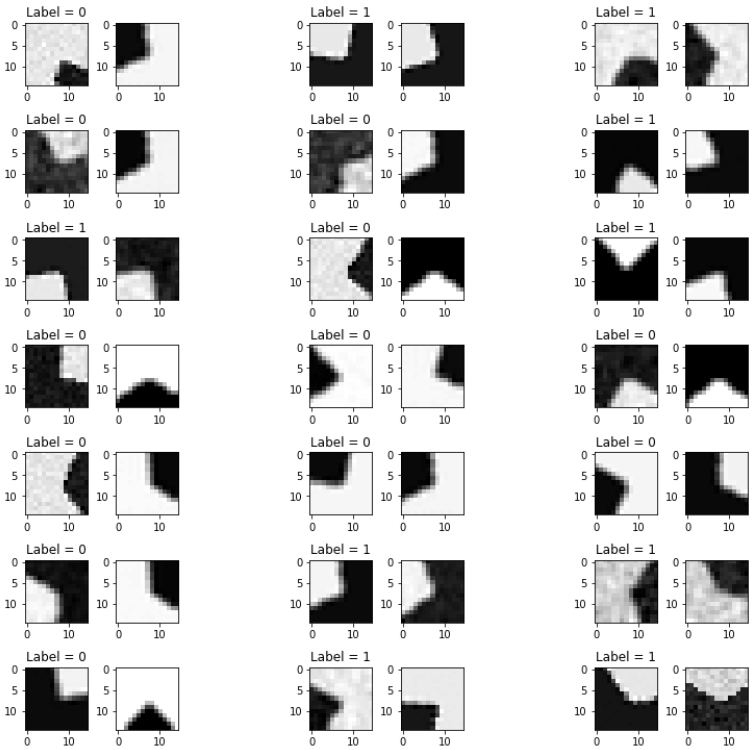

FIGURE 7.31 – Échantillons aléatoires de l'ensemble de données des paires de Features. Les paires similaires ont une étiquette 1 et les paires différentes ont une étiquette 0.

Les échantillons sont mélangés aléatoirement et stockés au format HDF5.
Nous avons utilisé cet ensemble de données pour entraîner le modèle convolutif siamois représenté sur la figure 7.32.

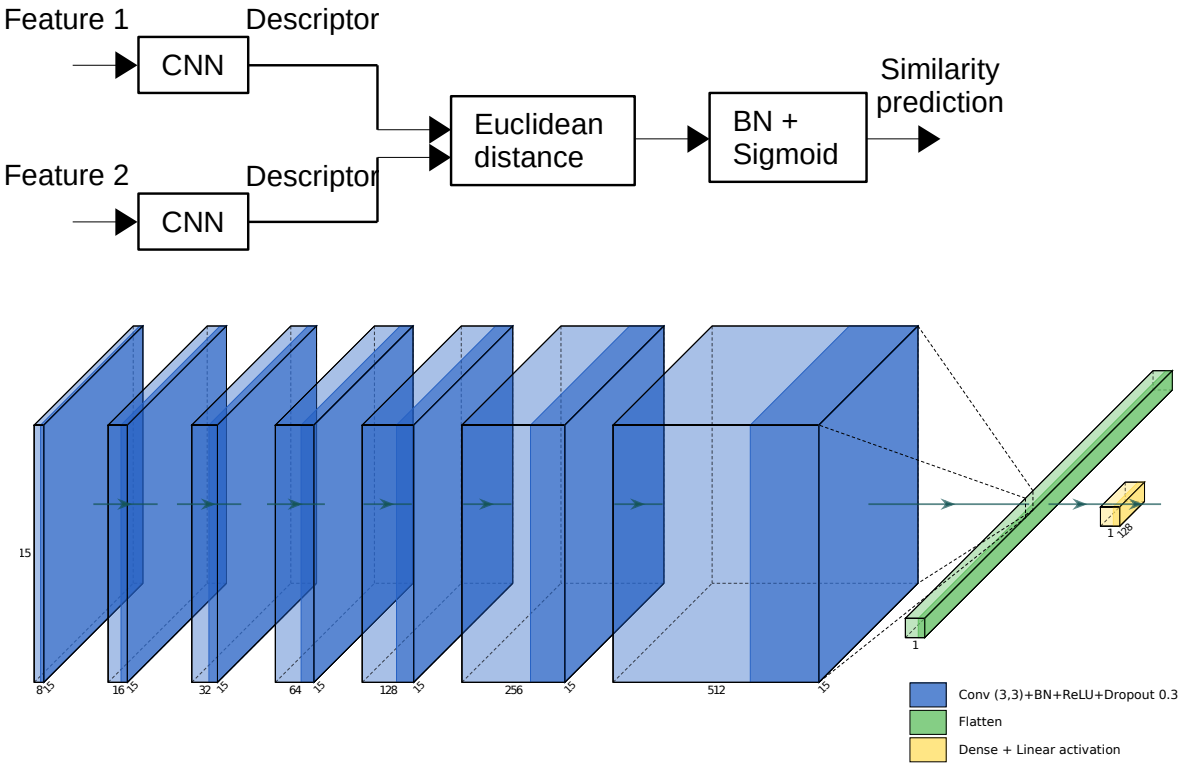

FIGURE 7.32 – Modèle convolutif siamois (en haut) et architecture CNN utilisée (en bas).

Le but d'un modèle siamois [TYRW14] est de différencier les patchs en maximisant la distance euclidienne entre les descripteurs de paires de Features différents :

$$dist(a, b) = ||f(a) - f(b)||_2^2, \tag{7.56}$$

avec a et b deux Features, et f(a) et f(b) les descripteurs correspondants obtenus à la sortie du CNN.
À cette fin, nous optimisons les CNN en utilisant une fonction de coût contrastive qui maximise le contraste (distance) entre les classes positive et négative :

$$L = (1 - Y_{vrai})Y_{prdit}^2 + Y_{vrai}(max(0, \alpha - Y_{pred}))^2 \tag{7.57}$$

avec $alpha = 1$ la marge qui garantit que la distance des paires négatives est supérieure à la distance des paires positives.
Y_{vrai} : Les étiquettes des paires.
Y_{pred} : La prédiction du modèle.
Notez que les deux CNNs ont la même architecture et apprennent les mêmes paramètres afin de calculer les descripteurs de la même manière.

Nous avons utilisé les paramètres suivants pour l'apprentissage : ensemble d'apprentissage de 70%, ensemble de validation de 15% ; ensemble de test de 15% ; taille du batch de 512 ; nombre d'itérations de 30 ; taux d'apprentissage décroissant entre 0.1 et 0.0001 ; Optimiseur Adam, fonction de coût contrastive, métrique Accuracy.
L'apprentissage a duré environ 7 heures sur GPU (Tesla P100-PCIE-16GB).
La figure 7.33 montre les courbes d'apprentissage et le tableau 7.4 les valeurs de la fonction de coût contrastive et de la mesure Accuracy obtenues à la fin de l'apprentissage.

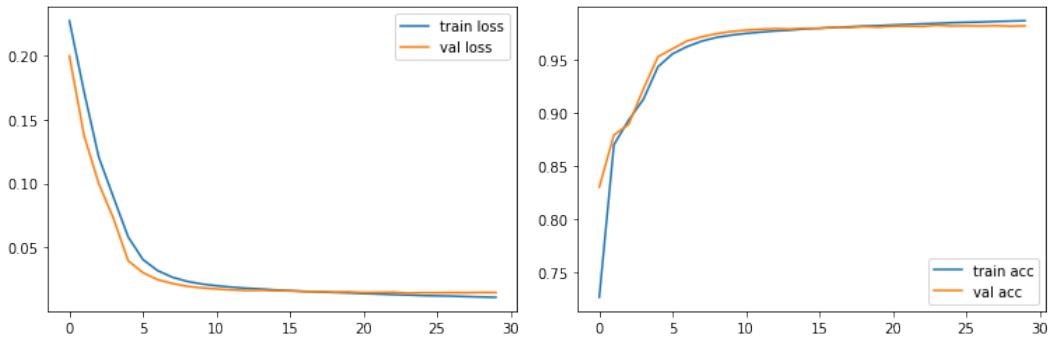

FIGURE 7.33 – Apprentissage du modèle siamois sur l'ensemble de données de paires de Features : Fonction de coût et Accuracy en fonction du nombre d'itérations, pour les ensembles d'apprentissage et de validation.

TABLE 7.4 – Fonction de coût contrastive et Accuracy après l'apprentissage du modèle siamois de mise en correspondance sur l'ensemble de données de paires de Features.

Fonction de coût			Accuracy		
Apprentissage	Validation	Test	Apprentissage	Validation	Test
0.011	0.0146	0.0148	98.59%	98.13%	98.09%

4.3 Estimation de l'homographie

Afin de tester la combinaison des deux modèles de détection et de mise correspondance et leur généralisation à de nouvelles images, nous avons appliqué différentes transformations à des images et estimé l'homographie à partir de la mise en correspondance des Features. Les valeurs aberrantes sont éliminées par RANSAC. La figure 7.34 montre quelques exemples d'estimation.

4.4 Conclusion

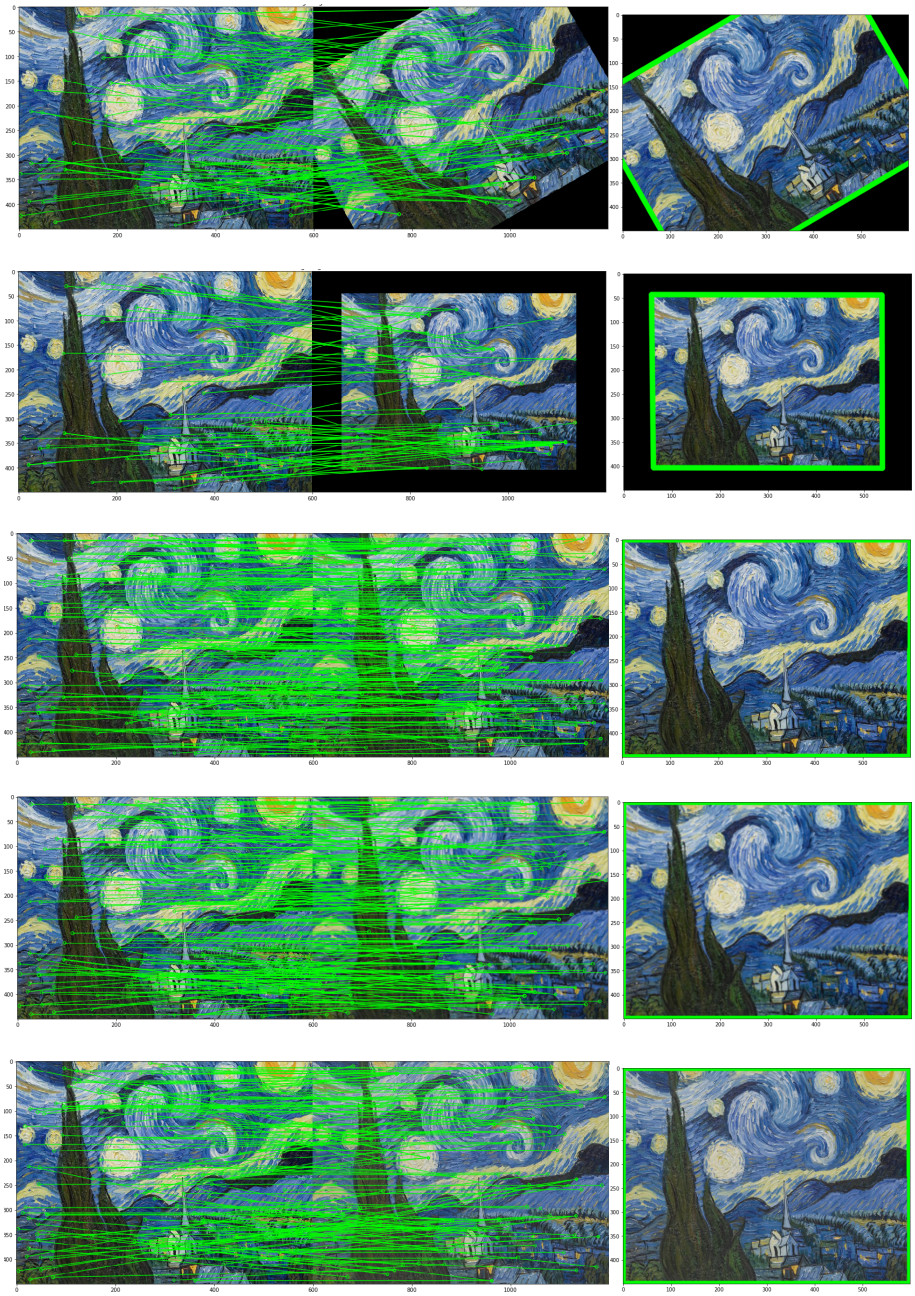

FIGURE 7.34 – De haut en bas : Simulation de plusieurs transformations de rotation 30°, échelle 0.8, bruit $\mathcal{N}(0, (0.001)^2)$, flou de mouvement avec un noyau 3×3 et changement linéaire de luminance. À gauche : mise en correspondance des Features détectés. À droite : estimation de l'homographie à partir des Features correspondants.

Nous avons proposé un modèle convolutif siamois permettant la description et la mise en correspondance des points d'intérêt détectés par la méthode de la section précédente. Nous complétons ainsi l'ensemble de la pipeline de mise en correspondance par des modèles CNN. L'évaluation quantitative

et qualitative du modèle atteste de son efficacité pour des applications tel que le suivi de mouvement. Les ensembles de données de synthèse proposés permettent une nouvelle approche pour entraîner les différents modules de mise en correspondance sans utiliser les méthodes traditionnelles. À notre connaissance, il s'agit des premiers ensembles de données de Features de synthèse pour la mise en correspondance d'images. Ces données ont été rendues publiques et mises à la disposition de la communauté.

Chapitre 8

Conclusion et perspectives

Les domaines des effets visuels et du cinéma n'ont cessé, depuis le début du numérique, de tirer profit des évolutions technologiques du monde de l'informatique. Aujourd'hui encore, avec la révolution de l'intelligence artificielle, le domaine dans lequel le "Deep Learning" a le plus montré son efficacité est celui de l'image. Cela est dû, d'un côté, à l'abondance de ce type de médias sur internet, ce qui permet d'améliorer l'apprentissage des modèles avec des ensembles de données de plus en plus large. De l'autre côté, la puissance de calcul et l'accessibilité des nouvelles cartes graphiques permettent l'apprentissage de modèles de plus en plus profonds et complexes.

Dans ce rapport, nous avons présenté différents algorithmes de vision par ordinateur et d'apprentissage profond appliqués aux VFX :

- Mise en correspondance des images.
- Synthèse et rendu d'images 3D.
- Détection de la pose humaine.
- Séparation d'objet de l'arrière plan.
- Composition des images.
- Restauration des images.

Ces algorithmes comptent parmi les outils les plus utilisés dans le domaine des effets visuels.
Dans le cadre de nos travaux de recherche, nous nous sommes intéressés à ces méthodes, car ils constituent les différents modules d'un système de Matchmoving ou de réalité augmentée, qui sont les deux axes principaux de nos recherches.

Nos principales contributions sont les suivantes :

- Adaptation de différents algorithmes pour des applications des effets visuels de réalité augmentée 2D et 3D et de stabilisation vidéo.
- Élaboration d'un système de pré-visualisation de Matchmoving fondé sur la détection et le suivi de marqueurs artificiels.

- Nouveau modèle CNN à fenêtre glissante de détection rapide et de description des points d'intérêt pour la mise en correspondance.
- Nouveau modèle CNN Siamois de mise en correspondance des points d'intérêt.
- Création de trois nouveaux ensembles de données très large d'images de synthèse pour l'apprentissage des modèles proposés.
- Étude comparative de différentes méthodes de synthèse et de rendu 3D.

Parmi nos perspectives :

- Pour les applications VFX, nous avons l'intention d'évaluer les méthodes proposées sur une base de données vidéo, qui comprendra également des objets en mouvement, en testant et en comparant d'autres algorithmes dans chaque module de la pipeline et de considérer l'aspect temps réel.
- Dans le cadre des travaux sur le Matchmoving, nous souhaitons tester d'autres méthodes pour la détection des Features, l'estimation de la pose et la calibration, afin d'améliorer la précision. Pour la qualité de rendu, nous envisageons de travailler sur l'aspect éclairage en considérant le type de matériaux et d'autres modèles d'éclairage. Nous avons également l'intention d'appliquer la méthode à des cas plus pratiques tels que la création d'environnements virtuels, et de considérer le cas des objets 3D animés.
- Concernant les modèles CNN proposés, nous souhaitons améliorer leurs performances pour une meilleure généralisation à d'autres ensembles de données d'images réelles et fournir des descripteurs plus robustes pour les images à large variations.
- Enfin, nous souhaitons tester les différents algorithmes sur d'autres applications d'effets visuels et dans d'autres domaines.

Bibliographie

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017. 83

[AF04] Nicholas Apostoloff and Andrew Fitzgibbon. Bayesian video matting using learnt image priors. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. IEEE, 2004. 99

[AFP07] John Arnold, Michael Frater, and Mark Pickering. *Digital television : technology and standards*. John Wiley & Sons, 2007. 1

[AKB08] Motilal Agrawal, Kurt Konolige, and Morten Rufus Blas. Censure : Center surround extremas for realtime feature detection and matching. In *European conference on computer vision*, pages 102–115. Springer, 2008. 20

[BA83] Peter J Burt and Edward H Adelson. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics (TOG)*, 2(4) :217–236, 1983. 102

[BAC+16] Filippo Bergamasco, Andrea Albarelli, Luca Cosmo, Emanuele Rodola, and Andrea Torsello. An accurate and robust artificial marker based on cyclic codes. *IEEE transactions on pattern analysis and machine intelligence*, 38(12) :2359–2373, 2016. 121

[Ber14] Marcelo Bertalmío. *Image processing for cinema*. CRC Press, 2014. 1

[BGPS07] Sebastiano Battiato, Giovanni Gallo, Giovanni Puglisi, and Salvatore Scellato. Sift features tracking for video stabilization. In *14th International Conference on Image Analysis and Processing (ICIAP 2007)*, pages 825–830. IEEE, 2007. 126

[BHW10] Matthew Brown, Gang Hua, and Simon Winder. Discriminative learning of local image descriptors. *IEEE*

transactions on pattern analysis and machine intelligence, 33(1) :43–57, 2010. 60

[BL07] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1) :59–73, 2007. 4

[BLVM17] Vassileios Balntas, Karel Lenc, Andrea Vedaldi, and Krystian Mikolajczyk. Hpatches : A benchmark and evaluation of handcrafted and learned local descriptors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5173–5182, 2017. 60

[BM09] Lubomir Bourdev and Jitendra Malik. Poselets : Body part detectors trained using 3d human pose annotations. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1365–1372. IEEE, 2009. 92

[BMB+13] Andreas Baak, Meinard Müller, Gaurav Bharaj, Hans-Peter Seidel, and Christian Theobalt. A data-driven approach for real-time full body pose reconstruction from a depth camera. In *Consumer Depth Cameras for Computer Vision*, pages 71–98. Springer, 2013. 92

[BRPM16] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *Bmvc*, volume 1, page 3, 2016. 48, 59

[BS11] Martin Bäuml and Rainer Stiefelhagen. Evaluation of local features for person re-identification in image sequences. In *2011 8th ieee international conference on advanced video and signal based surveillance (avss)*, pages 291–296. IEEE, 2011. 120

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf : Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006. 17, 19

[CB03] Wojciech Chojnacki and Michael J Brooks. Revisiting hartley's normalized eight-point algorithm. *IEEE transactions on pattern analysis and machine intelligence*, 25(9) :1172–1177, 2003. 124

[CBS18] Titus Cieslewski, Michael Bloesch, and Davide Scaramuzza. Matching features without descriptors : implicitly matched interest points. *arXiv preprint arXiv :1811.10681*, 2018. 48

[CCSS01] Yung-Yu Chuang, Brian Curless, David H Salesin, and Richard Szeliski. A bayesian approach to digital matting. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 2, pages II–II. IEEE, 2001. 100

[CHL05] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005. 54

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief : Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010. 10, 20

[CMM10] Hongping Cai, Krystian Mikolajczyk, and Jiri Matas. Learning linear discriminant projections for dimensionality reduction of image descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(2) :338–352, 2010. 48

[Coo86] Robert L Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)*, 5(1) :51–72, 1986. 80

[CPC84] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145, 1984. 80

[CTC+12] Vijay Chandrasekhar, Gabriel Takacs, David M Chen, Sam S Tsai, Yuriy Reznik, Radek Grzeszczuk, and Bernd Girod. Compressed histogram of gradients : A low-bitrate descriptor. *International journal of computer vision*, 96(3) :384–399, 2012. 16

[Data] http://www.cs.cornell.edu/projects/1dsfm/. 66

[Datb] https://data.mendeley.com/datasets/8jx8y9yfn5/1. 142, 151

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet : A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 48, 55

[DMR16] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Deep image homography estimation. *arXiv preprint arXiv :1606.03798*, 2016. 122

[DMR18] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint : Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 224–236, 2018. 48

[DRP+19] Mihai Dusmanu, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler.

D2-net : A trainable cnn for joint description and detection of local features. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pages 8092–8101, 2019. 48

[DSH00] Yves Dufournaud, Cordelia Schmid, and Radu Horaud. Matching images with different resolutions. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 1, pages 612–618. IEEE, 2000. 9

[ELJ18] Sovann En, Alexis Lechervy, and Frédéric Jurie. Rpnet : An end-to-end network for relative camera pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018. 129

[ERB+18] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394) :1204–1210, 2018. 83

[FB81] Martin A Fischler and Robert C Bolles. Random sample consensus : a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6) :381–395, 1981. 49, 121

[FDB14] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks : a comparison to sift. *arXiv preprint arXiv :1405.5769*, 2014. 48, 55

[Gav00] Dariu M Gavrila. Pedestrian detection from a moving vehicle. In *European conference on computer vision*, pages 37–49. Springer, 2000. 92

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010. 40

[GC18] V Scott Gordon and John L Clevenger. *Computer Graphics Programming in OpenGL with C++*. Stylus Publishing, LLC, 2018. 129

[GC20] V Scott Gordon and John L Clevenger. *Computer Graphics Programming in OpenGL with C++*. Stylus Publishing, LLC, 2020. 72

[GDDM15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate

object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1) :142–158, 2015. 25

[GJMSMCMC16] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Rafael Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51 :481–491, 2016. 129, 134

[GJMSMCMJ14] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6) :2280–2292, 2014. 121

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 82

[GPKT10] Varun Ganapathi, Christian Plagemann, Daphne Koller, and Sebastian Thrun. Real time motion capture using a single time-of-flight camera. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 755–762. IEEE, 2010. 92

[GWK05] Daniel Grest, Jan Woetzel, and Reinhard Koch. Nonlinear body pose estimation from depth images. In *Joint Pattern Recognition Symposium*, pages 285–292. Springer, 2005. 92

[HAA16] M Hassaballah, Aly Amin Abdelmgeid, and Hammam A Alshazly. Image features detection, description and matching. In *Image Feature Detectors and Descriptors*, pages 11–45. Springer, 2016. 118

[Hal12] Houssam Halmaoui. *Restauration d'images par temps de brouillard et de pluie : applications aux aides à la conduite*. PhD thesis, Université d'Evry-Val d'Essonne, 2012. 113

[HAM19] Eric Haines and Tomas Akenine-Möller. *Ray Tracing Gems : High-Quality and Real-Time Rendering with DXR and Other APIs*. Apress, 2019. 72, 80

[Har] https://github.com/DagnyT/hardnet. 64

[HGSH+17] Yannick Hold-Geoffroy, Kalyan Sunkavalli, Sunil Hadap, Emiliano Gambaretto, and Jean-François Lalonde. Deep outdoor illumination estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7312–7321, 2017. 129

[HH20a] Houssam Halmaoui and Abdelkrim Haqiq. Feature detection and tracking for visual effects : Augmented reality

and video stabilization. In *International Conference on Artificial Intelligence & Industrial Applications*, pages 291–311. Springer, 2020. 117

[HH20b] Houssam Halmaoui and Abdelkrim Haqiq. Matchmoving previsualization based on artificial marker detection. In *International Conference on Advanced Intelligent Systems and Informatics*, pages 79–89. Springer, 2020. 76, 128

[HH21] Houssam Halmaoui and Abdelkrim Haqiq. Convolutional sliding window based model and synthetic dataset for fast feature detection. In *The International Conference on Artificial Intelligence and Computer Vision*, pages 101–111. Springer, 2021. 140

[Hom] https://www.robots.ox.ac.uk/~vgg/hzbook/code/. 127

[HS+88] Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. Citeseer, 1988. 5, 140

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. 122, 133, 134

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 41

[IF01] Sergey Ioffe and David A. Forsyth. Probabilistic methods for finding people. *International Journal of Computer Vision*, 43(1) :45–68, 2001. 92

[JGB08] Michael Jahrer, Michael Grabner, and Horst Bischof. Learned local descriptors for recognition and matching. In *Computer Vision Winter Workshop*, volume 2, 2008. 48, 52

[Kaj86] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986. 80

[KDRZ20] Mohammad Rami Koujan, Michail Christos Doukas, Anastasios Roussos, and Stefanos Zafeiriou. Head2head : Video-based neural head synthesis. In *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020)*, pages 16–23. IEEE, 2020. 86

[KE19] Yoshihiro Kanamori and Yuki Endo. Relighting humans : occlusion-aware inverse rendering for full-body human images. *arXiv preprint arXiv :1908.02714*, 2019. 85

[KGC15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet : A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE*

international conference on computer vision, pages 2938–2946, 2015. 129

[KGT+18] Hyeongwoo Kim, Pablo Garrido, Ayush Tewari, Weipeng Xu, Justus Thies, Matthias Niessner, Patrick Pérez, Christian Richardt, Michael Zollhöfer, and Christian Theobalt. Deep video portraits. *ACM Transactions on Graphics (TOG)*, 37(4) :1–14, 2018. 85

[KHS10] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3d mesh segmentation and labeling. In *ACM SIGGRAPH 2010 papers*, pages 1–12. 2010. 92

[KK17] Manikanta Kotaru and Sachin Katti. Position tracking for virtual reality using commodity wifi. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 68–78, 2017. 129

[KLA+20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020. 82

[KS04] Yan Ke and Rahul Sukthankar. Pca-sift : A more distinctive representation for local image descriptors. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–II. IEEE, 2004. 48, 49

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25 :1097–1105, 2012. 25

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6) :84–90, 2017. 39

[KSS16] John Kessenich, Graham Sellers, and Dave Shreiner. *OpenGL Programming Guide : The official guide to learning OpenGL, version 4.5*. Addison-Wesley Professional, 2016. 78

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998. 38

[LCS11] Stefan Leutenegger, Margarita Chli, and Roland Siegwart. Brisk : Binary robust invariant scalable keypoints. In *2011*

IEEE international conference on computer vision (ICCV), pages 2548–2555. Ieee, 2011. 10

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv :1312.4400*, 2013. 42

[LHK+19] Jaehyun Lee, Jahanzeb Hafeez, Kwangjib Kim, Seunghyun Lee, and Soonchul Kwon. A novel real-time match-moving method with hololens. *Applied Sciences*, 9(14) :2889, 2019. 129

[Lin98] Tony Lindeberg. Feature detection with automatic scale selection. *International journal of computer vision*, 30(2) :79–116, 1998. 8

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2) :91–110, 2004. 10

[LP09] Nan Liu and Ming-Yong Pang. A survey of shadow rendering algorithms : projection shadows and shadow volumes. In *2009 Second International Workshop on Computer Science and Engineering*, volume 1, pages 488–492. IEEE, 2009. 78

[LS19] Minhyeok Lee and Junhee Seok. Controllable generative adversarial network. *Ieee Access*, 7 :28158–28169, 2019. 83

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 25

[LSS+19] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes : Learning dynamic renderable volumes from images. *arXiv preprint arXiv :1906.07751*, 2019. 84

[MAT17] Raul Mur-Artal and Juan D Tardós. Orb-slam2 : An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5) :1255–1262, 2017. 4

[MCUP04] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10) :761–767, 2004. 12

[MMRM17] Anastasiya Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. Working hard to know your neighbor's margins : Local descriptor learning loss. *arXiv preprint arXiv :1705.10872*, 2017. 48, 63

[Mor80] Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, Stanford Univ CA Dept Of Computer Science, 1980. 4

[MS04] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *International journal of computer vision*, 60(1) :63–86, 2004. 9, 17, 129

[MS05] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10) :1615–1630, 2005. 16, 55

[MS18] Steve Marschner and Peter Shirley. *Fundamentals of computer graphics*. CRC Press, 2018. 76

[MTS+05] Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, Jiri Matas, Frederik Schaffalitzky, Timor Kadir, and Luc Van Gool. A comparison of affine region detectors. *International journal of computer vision*, 65(1) :43–72, 2005. 55

[MY09] Jean-Michel Morel and Guoshen Yu. Asift : A new framework for fully affine invariant image comparison. *SIAM journal on imaging sciences*, 2(2) :438–469, 2009. 17

[MYKR17] Iaroslav Melekhov, Juha Ylioinas, Juho Kannala, and Esa Rahtu. Relative camera pose estimation using convolutional neural networks. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 675–687. Springer, 2017. 129

[NAS+17] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-scale image retrieval with attentive deep local features. In *Proceedings of the IEEE international conference on computer vision*, pages 3456–3465, 2017. 48

[OS08] Ryuzo Okada and Björn Stenger. A single camera motion capture system for human-computer interaction. *IEICE TRANSACTIONS on Information and Systems*, 91(7) :1855–1862, 2008. 92

[OTFY18] Yuki Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. Lf-net : Learning local features from images. *arXiv preprint arXiv :1805.09662*, 2018. 48

[PGKT10] Christian Plagemann, Varun Ganapathi, Daphne Koller, and Sebastian Thrun. Real-time identification and localization of body parts from depth images. In *2010 IEEE International Conference on Robotics and Automation*, pages 3108–3113. IEEE, 2010. 92

[PGZ+19] Julien Philip, Michaël Gharbi, Tinghui Zhou, Alexei A Efros, and George Drettakis. Multi-view relighting using a geometry-aware network. *ACM Transactions on Graphics*, 38 :1–14, 2019. 85

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering : From theory to implementation*. Morgan Kaufmann, 2016. 72

[PLWZ19] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019. 82

[PM90] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7) :629–639, 1990. 8

[Rad13] Richard J Radke. *Computer vision for visual effects*. Cambridge University Press, 2013. 1, 128, 130

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006. 10

[RDL+15] Peiran Ren, Yue Dong, Stephen Lin, Xin Tong, and Baining Guo. Image based relighting using neural networks. *ACM Transactions on Graphics (ToG)*, 34(4) :1–12, 2015. 85

[RMC15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *preprint arXiv :1511.06434*, 2015. 83

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R Bradski. Orb : An efficient alternative to sift or surf. In *ICCV*, volume 11, page 2. Citeseer, 2011. 10, 20

[RRMSMC18] Francisco J Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and vision Computing*, 76 :38–47, 2018. 121, 129

[RTPS16] Jason R Rambach, Aditya Tewari, Alain Pagani, and Didier Stricker. Learning to fuse : A deep learning approach to visual-inertial camera pose estimation. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 71–76. IEEE, 2016. 129

[SA20] M Segal and K Akeley. The opengl graphics system : A specification. version 4.6, core profile. the khronos group inc., 2006-2018, 2020. 76

[SB96] Alvy Ray Smith and James F Blinn. Blue screen matting. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 259–268, 1996. 97

[SBT+19] Tiancheng Sun, Jonathan T Barron, Yun-Ta Tsai, Zexiang Xu, Xueming Yu, Graham Fyffe, Christoph Rhemann, Jay Busch, Paul E Debevec, and Ravi Ramamoorthi. Single image portrait relighting. *ACM Trans. Graph.*, 38(4) :79–1, 2019. 85

[SDMR20] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue : Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4938–4947, 2020. 48

[SEZ+13] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat : Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv :1312.6229*, 2013. 140

[SF16] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016. 4

[Shi16] Peter Shirley. Ray tracing in one weekend. *Amazon Digital Services LLC*, 1, 2016. 80

[SIVA17] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017. 43

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 41

[SM10] Matheen Siddiqui and Gérard Medioni. Human pose estimation from a single view point, real-time range sensor. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops*, pages 1–8. IEEE, 2010. 92

[SSTF+15] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *Proceedings of the IEEE international*

conference on computer vision, pages 118–126, 2015. 59, 67

[SVZ14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8) :1573–1585, 2014. 48, 50, 52

[SWJH13] Graham Sellers, Richard S Wright Jr, and Nicholas Haemel. *OpenGL superBible : comprehensive tutorial and reference*. Addison-Wesley, 2013. 79

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*, 2014. 40, 57, 144

[TFe] https://github.com/vbalnt/tfeat. 60

[TFT+20] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, et al. State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley, 2020. 72

[TFW17] Yurun Tian, Bin Fan, and Fuchao Wu. L2-net : Deep learning of discriminative patch descriptor in euclidean space. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 661–669, 2017. 48, 60

[TLF09] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy : An efficient dense descriptor applied to wide-baseline stereo. *IEEE transactions on pattern analysis and machine intelligence*, 32(5) :815–830, 2009. 16

[TS04] Prithiraj Tissainayagam and David Suter. Assessing the performance of corner detectors for point feature tracking applications. *Image and Vision computing*, 22(8) :663–679, 2004. 118

[TYF+19] Yurun Tian, Xin Yu, Bin Fan, Fuchao Wu, Huub Heijnen, and Vassileios Balntas. Sosnet : Second order similarity regularization for local descriptor learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11016–11025, 2019. 48, 64

[TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface : Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1701–1708, 2014. 156

[Vla71] Petro Vlahos. Electronic composite photography, 1971. US3595987A. 96

[VYFL15] Yannick Verdie, Kwang Yi, Pascal Fua, and Vincent Lepetit. Tilde : A temporally invariant learned detector. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5279–5288, 2015. 67

[WDT+09] Jiaping Wang, Yue Dong, Xin Tong, Zhouchen Lin, and Baining Guo. Kernel nyström method for light transport. In *ACM SIGGRAPH 2009 papers*, pages 1–10. 2009. 85

[Whi05] Turner Whitted. An improved illumination model for shaded display. In *ACM Siggraph 2005 Courses*, pages 4–es. 2005. 80

[WM19] Chris Wyman and Adam Marrs. Introduction to directx raytracing. In *Ray Tracing Gems*, pages 21–47. Springer, 2019. 129

[WO16] John Wang and Edwin Olson. Apriltag 2 : Efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4193–4198. IEEE, 2016. 121

[WS14] Kyle Wilson and Noah Snavely. Robust global translations with 1dsfm. In *European conference on computer vision*, pages 61–75. Springer, 2014. 66

[Wu13] Changchang Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 127–134. IEEE, 2013. 66

[WZX+16] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 82–90, 2016. 82

[XSHR18] Zexiang Xu, Kalyan Sunkavalli, Sunil Hadap, and Ravi Ramamoorthi. Deep image-based relighting from optimal sparse samples. *ACM Transactions on Graphics*, 37(4) :1–13, 2018. 85

[YNHB20] Tsun-Yi Yang, Duy-Kien Nguyen, Huub Heijnen, and Vassileios Balntas. Ur2kid : Unifying retrieval, keypoint detection, and keypoint description without local correspondence supervision. *arXiv preprint arXiv :2001.07252*, 2020. 48

[YTLF16] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift : Learned invariant feature transform. In *European*

conference on computer vision, pages 467–483. Springer, 2016. 48, 66

[YTO+18] Kwang Moo Yi, Eduard Trulls, Yuki Ono, Vincent Lepetit, Mathieu Salzmann, and Pascal Fua. Learning to find good correspondences. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2666–2674, 2018. 48

[YVFL16] Kwang Moo Yi, Yannick Verdie, Pascal Fua, and Vincent Lepetit. Learning to assign orientations to feature points. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 107–116, 2016. 67

[ZCKB12] Andrew Ziegler, Eric Christiansen, David Kriegman, and Serge Belongie. Locally uniform comparison image descriptor. *Advances in Neural Information Processing Systems*, 25 :1–9, 2012. 21

[ZF07] Youding Zhu and Kikuo Fujimura. Constrained optimization for human pose estimation from depth sequences. In *Asian Conference on Computer Vision*, pages 408–418. Springer, 2007. 92

[Zha00] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11) :1330–1334, 2000. 129, 131, 132, 134

[ZK15] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2015. 48, 57

[ZSBL19] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9459–9468, 2019. 82, 86

Computer vision and deep learning algorithms applied to visual effects and augmented reality

Abstract : Computer vision algorithms and more recently machine learning have been successfully applied in the field of audiovisual and cinema (image restoration and de-noising, oversampling, slow motion, compression, color correction, HDR imaging or stereoscopic cinema) and more particularly in the field of visual effects VFX (image retouching, object separation from the background, motion tracking, body and face motion capture or 3D modeling and animation). Our research activities focus on the VFX technique of "Matchmoving" which consists in combining images from real and synthetic scenes. The methods used are similar to those of augmented reality, with the difference that the realism effect is important in the case of VFX. Matchmoving can be summarized in three steps : 1) Tracking of the camera movement to allow the insertion of the CG images in a geometrically coherent way with the real scene. 2) Composition of the images with Alpha Matting techniques. 3) Photo-realistic rendering of the synthesized images in a manner consistent with the photometry of the real scene. In a first step, we studied different existing algorithms for each of these steps. We applied traditional motion tracking algorithms to VFX and augmented reality problems. Then, we combined different algorithms to propose a real-time Matchmoving pre-visualization system based on artificial marker tracking and rasterization rendering techniques. A theoretical and comparative study of traditional and machine learning based matching methods has been performed in order to choose the most suitable approaches for an augmented reality system. New methods based on deep learning have been proposed. We have developed new CNN models as well as various synthetic image datasets to allow the detection, description and matching of points of interest (main steps of camera motion tracking). The datasets created were made public and available to the community. Finally, we proposed a comparative study of 3D rendering methods of rasterization, Ray Tracing and deep learning.

Keywords : Computer vision ; Deep learning ; Image matching ; 3D rendering ; Matchmoving, Augmented reality ; Visual effects.

Algorithmes de vision par ordinateur et d'apprentissage profond appliqués aux effets visuels et à la réalité augmentée

Résumé : Les algorithmes de vision par ordinateur et plus récemment l'apprentissage automatique ont été appliqués avec succès dans le domaine de l'audiovisuel et du cinéma (restauration et dé-bruitage d'images, sur-échantillonnage, slow motion, compression, correction des couleurs, imagerie HDR ou cinéma stéréoscopique) et plus particulièrement dans le domaine des effets visuels VFX (retouche d'images, séparation des objets de l'arrière plan, suivi de mouvement, capture de mouvement du corps et des visages ou la modélisation et l'animation 3D). Nos activités de recherche portent essentiellement sur la technique VFX de "Matchmoving" qui consiste à combiner des images de scènes réelles et de synthèse. Les méthodes utilisées sont similaires à celles de la réalité augmentée, à la différence que l'effet de réalisme est important dans le cas des VFX. Le Matchmoving peut se résumer en trois étapes : 1) Suivi du mouvement caméra pour permettre l'insertion des images de synthèse de manière géométriquement cohérente avec la scène réelle. 2) Composition des images avec des techniques de Alpha Matting. 3) Rendu photo-réaliste des images de synthèse de manière cohérente avec la photométrie de la scène réelle.
Dans un premier temps, nous avons étudié différents algorithmes existants de chacune de ces étapes. Nous avons appliqué des algorithmes traditionnels de suivi de mouvement à des problématiques de VFX et de réalité augmentée. Ensuite, nous avons combiné différents algorithmes pour proposer un système de pré-visualisation de Matchmoving en temps réel fondé sur le suivi de marqueurs artificiels et sur des techniques de rendu par rastérisation. Une étude théorique et comparative des méthodes de mise en correspondance traditionnelles et fondées sur l'apprentissage automatique et profond a été réalisée dans le but de choisir les approches les plus adaptée pour un système de réalité augmentée. Des nouvelles méthodes fondées sur l'apprentissage profond ont été proposées. Nous avons élaboré de nouveaux modèles CNN ainsi que divers datasets d'images de synthèse pour permettre la détection, la description et la mise en correspondance des points d'intérêt (étapes principales du suivi de mouvement caméra). Les datasets créées ont été rendues publiques et mises à la disposition de la communauté. Enfin, nous avons proposé une étude comparative des méthodes de rendu 3D de rastérisation, de Ray Tracing et d'apprentissage profond.

Mots Clés : Vision par ordinateur ; Apprentissage profond ; Mise en correspondance image ; Rendu 3D ; Matchmoving, Réalité augmentée ; Effets visuels.