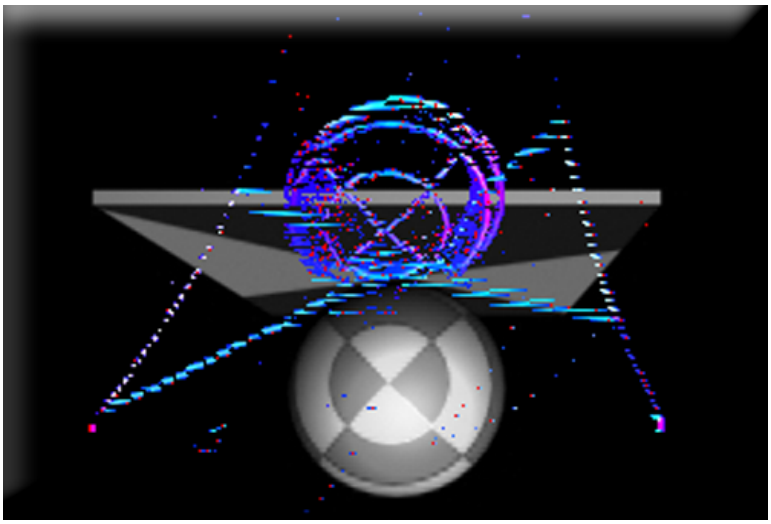
Projet IA54-IN54 :

Mise en correspondance d'images stéréoscopiques par une approche multi-agents



Béatrice Frey, Audrey Cholewka, Cyril Crassin
GI05 - Automne 2005
Suiveur IA54 : Olivier Simonin
Suiveur IN54 : Franck Gechter

Table des matières

Introduction

Dans le cadre du projet véhicule intelligent mené par le laboratoire Systèmes et Transports de l'UTBM, un certain nombre de recherches sont menées dans le domaine de la vision et du traitement d'images. Le but de ces recherches est de permettre d'améliorer l'autonomie de systèmes de transport en les dotant de moyens élaborés de détection et de prise en compte de l'environnement. Le projet que nous avons mené dans le cadre des UV IA54 (Systèmes multi-agents) et IN54 (Reconnaissance de formes) s'inscrit dans cette optique et concerne la détection d'obstacles à l'avant d'un véhicule par vision stéréoscopique. Notre travail s'est focalisé plus particulièrement sur l'algorithme de mise en correspondance au coeur de la vision stéréoscopique permettant de reconstituer la profondeur d'une scène observée.

Ce problème a fait l'objet de plusieurs publications par le laboratoire ([HRK03]), publications présentant une méthode efficace et temps réel basée sur une procédure de vote. En liaison avec le cours d'IA54, notre travail a consisté à concevoir, développer et expérimenter une approche nouvelle de ce problème basée sur une méthode multi-agents.

Notre première démarche a consisté à nous documenter sur le principe général de vision stéréoscopique et sur l'ensemble des techniques de mise en correspondance existantes. Nous avons ensuite effectué une étude plus poussée de la technique de vote afin d'en extraire les principes potentiellement transposables dans notre approche. A partir des éléments retenus et des concepts de programmation multi-agents étudiés en cours, nous avons dans une deuxième étape fixé avec nos suiveurs les objectifs à atteindre et conçu une méthode de résolution basée sur un ensemble d'agents réactifs au sein d'une architecture de traitement d'images. Cette architecture a ensuite été implémentée en C++ sous la forme d'un module de traitement doté d'un noyau multi-agents et s'intégrant dans un pipeline de traitement graphique générique. Deux algorithmes multi-agents ont été mis en oeuvre et un travail important de calibrage et d'expérimentation a été mené afin d'optimiser le temps de traitement ainsi que la précision de la mise en correspondance et l'adaptabilité de la méthode au traitement d'images issues de divers types d'environnements.

Chapitre 1

Présentation du problème

1.1 Principe général

En stéréovision, les deux problèmes à résoudre sont l'identification des points homologues entre les deux images ou mise en correspondance (en anglais matching) et la calibration du système de prise de vue, c'est à dire sa modélisation dans l'espace. Les deux étapes fondamentales de mise en correspondance des images et de calibration mènent au calcul d'un point par triangulation (intersection de deux droites dans l'espace).

Le principe général d'un algorithme de mise en correspondance est de trouver, dans deux images stéréoscopiques, les paires de primitives qui correspondent à un même objet dans l'espace 3D. Pour faire face à ce problème, qui est de nature combinatoire, on utilise des contraintes qui sont liées à la scène 3D observée ou encore à la géométrie du capteur stéréoscopique. Il nous a été proposé d'aborder le problème de mise en correspondance avec une approche multi-agents pouvant s'inspirer de l'Eco-résolution (cf. section 1.5.2). L'idée est de faire interagir les couples de primitives candidats à l'appariement (en tant qu'agents) via les contraintes stéréoscopiques pour faire émerger les bonnes mises en correspondance.

1.2 La stéréoscopie

La stéréoscopie ou vision en relief, est obtenue en calculant les points dans l'espace à l'aide de deux images observées à partir de points de vue différents. Pour cela, on utilise généralement un modèle simplifié pour représenter une caméra qui est le modèle sténopé ou trou d'épingle ([Gec99]). Ce modèle est formé d'un centre optique et d'un plan image et tous les rayons lumineux issus de l'objet observé dans l'espace passent par le centre optique et se propagent en ligne droite.

1.2.1 Principe

Pour deux appareils de prise de vue de type sténopé (cf. figure 1.1), la projection du point M sur un plan image est l'intersection du plan avec la droite passant par le centre optique et le point M. Les points M, Mg, Cg, Md, Cd sont donc coplanaires. La position d'un point M de l'espace est obtenue en calculant l'intersection des droites (Mg, Cg) et (Md, Cd) qui passent par l'image du point et son centre optique associé dans chaque appareil de prise de vue.

Ce calcul n'est possible que si l'on connaît la position respective des appareils de prise de vue dans l'espace ainsi que les projections du point M sur les deux plans images.

La position d'un appareil de prise de vue par rapport à l'autre revient à calculer le changement de repère entre leurs deux référentiels soit 3 paramètres qui déterminent la rotation entre les axes des deux repères et 3 paramètres qui déterminent la translation entre les origines des deux repères. Ces six paramètres constituent les paramètres extrinsèques ou paramètres extérieurs du système de prise de vue.

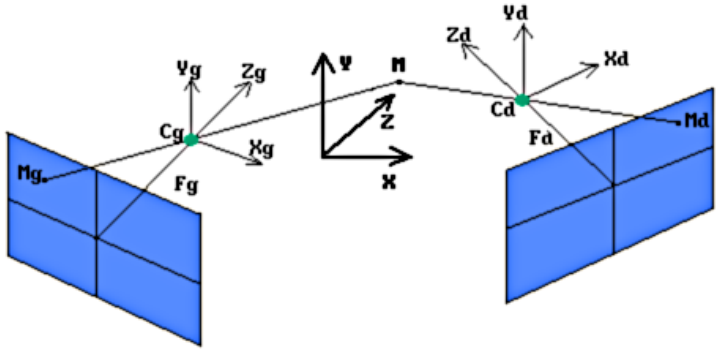


FIG. 1.1 – Schéma de principe de la vision stéréoscopique

1.2.2 Géométrie épipolaire

La géométrie épipolaire permet de simplifier les calculs de mise en correspondance en caractérisant les liens entre deux images stéréoscopiques. Dans cette géométrie, l'homologue d'un point de l'image de gauche -réciproquement de l'image droite- se trouve sur une droite connue dans l'image de droite -réciproquement dans l'image de gauche- (cf. figure 1.2). Le plan passant par un point dans l'environnement (P) et les points images (p et p') est alors appelé plan épipolaire. En traçant tous ces plans, on constate qu'ils se croisent en un point dans chaque image, les épipoles (e et e') qui correspondent à la projection du centre optique de l'image opposée. Une droite épipolaire (l ou l') d'une image est l'ensemble des points qui correspondent potentiellement à un point de l'autre image. Deux points d'une droite épipolaire gauche ont leur correspondant dans une même droite épipolaire droite (et inversement).

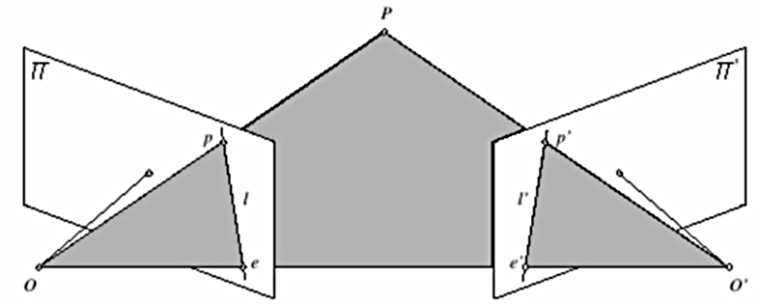


FIG. 1.2 – Schéma de principe de la géométrie épipolaire

Nous nous sommes limités pour nos travaux à un modèle de stéréoscope idéal dont les lignes épipolaires sont alignées et dont la distorsion radiale des lentilles des caméras peut être considérée comme négligeable. Autrement dit, lorsque les droites épipolaires sont parallèles et les droites épipolaires conjuguées sont confondues. Dans ce cas, les deux épipôles se trouvent à l'infini. Dans le cas du non respect de ces contraintes par le capteur stéréoscopique, il faudra nécessairement précéder le traitement d'une rectification d'image qui aligne les lignes épipolaires.

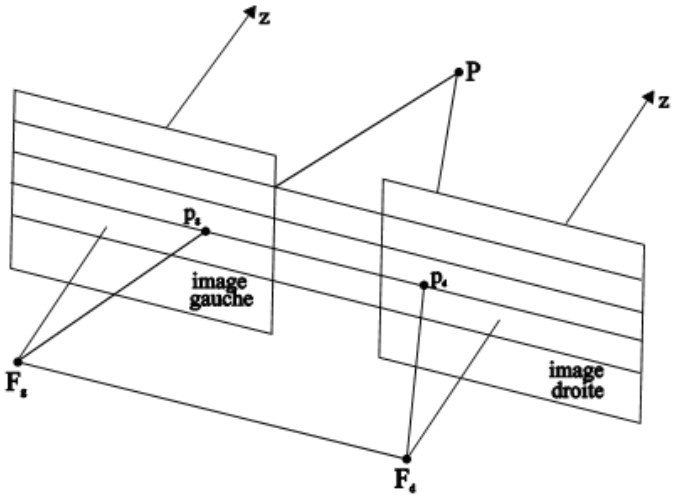


FIG. 1.3 – Illustration du modèle idéal avec droites épipolaires parallèles

1.2.3 Paramètres intrinsèques

Outre les paramètres extrinsèques qui définissent la géométrie du capteur stéréoscopique, en particulier la position et l'orientation de chaque caméra par rapport à la scène, d'autres paramètres doivent être choisis judicieusement afin d'optimiser, en fonction de l'application, le système de prise de vue. Il s'agit des paramètres correspondant aux caractéristiques propres des caméras :

- Distance focale des objectifs.
- Distance entre les centres optiques.
- Taille et résolution des capteurs.

Les deux premiers paramètres interviennent lors des calculs de triangulation géométrique. Ils permettent aussi de déterminer :

- La taille de la scène observable à une distance donnée.
- La distance minimale de prise de vue permettant d'appliquer le principe de stéréovision.

1.3 Méthodes de mise en correspondance

Il n'existe pas une méthode de mise en correspondance suffisamment générale pour pouvoir être appliquée à une paire d'image stéréoscopique indépendamment du contenu et du type de ces images. Mise à part la méthode de vote que nous aborderons dans la section suivante, il existe 4 méthodes principales dont voici les principales caractéristiques ([Rab00]) :

- **Mise en correspondance hiérarchique** : Cette méthode a été développée pour la reconstruction de terrain à partir de 2 vues aériennes. A partir d'une paire d'images, on produit une hiérarchie d'images. Cette hiérarchie consiste en deux pyramides d'images. Chaque pyramide a n niveaux de précision du plus fin (ou haute résolution) au plus grossier (ou basse résolution). Une sélection de caractéristiques est réalisée dans l'image gauche à partir de l'image de résolution inférieure ce qui permet d'initialiser une zone de recherche pour trouver le correspondant du point recherché à cette résolution dans l'image droite. On recherche un correspondant à ce point dans l'image de droite au niveau de plus basse résolution et cette stratégie est poursuivie jusqu'à atteindre l'image de plus haute résolution.

- **Mise en correspondance par programmation dynamique** : La programmation dynamique est une technique utile pour appareiller deux séquences tout en respectant l'ordre des éléments à l'intérieur de chaque séquence. En effet, si on suppose que la scène observée contient peu de surfaces transparentes, l'ordre des projections est le même dans les deux images. L'algorithme cherche alors le meilleur appariement entre les deux séquences parmi tous les appariements possibles. Pour trouver ce meilleur appariement, on ajoute une autre contrainte qui est la contrainte d'unicité (cf. section 1.4.2).

- **Mise en correspondance par relaxation** : Le problème de mise en correspondance peut être vu comme un problème d'étiquetage : aux points caractéristiques d'une image, on associe les points caractéristiques de l'autre image. On est alors amené à faire coopérer les points d'une image entre eux en mettant en oeuvre des relations découlant des contraintes géométriques/physiques. La mise en correspondance de ces points dépend du point considéré ainsi que de son voisinage. Le terme de relaxation vient du fait que, pour chaque itération, on cherche une combinaison entre les étiquettes qui minimise un critère calculé sur l'ensemble des étiquettes.

- **Mise en correspondance par invariants** : Le principe de cette méthode repose sur le fait que chaque point réel s'étant projeté dans les plans images possède des caractéristiques invariantes (par exemple l'intensité lumineuse). On caractérise ainsi chaque point de chaque image avec une liste d'invariants et, à l'aide de la contrainte épipolaire, on cherche à mettre en correspondance les points qui offrent une liste d'invariants similaire.

1.4 Méthode de vote pour la mise en correspondance stéréoscopique

1.4.1 Présentation

Cette méthode a été introduite en 2003 par une équipe du SET ([HRK03]) dans le cadre de la détection d'obstacles pour le véhicule intelligent. Dans un contexte de vision artificielle basé sur des caméras matricielles (caméras standards produisant des images bi-dimensionnelles de la scène), le problème de mise en correspondance est généralement simplifié en posant des hypothèses sur le type d'objet observé et l'environnement visuel ce qui permet de réduire substantiellement le nombre de caractéristiques à appareiller et d'obtenir ainsi des algorithmes temps réel. Mais ce type de simplification ne peut pas être utilisé dans la détection d'obstacle à l'avant d'un véhicule mobile dans un environnement routier. Afin de diminuer la quantité de données à traiter et permettre ainsi un traitement temps réel des images, des caméras linéaires sont utilisés à la place des caméras matricielles. Ce type de caméra produit des images mono-dimensionnelles qui correspondent à une ligne horizontale de la scène observée et elles s'avèrent en effet généralement suffisantes pour la détection d'obstacles en environnement routier. La méthode proposée s'appuie sur une matrice de mise en correspondance sur laquelle un algorithme de vote est appliqué afin de déterminer les appariements valides.

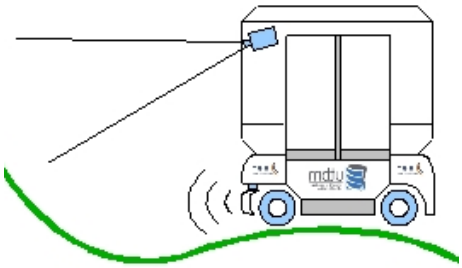


FIG. 1.4 – Le véhicule intelligent

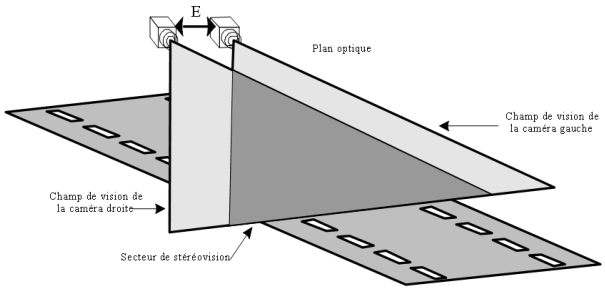


FIG. 1.5 – Mise en situation des capteurs stéréoscopiques sur le véhicule intelligent

1.4.2 Détails de l'algorithme

Extraction de caractéristiques

La première étape d'un algorithme de mise en correspondance est l'extraction de caractéristiques dans les images à appareiller. Cette extraction permet de limiter les données traitées aux points les plus caractéristiques d'une scène. Elle consiste en un premier filtrage de lissage permettant d'éliminer le bruit le plus important présent dans les images (cf. 2.2.2). Une détection de contour est ensuite effectuée sur les images linéaires par convolution à l'aide de l'opérateur de *Deriche*

([Der90], cf. 2.2.1) qui permet d'extraire les gradients en chaque point des images. Une dernière étape de seuillage est appliquée pour éliminer les gradients les moins pertinents. Les gradients de plus fortes amplitudes par intervalle de signe constant sont retenus et une liste de caractéristiques pour chacune des images est créée.

Matrice de mise en correspondance

Suite à cette extraction, les gradients de l'image gauche et de l'image droite sont mis en relation via une matrice de mise en correspondance bidimensionnelle dans laquelle chaque élément correspond à un couple d'appariement potentiel (chaque caractéristiques de l'image gauche est associée à l'ensemble des caractéristiques de l'image droite). Le problème se ramène ensuite à la détermination des bons appariement satisfaisant des contraintes locales et globales.

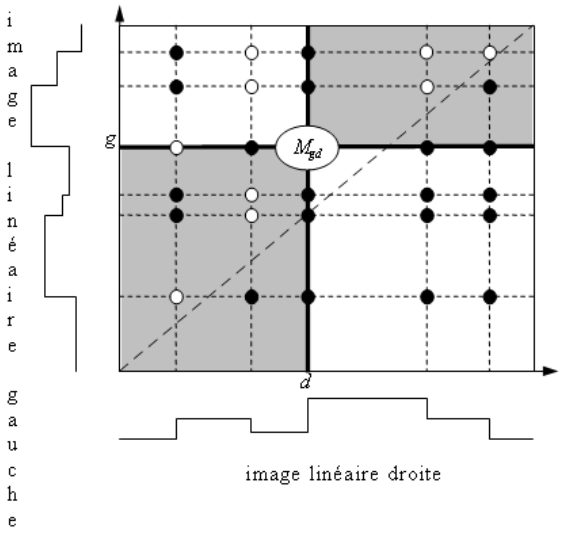


FIG. 1.6 – Matrice de mise en correspondance

Contraintes locales

Afin d'optimiser la recherche de correspondances valides, un certain nombre de contraintes locales sont utilisées en amont de l'algorithme de vote pour éliminer les appariements non conformes. La première contrainte utilisée est appelée *contrainte géométrique*, elle stipule que les gradients g et d, de coordonnées x_g et x_d, sont appareillables si, et seulement si, $x_g > x_d$. Cette contrainte est liée au principe même de la vision stéréoscopique (cf. 1.2). La deuxième contrainte est la *contrainte de signe* qui interdit l'appariement de deux gradients de signes contraires. Via ces contraintes, on retient dans la matrice de mise en correspondance uniquement les couples valides utilisés dans la méthode de vote.

La méthode de vote

Un processus de vote, dans lequel chaque élément valide M_{gd} de la matrice est considéré à la fois comme votant et comme candidat, est ensuite mis en oeuvre. Un score de vote est associé à

chaque couple M_{gd} et représente une évaluation de la validité de l'appariement.

Le processus de vote est mis en oeuvre en utilisant 3 *contraintes globales* :

- La *contrainte d'unicité* qui garantit qu'un gradient de l'image gauche correspond à un seul gradient de l'image droite et réciproquement.
- La *contrainte d'ordre* qui stipule que si un gradient g de l'image gauche a été associé à un gradient d de l'image droite, il est impossible qu'un gradient g' tel que $x_{g'}<x_g$, soit associé à un gradient d' tel que $x_{d'}>x_d$. Cette contrainte n'est pas absolue et peut être violée dans de nombreuses situations mais celles-ci restent rares en environnement routier.
- La *contrainte de continuité de la disparité* qui implique que des gradients voisins spatialement ont des disparités proches.

Les contraintes d'unicité et d'ordre sont utilisées pour déterminer les éléments votants pour chaque élément candidat et la contrainte de disparité intervient dans la mise à jour des scores de vote en accordant plus d'importance aux votes des éléments proches du candidat.

1.5 Approches multi-agents

1.5.1 Principe général

L'idée de base des approches multi-agents est de distribuer la résolution d'un problème à un ensemble d'entités autonomes, réactives, pro-actives et possédant une certaine "habilité sociale". Ces approches permettent de définir un contexte et des méthodes génériques pour la mise en oeuvre d'algorithmes distribués. [SimIA54], [HilIA54]
Il existe deux grands types d'agents :

- Les *agents cognitifs* qui ont un comportement individuel complexe et disposent de connaissances nécessaires à la réalisation de leur tâche et à la gestion de leurs interactions avec les autres agents et avec leur environnement.
- Les *agents réactifs* qui ont un comportement individuel simple et possèdent un mécanisme de réaction aux évènements, ne prenant en compte ni une explicitation des buts, ni des mécanismes de planification.

Un système multi-agents (*SMA*) est un système constitué d'agents qui interagissent dans et au travers d'un environnement. Deux niveaux sont donc à distinguer : un niveau micro pour l'agent et un niveau macro pour le système. Le comportement du niveau macro émergeant du comportement micro individuel de chaque agent et la construction d'un *SMA* doit donc tenir compte de ces deux niveaux.

1.5.2 L'éco-résolution

L'éco-résolution, qui est la technique sur laquelle nous nous sommes basés pour ce projet, prend le contre pied des techniques classiques de conception d'un système dédié à la résolution de problèmes. Avec cette technique, on ne formalise pas le problème de manière globale mais de manière à ce qu'il soit conçu comme un ensemble d'agents en interactions qui tentent de satisfaire individuellement leur propre but. Tous les éléments du problème deviennent alors agents en interaction. [SimIA54]

Le principe général de cette méthode est que chaque agent tente de réaliser son but et s'il perçoit un gêneur, il l'agresse. Un agent agressé tente de fuir en tenant compte des contraintes de son agresseur. Ce schéma se reproduit et fait évoluer le système jusqu'à aboutir à un état stable (satisfaction de tous les agents) qui correspond à la solution du problème.

Chapitre 2

Méthode proposée

2.1 Description générale

La méthode de reconstruction 3D que nous avons élaboré et mis en oeuvre se décomposé en 3 grandes étapes de traitements.

La première étape regroupe l'ensemble des pré-traitements graphiques à appliquer aux images d'entrées avant de pouvoir procéder à leur appariement. Il s'agit d'algorithmes de traitement d'images purs qui permettent d'extraire des images un ensemble de primitives (*features*) qui représentent les points caractéristiques de la scène et qu'il faudra appareiller pour en calculer la profondeur. Le résultat de cette première étape est un couple d'images 2D stockant l'ensemble des primitives. Nous avons en effet fait le choix d'unifier le format des données manipulées par nos algorithmes de traitement d'images et rendre ainsi nos implémentations les plus génériques possibles.

La deuxième étape est l'appariement multi-agents des primitives. Il s'agit du coeur de notre méthode de reconstruction 3D. Le résultat de cette étape est l'association d'un ensemble de primitives de l'image gauche avec leurs correspondants dans l'image droite.

La troisième étape est la reconstruction 3D en elle même avec le calcul de la profondeur des primitives et leur visualisation 3D. Cette étape est également très importante, et en particulier lors des phases d'expérimentation, car elle permet d'estimer visuellement la validité des résultats obtenus.

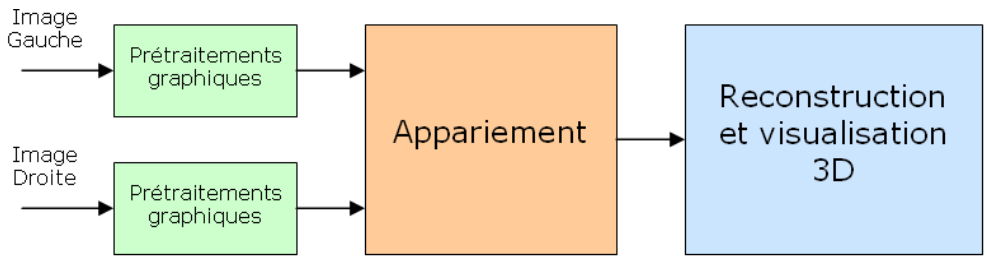


FIG. 2.1 – Étapes successives de la méthode de reconstruction 3D mise en oeuvre

2.2 Traitements graphiques

L'étape de traitements graphiques se décompose en 3 opérations appliquées sur les images stéréoscopiques. Ces opérations permettent d'obtenir 2 champs de gradients (cf. section 2.2.3) utilisables comme primitives pour l'appariement.

Le premier traitement est un filtrage gaussien appliqué directement sur l'image acquise (dans notre application il s'agit d'une image chargée à partir d'un fichier). Ce filtrage permet d'éliminer une partie du bruit parasite qui peut être présent sur l'image.

Une fois l'image filtrée, on procède à l'extraction de contours à l'intérieur de l'image, ces contours sont obtenus par calcul de gradients. Ce sont ces gradients qui seront par la suite utilisés, en combinaison avec 2 autres valeurs caractéristiques (cf. section 2.2.3), comme primitive pour l'appariement.

Calcul de gradients et filtrage gaussien sont effectués par convolution de l'image comme expliqué section 2.2.1.

La dernière opération mise en oeuvre est un seuillage appliqué sur les champs de gradients en fonction de la norme des vecteurs gradients afin d'éliminer les moins pertinents pour la mise en correspondance.

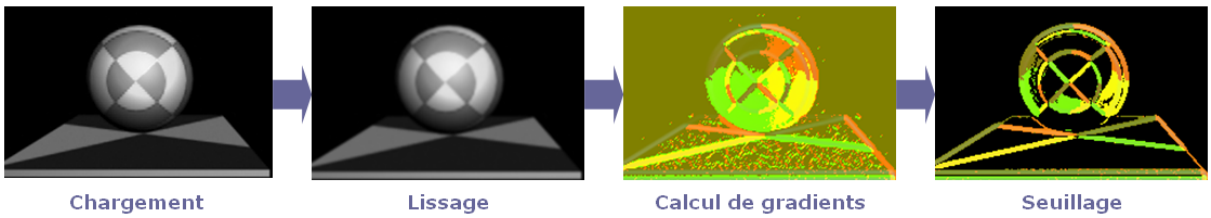


FIG. 2.2 – Étapes de traitements graphiques appliquées aux images stéréo

2.2.1 Filtres de convolution

Le filtrage consiste à appliquer une transformation (appelée *Noyau* ou *Kernel*) identique à l'ensemble des pixels d'une image en appliquant un opérateur. Un filtre est une transformation mathématique (appelée produit de convolution) permettant, pour chaque pixel de la zone à laquelle il s'applique, de modifier sa valeur en fonction des valeurs des pixels avoisinants, affectées de coefficients.

Le *Noyau* est représenté par une matrice, caractérisée par ses dimensions, ses coefficients (et éventuellement un facteur d'échelle), dont le centre correspond au pixel concerné. Les coefficients présents dans la matrice déterminent les propriétés du filtre. Le Noyau est appliqué à chaque pixel en sommant les valeurs des pixels de la zone couverte par la matrice pondérée par les coefficients. Le résultat peut ensuite éventuellement être divisé par un facteur d'échelle.

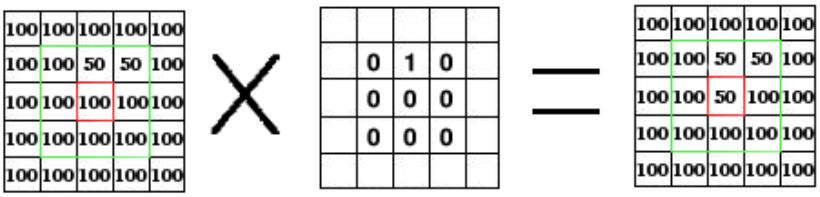


FIG. 2.3 – Exemple d'application d'un *Noyau* de convolution

2.2.2 Filtre gaussien

Le filtrage gaussien se base sur le principe que l'information contenue dans une image est redondante. Cette redondance est utilisée pour supprimer le bruit en lissant l'image. Pour cela, on définit un Noyau de convolution dont les pondérations suivent une courbe gaussienne 2D centrée sur le milieu de la matrice. Ce noyau est associé à un facteur d'échelle correspondant à la somme des coefficients afin de conserver globalement les intensités lumineuses.

Dans notre application, nous avons utilisé un *Noyau* 3x3 défini par la matrice suivante :

$$\frac{1}{16} \times \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 1 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

2.2.3 Filtre d'extraction de gradients

Le gradient représente la variation d'une image dans 1, 2 ou 3 directions, selon si l'image est 1D, 2D ou 3D. En termes mathématiques il s'agit de la dérivée de degré 1 d'une image estimée de manière discrète en chaque pixel et pour chaque direction. Le gradient est extrêmement utile pour détecter les contours présents dans une image et permet d'en connaître les orientations.

Définition mathématique

Le gradient est une fonction mathématique notée $\overrightarrow{\nabla}(P)$ de $\mathbb{R}^N$ dans $\mathbb{R}^N$, avec N la dimensionalité des signaux considérés. Dans le cas d'un signal 2D (comme c'est le cas de nos images) $P(X,Y)$ représente la valeur du signal en un point. Cette fonction fournit une grandeur vectorielle qui indique comment une grandeur physique varie en fonction de ses différents paramètres, elle est définie en 2D par la formule :

$$\overrightarrow{\nabla}(P) = \left(\frac{\partial P}{\partial x}, \frac{\partial P}{\partial y}\right)$$

Méthode d'estimation

Pour l'application des image 2D, une première estimation de $\overrightarrow{\nabla}(P)$ peut être faite de manière discrète par simple calcul de la variation de pas 2 sur les deux axes x et y :
Soit P définit par ses coordonnées x et y dans E, l'ensemble des points de l'image :

$$\overrightarrow{\nabla}(P(x,y)) = (P(x-1,y) - p(x+1,y), P(x,y-1) - P(x,y+1))$$

Il s'agit d'un filtrage de *Roberts* défini par les *noyaux de convolution* suivants :

$$\text{Axe X : } \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{Axe Y : } \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Une meilleur estimation du gradient peut être obtenue par utilisation d'un filtre de type *Prewitt* ou *Sobel* qui introduisent un opérateur de lissage et sont de ce fait moins sensibles au bruit que ceux de Roberts. L'image est convoluée avec les masques suivants (*Prewitt* c=1, *Sobel* c=2) :

$$\text{Axe X : } \begin{bmatrix} -1 & 0 & 1 \\ -c & 0 & c \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Axe Y : } \begin{bmatrix} -1 & -c & -1 \\ 0 & 0 & 0 \\ 1 & c & 1 \end{bmatrix}$$

Mise en oeuvre

Les filtrages de type *Roberts*, *Prewitt* et *Sobel* ont été testés dans notre application. Le filtrage *Sobel* est le plus intéressant en terme de qualité des gradients obtenus mais c'est également le plus coûteux en terme de temps de calcul. Le filtrage de *Roberts* est moins précis mais offre tout de même des résultats intéressants tout en optimisant les temps de calcul ce qui est un critère primordial dans l'optique du temps réel.

2.2.4 Primitives obtenues et encodage

Nous avons fait le choix de manipuler des données au format image, c'est à dire des matrices régulières à une ou deux (voir trois) dimensions composées de 1 à 4 composantes (de niveaux de gris à RGBA). De ce fait, le champ de gradient est encodé dans une image RGBA de la taille de l'image en niveaux de gris initiale. Comme illustré figure 2.4, les composantes R et G des pixels contiennent les composantes en X et Y du vecteur gradient. Le module d'extraction ajoute en plus dans la composante B un pré-calcul de la norme de ce vecteur ainsi que l'intensité d'origine du pixel dans la composante Alpha. Ces deux informations seront utiles lors de l'étape de mise en correspondance, l'informations d'intensité permettant d'ajouter un élément de discrimination des primitives.

FIG. 2.4 – Illustration de l'encodage des primitives dans un pixel RGBA

2.2.5 Accélération matérielle

Depuis environ 6 ans, poussé par l'industrie du jeu vidéo, le matériel graphique grand public (appelé *GPU*, Graphical Processing Unit) présent sur les PC standards connait une évolution extrêmement importante, bien plus importante que celle des processeurs centraux (CPU). Les GPU offrent maintenant des unités programmables de traitement qui permettent aux programmeurs de paramétrer de manière extrêmement flexible (via des extensions OpenGL, [OGL01]) les opérations d'affichage fournies par la carte.

L'unité programmable de traitement de fragments, qui est l'unité en charge de l'application de textures par la carte est spécialisée dans le calcul vectoriel flottant. Cette unité est maintenant totalement programmable et offre sur les derniers modèles de GPU tel que le NV40 de NVidia, une puissance de calcul de l'ordre de 40Gflops ce qui est plus du double de la puissance qu'est capable de fournir le dernier Pentium 4. Le problème est que cette unité reste tout de même extrêmement spécialisée mais il est tout de même possible, et c'est un sujet de recherche appelé GPGPU (General Purpose Graphical Programming Unit) extrêmement actif en ce moment, de l'utiliser pour des traitements autres que des opérations d'affichage.

L'idée est donc de profiter de cette énorme puissance de calcul vectorielle pour réaliser des opérations de traitement d'images. Les GPU sont conçus selon une architecture SIMD très efficace lorsqu'il s'agit d'appliquer un traitement identique et indépendant sur un vaste ensemble de données. Cela correspond exactement au besoin d'un filtrage par convolution et ce sont ces traitements que nous avons mis en oeuvre sur le GPU.

La principale limite de ce type d'architecture est le bus d'accès AGP (le problème est moins présent avec le PCI-Express) qui offre une bande passante de la carte au processeur central extrêmement limitée. Notre implémentation nous a tout de même permis de diviser par 2 le temps nécessaire aux opérations de pré-traitement graphiques (sur des images de taille importante). Il semble qu'il s'agisse donc d'une piste importante à explorer pour la mise en ouvre temps réel de méthodes de reconstruction 3D.

2.3 Appariement multi-agents

2.3.1 Présentation

L'idée à la base de notre projet d'appariement multi-agents est de développer un modèle basé sur un ensemble d'agents réactifs collaborant afin de faire émerger de leur comportement individuel simple, la meilleur configuration d'appariement des primitives présentes dans un couple d'images stéréoscopiques. L'utilisation d'agents réactifs permet d'assurer un coût d'exécution individuel des agents assez faible ce qui est intéressant pour tendre vers des traitements temps réel.

La conception de cette méthode a donc été effectuée via une approche orientée agent, où le problème est abordé sur 2 plans : le comportement individuel des agents et leur comportement collectif qui doit permettre de faire émerger la solution.

Deux approches multi-agent ont été développées pour résoudre ce problème : la méthode ASI (Agents Sur Image) et la méthode ASM (Agents Sur Matrice), mais elles se basent toutes les deux sur le même principe.

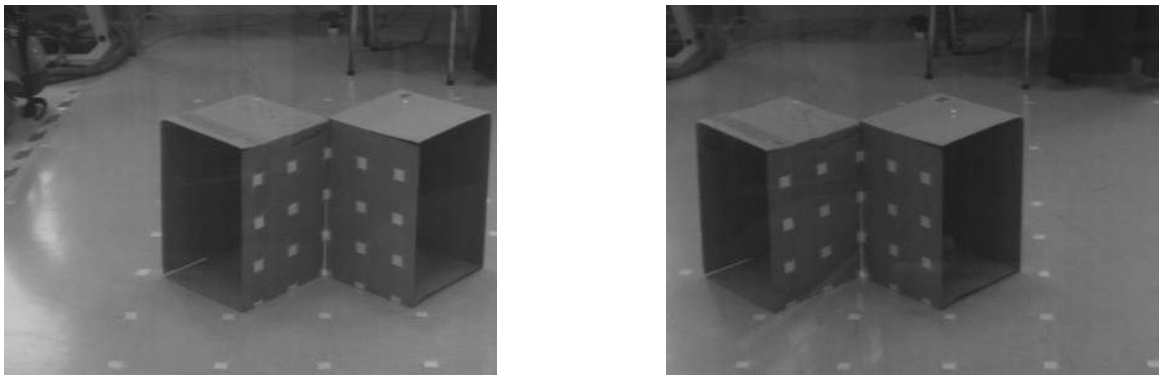


FIG. 2.5 – Exemple de couple d'images stéréoscopiques monochromes traitées par notre méthode

2.3.2 Agentification du problème

Le principe à la base de nos deux méthodes est d'associer un agent à chaque primitive non nulle extraite de l'image gauche puis de faire rechercher à chaque agent la primitive de l'image droite lui correspondant le mieux en interagissant avec les autres agents. Elles se basent sur des agents spatialement localisés dans l'environnement dans l'optique du principe décrit dans l'article [GeSi05].

La recherche d'appariements a été développée et mise en oeuvre pour des images matricielles 2D à partir d'une *contrainte épipolaire* qui assure l'alignement et le parallélisme des droites épipolaires des capteurs stéréoscopiques et le parallélisme de leurs axes optiques. Cette contrainte permet de se placer dans une configuration idéale et de limiter ainsi l'espace de recherche de correspondants aux seules primitives situées sur la même ligne de l'image 2D acquise.

Ceci permet également d'utiliser les contraintes *géométrique*, *d'unicité*, *d'ordre* et de *continuité de la disparité* présentées sections 1.4.2 et 1.4.2.

La contrainte de signe a elle été adaptée à des gradients 2D et devient une *contrainte d'orientation* des vecteurs.

2.3.3 Modèle ASI : Agents Sur Image

Le modèle ASI que nous avons élaboré se base sur l'utilisation directe des images contenant les primitives à appareiller comme environnement pour agents réactifs spatialement localisés.

Environnement initial

L'environnement utilisé est un environnement discret représenté par une matrice bidimensionnelle de la taille des images à appareiller. Il est initialisé avec l'image droite du couple stéréoscopique et c'est sur celle-ci que les agents vont se déplacer. Chaque case de cet environnement se voit donc associée une primitive, éventuellement nulle, correspondant à un pixel encodé sur l'image droite, ainsi qu'un emplacement pour un (et un seul, cf. *contrainte d'unicité*) agent localisé.

Une fois associés aux primitives de l'image gauche, les agents sont placés dans l'environnement à l'emplacement occupé dans l'image gauche par leur primitive. Les agents sont ainsi associés à une deuxième primitive correspondant à leur emplacement et définissent un appariement initial (état initial du système).

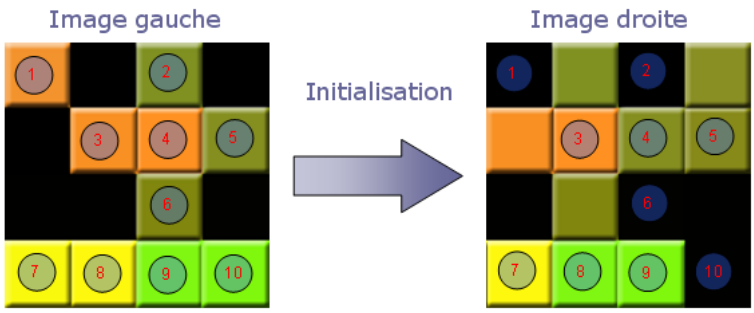


FIG. 2.6 – Exemple d'initialisation de 10 agents sur un champ de gradients de 4x4

Principe de base

Les agents localisés sur l'image droite cherchent à maximiser une fonction de satisfaction qui évalue la validité de leur appariement courant (cf. section 2.3.3), en se déplaçant de droite à gauche sur la ligne sur laquelle ils ont été initialisés.

Le sens de déplacement vers la gauche est fixé par la *contrainte géométrique* qui stipule que le correspondant d'une primitive située sur l'image gauche se trouve obligatoirement à une coordonnée plus petite, donc à gauche, sur l'image de droite.

Si la case à gauche d'un agent lui offre une meilleure satisfaction ($testSat$), celui-ci cherche à s'y déplacer. Si la case contient déjà un autre agent, c'est à dire que la primitive est déjà appareillée avec une primitive de l'image gauche, la *contrainte d'unicité* interdit à l'agent de s'y rendre directement.

Un rapport de force s'engage alors, d'une manière inspirée du principe d'agression de l'*éco-résolution* et du calcul de satisfaction de l'*éco-résolution* ([SimIA54]). L'agent voulant se rendre sur une case va attaquer l'agent qui y est déjà présent en lui communiquant le gain de satisfaction qu'il obtiendrait s'il se trouvait sur la case convoitée. L'agent sur la case passe alors dans l'état "Attaqué".

L'agent attaqué va alors chercher à fuir sur la case à sa gauche en fonction de sa satisfaction courante ($curSat$) et du gain ($oppGain$) qu'il apporterait à l'agent qui l'attaque (forme d'altruisme). Un coefficient d'altruisme $altCoef$ est introduit et si $(testSat + altCoel * oppGain) > curSat)$, l'agent cherche à se déplacer sur cette case.

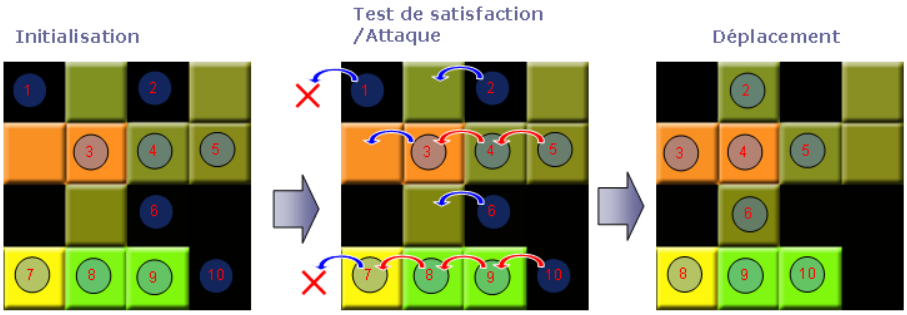


FIG. 2.7 – Illustration du principe d'attaque et de déplacement des agents localisés sur une image droite

Minimums locaux

Le problème de l'algorithme en l'état est que les agents risquent tout de même de tomber dans des minimums locaux de leur fonction de satisfaction et bloquer ainsi la résolution avant d'arriver à un résultat correct. Pour contrer ce problème, nous avons modifié l'algorithme pour créer un effet de "pression de masse" des agents entre eux. Cette pression est mise en oeuvre lors de la transmission du gain $|curSat - testSat|$ d'un agent attaqué à l'agent dont il veut la place. A ce gain vient s'ajouter le gain interne $oppGain$ déjà transmis lors de l'attaque de l'agent attaquant. De ce fait, lors de la formation de chaînes d'attaques, plus le gain global est important et plus la pression sur l'agent bloquant est importante ce qui permet de le repousser.

Agents non appareillables

Un certain nombre de primitives de chaque image n'ont pas de correspondant dans l'image opposée. Cela est du à des parties à gauche et à droite des images invisibles sur l'image associée, à des éléments masqués par d'autres du fait de l'angle de vue, à des imprécisions d'acquisition ou de traitement des images...

De ce fait, un certain nombre d'agents doivent être éliminés au cours du processus d'appariement, du fait également des imperfections de la méthode qui peuvent conduire des agents à ne pas trouver de correspondant alors qu'il existe.

Nous avons mis en évidence deux situations typiques dans lesquelles les agents doivent être éliminés :

- Lorsqu'un agent arrive en bout de ligne, à gauche, sans avoir trouvé d'appariement. L'agent a soit raté son appariement, soit il était situé dans une zone à gauche de l'image non visible dans l'image droite.
- Lorsqu'un agent n'a pas d'appariement car sa caractéristique n'existe pas dans l'image droite (elle est masquée ou n'a pas été acquise pour une raison quelconque). Dans ce cas, l'agent

a de grandes chances de se retrouver dans une situation où il est en "sandwich" entre un agent fortement satisfait à sa gauche et un autre qui l'attaque à sa droite.

La situation d'un agent en bout de ligne est facilement détectable en testant la coordonnée en x de l'agent qu'il suffit alors de détruire (ou faire *popper*).

La deuxième situation est plus délicate à traiter. Elle doit être contrôlée avant une tentative d'attaque par un agent déjà attaqué. Deux approches ont été testées pour traiter ce cas de figure, une approche de satisfaction totalement relative par rapport aux autres agents et une approche basée sur un seuil de satisfaction. Pour le test par satisfaction relative, l'agent est détruit s'il est attaqué et si $testSat < O1.curSatETcurSat < (O2.curSat + oppGain)$, avec $O1$ l'agent sur la case de gauche et $O2$ l'agent sur la case de droite. Pour le test par seuil, l'agent est détruit si il est attaqué et si $O1.curSat > SEUILSAT$. Lors de nos essais c'est la méthode par seuil qui s'est avérée la plus efficace.

Calcul de satisfaction

Le calcul de satisfaction est un élément clef de la résolution car il permet d'évaluer la validité d'un appariement et conditionne donc l'évolution des agents. Nous l'avons définit comme une valeur entre 0 et 1. Un très grand nombre de fonctions ont été testées pour ce calcul et nous en avons retenu 2 qui donnent des résultats intéressants.

La première fonction de satisfaction se base sur le *critère d'orientation* du gradient et sur sa norme (P_l position sur l'image gauche, P_r position sur l'image droite) :

$$\tau(P_l, P_r) = \frac{1}{1 + \Delta O(P_l, P_r) + \Delta A(P_l, P_r)}$$

Avec ΔO la différence d'orientation des gradients en P_l et P_r :

$$\Delta O(P_l, P_r) = \left| atan\left(\frac{\nabla_l^y(P_l)}{\nabla_l^x(P_l)}\right) - atan\left(\frac{\nabla_r^y(P_r)}{\nabla_r^x(P_r)}\right)\right|,$$

avec $\nabla_b^a(X)$ la composante en a (entre 0 et 1) du gradient de l'image b au point X.

ΔA la différence d'amplitude des gradients en P_l et P_r :

$$\Delta A(P_l, P_r) = |\ \|\overrightarrow{\nabla}_l(P_l)\| - \|\overrightarrow{\nabla}_r(P_r)\|\ |$$

La deuxième fonction testée prend également en compte l'intensité lumineuse initiale du pixel qui a généré la primitive. Comme expliqué section 2.2.4, cette intensité est encodée dans le vecteur stockant la primitive (dans l'ordre gradient X et Y, norme et intensité). Le calcul de satisfaction se contente donc d'utiliser directement ce vecteur de primitive :

$$\tau(P_l, P_r) = \frac{1}{1 + distance(\overrightarrow{V}_l(P_l), \overrightarrow{V}(P_r))},$$

avec $\overrightarrow{V}_l(P_l)$ et $\overrightarrow{V}_r(P_r)$ les vecteurs encodant les primitives dans l'image gauche et droite aux positions P_l et P_r et $distance(\overrightarrow{A}, \overrightarrow{B})$ la distance euclidienne entre les deux vecteurs A et B.

Ce calcul donne une plus forte importance à la norme du vecteur gradient qui est prise en compte 2 fois mais donne de meilleurs résultats. C'est celui ci qui a été adopté pour nos deux méthodes. Par souci d'optimisations la fonction $distance$ a été remplacée par $distance^2$.

Un test supplémentaire peut être ajouté à ces deux calculs afin de fixer la satisfaction à 0 dans le cas d'une différence d'orientation supérieure à un seuil admissible (*contrainte d'orientation*).

Nous avons fait des essais d'utilisation d'un critère de *différence de disparité* par rapport aux agents présents localement dans l'environnement de l'agent, ce critère permettant d'estimer la satisfaction d'un agent en fonction de sa différence de disparité avec ses agents voisins. Cet essai ne s'est pas avéré concluant et nous nous sommes aperçu que ce critère amenait plutôt les agents à se regrouper dans des minimums locaux. Ce critère induit en plus un coût d'estimation très important puisqu'il faut effectuer une recherche dans un voisinage plus ou moins grand. Nous pensons par contre que ce critère pourrait être améliorer en pondérant la prise en compte des disparités voisines en fonction de la satisfaction des agents testés ce qui permettrait peut-être d'éviter la chute vers des minimums locaux.

Comportement complet

Initialisation() : calcul de satisfaction sur la case courante -> `curSat`.

Live() :
- Test de la case à gauche de l'agent *(contrainte d'ordre)*, calcul de satisfaction -> `testSat`.
- Si (`testSat + altCoef*oppGain`) >= `curSat` :
  - Si un agent O est présent sur la case,
    - Si `EstAttaqué` ET `O.curSat > SEUIL-SAT`, Pop() ;
    - Sinon Attaquer(O, abs(`curSat-testSat`)+`oppGain`) ; *(contrainte d'unicité)*.
  - Sinon aller sur case. `EstAttaqué`=Faux, `oppGain`=0 ;
- Sinon,
  - Si (`EstAttaqué` ET `curSat <= SEUIL-SAT-MORT`), Pop() ;

Attaquer(Agent, Gain) :
- `EstAttaqué`=Vrai, oppGain=Gain ;

2.3.4 Modèle ASM : Agents Sur Matrice

Le principe de cette méthode est très similaire à celui de la méthode ASI, il s'agit principalement d'une autre façon de voir l'algorithme et surtout de l'implémenter.

Description de l'environnement

Cette méthode est mise en oeuvre à partir de la matrice de correspondance telle qu'elle est décrite dans l'article [HRK03] , c'est à dire la matrice de couples $M_{l,r}$ associant les primitives de l'image gauche avec celles de l'image droite. Cette matrice est utilisée comme environnement pour les agents localisés.

Contrairement à la méthode décrite dans [HRK03], notre méthode a été élaborée pour traiter des images matricielles 2D. Il en résulte la nécessitée d'associer non plus une ligne de primitives qui est l'image gauche avec une ligne qui est l'image droite mais un ensemble de lignes de l'image gauche avec un ensemble de lignes de l'image droite. Heureusement, la *contrainte épipolaire* (cf. 2.3) que nous avons imposé sur le capteur stéréo permet de réduire l'espace de recherche en rendant totalement indépendantes l'ensemble des lignes des images. La matrice de mise en correspondance devient alors une matrice 3D dont chaque plan représente la mise en correspondance d'une ligne de l'image gauche et droite comme illustré figure A.1.

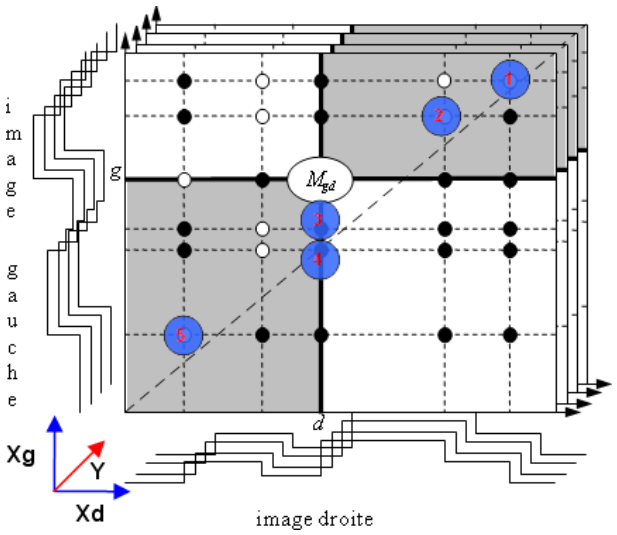


FIG. 2.8 – Illustration de la matrice 3D utilisée comme environnement

Cette matrice représente donc l'environnement des agents qui deviennent alors des agents localisés dans l'espace.

L'avantage d'utiliser cette matrice est qu'elle permet d'effectuer un certain nombre d'optimisations en amont de l'appariement en éliminant les couples ne répondant pas aux contraintes *géométrique* et *d'orientation*. Nous avons également utilisé cette matrice pour permettre aux agents d'inscrire le taux de satisfaction qu'ils calculent pour un appariement afin d'éviter aux autres agents passant sur cet appariement de le recalculer.

Le problème de cette matrice est qu'elle nécessite un espace de stockage extrêmement important. En effet, pour une image de 400 pixels de côté, la matrice stockant 16 octets par case (2x 4 octets pour les primitives et encore 2x 4 octets pour leurs positions), l'espace mémoire maximum nécessaire est de l'ordre de 900Mo (400*400*400*16 octets) ! Heureusement l'ensemble des pixels

d'une image ne contient généralement pas de primitives et la matrice atteint généralement une taille moins importante.

Il est à noter que l'ensemble des lignes d'une image ne contient pas le même nombre de primitives extraites, de ce fait, la matrice n'est pas naturellement régulière et certains plans sont plus grands que d'autres. Il faut donc considérer indépendamment l'ensemble des plans ou alors normaliser leur taille à celle du plus grand plan. Nous avons fait le choix pour notre implémentation de considérer indépendamment les plans.

Agentification du problème

Pour résoudre le problème, nos agents sont chargés de choisir les meilleurs couples $M_{l,r}$ en évaluant leur degré de validité. Dans cette vision des choses les agents peuvent être vus comme des sortes d'explorateurs allant de couple en couple à la recherche du meilleur de sa ligne.

L'initialisation se fait, pour chaque plan de la matrice, en créant un agent par ligne que l'on positionne sur le premier appariement valide à droite de la ligne (cf. figure A.1).

Le principe de l'algorithme est ensuite similaire à celui de la méthode ASI, les agents cherchent à maximiser leur satisfaction en se déplaçant de droite à gauche sur leur ligne. Les attaques se font cette fois sur la colonne entière, la contrainte d'unicité n'étant plus implicite au modèle adopté.

2.4 Reconstruction 3D

Une fois les primitives appareillées, la reconstruction 3D consiste en le calcul de la profondeur et de la position de chacune dans l'espace de la scène visualisée via un calcul de triangulation.

$$z_P = \frac{E.f}{d}$$

$$x_p = \frac{x_l.z_p}{f} - \frac{E}{2} = \frac{x_r.z_p}{f} + \frac{E}{2}$$

avec E la distance inter-oculaire, f la longueur focale des capteurs et d la disparité du point calculée $d = |x_l - x_r|$.

Chapitre 3

Architecture de l'application

Notre objectif a été tout au long de ce projet de fournir un développement suffisamment générique pour être réutilisé dans d'autres projets de traitement d'images multi-agents.

A partir des 3 étapes décrites section 2.1, l'application a été architecturée comme un pipeline de visualisation, c'est à dire une série de modules interconnectés et intervenant les uns à la suite des autres sur une ou plusieurs images afin de produire le résultat attendu (ici une évaluation de la profondeur des zones caractéristiques de la scène).

Nous avons pour cela développé un mini framework objet de traitement d'images (1D, 2D ou 3D) extensible et écrit en C++ pour permettre une optimisation maximale des algorithmes en vue d'une utilisation temps-réel.

Ce framework introduit une notion de filtres graphiques paramétrables (cf. annexe A). Il s'agit de modules indépendants constitués d'un ensemble d'entrées et de sorties et qui communiquent via un type de données *Image*. Ce type *Image* permet le stockage et la communication inter-filtre dans un format optimisé (cf. section 2.2.5). Nous avons mis en oeuvre les algorithmes nécessaires à la résolution de notre problème au sein de cette architecture.

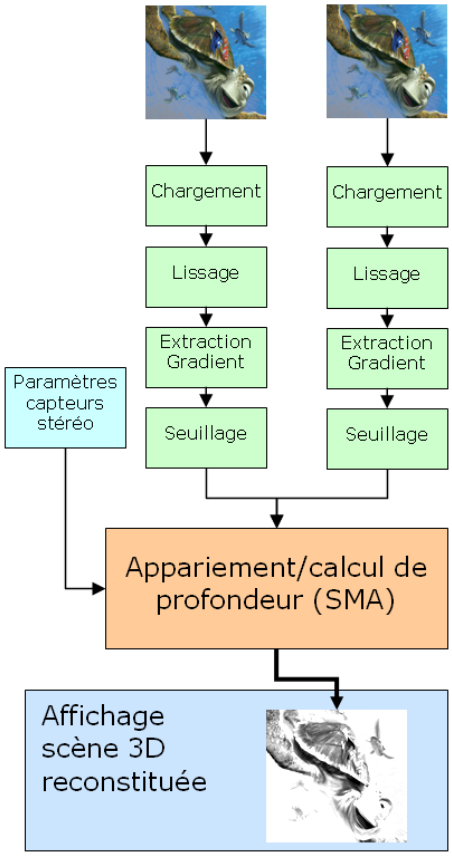


FIG. 3.1 – Schéma général du pipeline d'appariement

Grâce à la *contrainte épipolaire* sur la géométrie du capteur stéréo (cf. 2.3), nous avons pu mener une approche générique qui offre à nos développements la possibilité de traiter aussi bien des images linéaires que matricielles, le traitement des images linéaires devenant un cas particulier du traitement des images matricielles.

3.1 Modules de traitement

Des modules de traitement ont été développés pour l'ensemble des étapes du pipeline de visualisation. La première étape consiste à charger les images gauche et droite à partir d'un fichier et à les introduire dans le pipeline dans un format qu'il sait manipuler. Pour cela, un module de chargement de fichiers image au format *Jpeg* a été mis en oeuvre et intégré dans le framework de traitement graphique.

Une interface de module de filtrage générique a été conçue pour la mise en oeuvre de filtres (cf. section 2.2). Un module de convolution générique a été implémenté sur cette base permettant d'effectuer les étapes de filtrage de l'algorithme, ainsi qu'un module d'extraction de gradients. Un module de filtrage GPU a également été implémenté (cf. 2.2.5) ainsi qu'un type de données image adapté à son utilisation.

Les modules d'appariement correspondants aux méthodes exposées section 2.3 ont également été mis en oeuvre dans cette architecture en tant que filtres image.

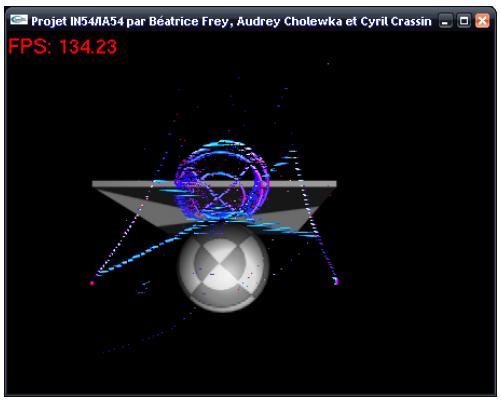
3.2 Plateforme multi-agents

Pour l'implémentation de notre méthode multi-agents, nous avons adjoint à notre framework de traitement d'images une mini-plateforme multi-agents pour agents spatialement localisés. Un noyau générique a été mis en oeuvre et spécialisé pour la simulation et l'ordonnancement d'agents par une méthode s'inspirant de l'éco-résolution.

L'architecture de communication de nos agents a été élaborée pour offrir une efficacité maximale via l'utilisation d'états internes et de valeurs modifiables directement entre agents (flags). Cela en évitant que les mécanismes d'attaque mis en oeuvre deviennent procéduraux ce qui assure l'indépendance totale et l'équilibrage de l'exécution des agents.

3.3 Architecture de visualisation

L'architecture de visualisation et d'interaction avec l'application a été conçue en utilisant la librairie d'affichage *OpenGL* et le système de gestion de fenêtrage *GLUT*. La gestion du fenêtrage et l'interaction a été abstraite au travers d'une classe *GLApplication* qui offre une série de services et dont il suffit de faire hériter sa propre classe application (*StereoMatchingApp*).



Chapitre 4

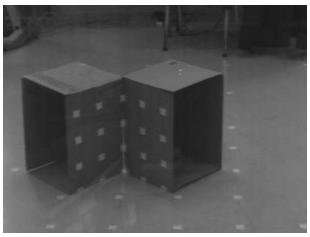
Résultats

4.1 Résultats temporels

Comme montré section 4.2, les deux méthodes d'appariement multi-agents développées possèdent chacune des avantages et des inconvénients. Nous avons tenté d'évaluer leurs performances respectives en effectuant une série de mesures de temps d'exécution sur les étapes d'initialisation (initialisation de l'environnement et création des agents) et d'exécution (mesure du temps nécessaire à l'exécution d'une itération sur l'ensemble des agents). Ces tests ont été réalisés sur un Athlon64 3400+ équipé de 2Go de RAM ainsi que d'une carte graphique GeForce6800GT.

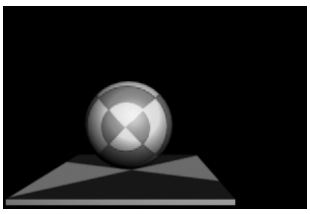
Temps moyens mesurés sur une scène réelle, images de résolution 400x300 pixels, 7880 Agents :

Méthode	Initialisation	Itération	Temps stabilisation
ASI	68ms	6.5ms	3000ms
ASM	250ms	4.1ms	400ms



Temps moyens mesurés sur une scène artificielle (générée avec 3DsMAX), images de résolution 300x200 pixels, 1553 Agents :

Méthode	Initialisation	Itération	Temps stabilisation
ASI	22ms	2.3ms	400ms
ASM	47ms	1.7ms	180ms



Ces résultats montrent que la méthode ASM possède l'avantage de converger beaucoup plus rapidement vers un état stable. Malheureusement, le résultat obtenu s'avère dans les deux tests et avec l'implémentation actuelle beaucoup moins bons qu'avec la méthode ASI (cf. figure 4.1). Nous estimons que cela est du au fait que l'effet de "forces de pression" généré par les chaînes d'attaques entre agent dans la méthode ASI est beaucoup moins présent dans la méthode ASM du fait de l'absence de positions intermédiaires entre appariements. Avec l'utilisation directe des

images issues de l'étape de pré-traitement graphiques, la méthode ASI a en plus l'avantage d'avoir un temps d'initialisation beaucoup moins important que la méthode ASM qui doit elle créer sa matrice d'appariement. Les optimisations mises en oeuvre dans la méthode ASM sous la forme de pré-calculs et d'enregistrement des taux de satisfaction dans la matrice permettent d'obtenir un temps moyen d'itération meilleur qu'avec la méthode ASI.

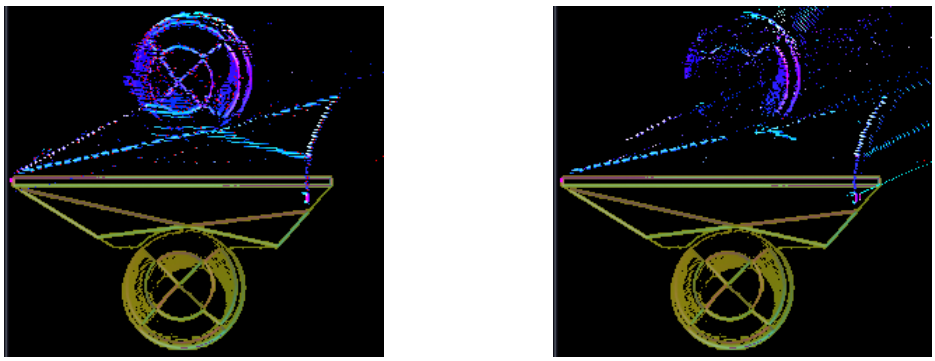


FIG. 4.1 – Comparaison des résultats obtenus sur la méthode ASI (à gauche) et ASM (à droite)

4.2 Comparaison des méthodes

La méthode ASM offre l'avantage de permettre un certain nombre d'optimisations lors de l'étape d'initialisation de l'algorithme. Elle permet également le stockage de résultats de calculs de satisfaction permettant leur réutilisabilité par d'autres agents. Enfin elle fournit des résultats temporels meilleurs que la méthode ASI dans l'implémentation que nous avons effectuée. Elle a par contre un coût en espace mémoire bien plus important que la méthode ASI.

Mais l'avantage des pré-calculs permis par la méthode ASM trouve ses limites lors du passage au traitement d'un flux continu d'images. Sur ce type de problème il n'est pas intéressant d'effectuer des calculs préalablement à l'exécutions des agents car c'est plutôt la capacité d'adaptation et d'évolution de l'algorithme qui est déterminant. La méthode ASI possède l'avantage d'offrir une structure de résolution contenant intrinsèquement de part la structure de son environnement et de manière implicite les contraintes *géométrique*, *d'unicité* et *d'ordre* ce qui démontre l'intérêt des méthodes à agents localisés basées sur l'environnement pour la résolution intuitive de ce type de problème. Elle a également l'avantage d'utiliser directement les images sans création d'une nouvelle structure de donnée ce qui permet d'économiser du temps sur l'étape d'initialisation de l'algorithme. De plus, l'accès aux agents situés dans le voisinage géographique (sur l'image) d'un agent est très simple dans cette approche ce qui n'est pas le cas dans la méthode ASM. Cette méthode semble ainsi la plus flexible et la plus apte à être améliorée vers des techniques adaptatives capables de fournir des résultats temporels pourquoi pas compatibles avec l'utilisation pour le traitement temps réel de flux d'images matricielles.

Bilan et perspectives

Avec ce projet, nous avons conçu, implémenté et calibré deux méthodes d'appariement stéréoscopique novatrices et montré l'utilisabilité d'approches à base d'agents réactifs localisés pour aborder des problèmes de traitement d'images. Nous avons également fourni une base de framework ouvert pour le traitement d'images par méthodes multi-agents qui constitue une plateforme de test et de prototypage pour ce type d'algorithme. Cette plateforme nous a permis de tester et de faire évoluer nos approches et constitue une base intéressante pour des développements futurs. Nous avons également montré l'intérêt du matériel graphique programmable pour l'application de filtres de convolution et en particulier pour des applications nécessitant un traitement temps réel des données.

Ce projet couplé entre une UV de traitement d'images et une UV sur les systèmes multi-agents s'est avéré extrêmement intéressant, aussi bien sur le plan technique qu'humain, en nous permettant de nous investir fortement sur un sujet de recherche novateur et qui reste encore largement ouvert.

Ce travail a été une première approche d'une méthode offrant encore énormément de perspectives d'améliorations et de pistes d'exploration. La suite logique de ce projet serait, dans un premier temps, de chercher à exploiter la capacité d'adaptation des agents, en partant sans doute de la méthode ASI, afin de permettre peut être l'utilisation de l'approche pour le traitement temps réel d'un flux d'image. Une deuxième évolution serait l'extension de l'approche ASI au traitement d'images issues d'un capteur stéréo non idéal (recherche multi-directionnelle). Enfin pourquoi ne pas tenter l'adaptation de la plateforme multi-agents à un environnement distribué de type *cluster*, voir un *cluster* de machines équipées de GPU, dédié aux traitements graphiques multi-agents.

Bibliographie

[HRK03] M. Hariti, Y. Ruichek, A. Koukam, 2003.
A voting stereo matching method for real-time obstacle detection.
In IEEE International Conference on Robotics and Automation. Taipei, Taiwan.

[GeSi05] F. Gechter, O. Simonin, 2005.
Conception de SMA réactifs pour la résolution de problèmes : Une approche basée sur l'environnement.
Soumission JFSMA'05. Volume 8/2005, pages 1 à 13.

[Gec99] Franck Gechter. LORIA, université Louis Pasteur
Etalonnage de Caméras et Vision stéréoscopique.
DEA Photonique et Image option Traitement d'Images, rapport de stage.

[Rab00] Christophe RABAUD. Laboratoire d'Informatique de Robotique et de Micro-électronique de Montpellier.
La mise en correspondance d'images stéréoscopiques.
`http://www.lirmm.fr/doctiss04/art/S03.pdf`.

[Der90] R. Deriche. 1990
"Fast algorithms for low-level vision".
IEEE Trans. on Patt. Anal. Mach. Intel., vol.12, no.1, pp.78-87.

[SimIA54] Olivier Simonin, Université de Technologie de Belfort Montbéliard.
Cours IA54 : Résolution Collective de Problèmes et Architectures Réactives.

[HilIA54] Vincent Hilaire, Université de Technologie de Belfort Montbéliard.
Cours IA54 : Intelligence Artificielle Distribuée et Systèmes Multi-Agents.

[OGL01] Site officiel d'OpenGL. *http://www.opengl.org*.

[NV01] NVIDIA Developer Web Site. *http://developer.nvidia.com*.

Annexe A

Diagramme de classes

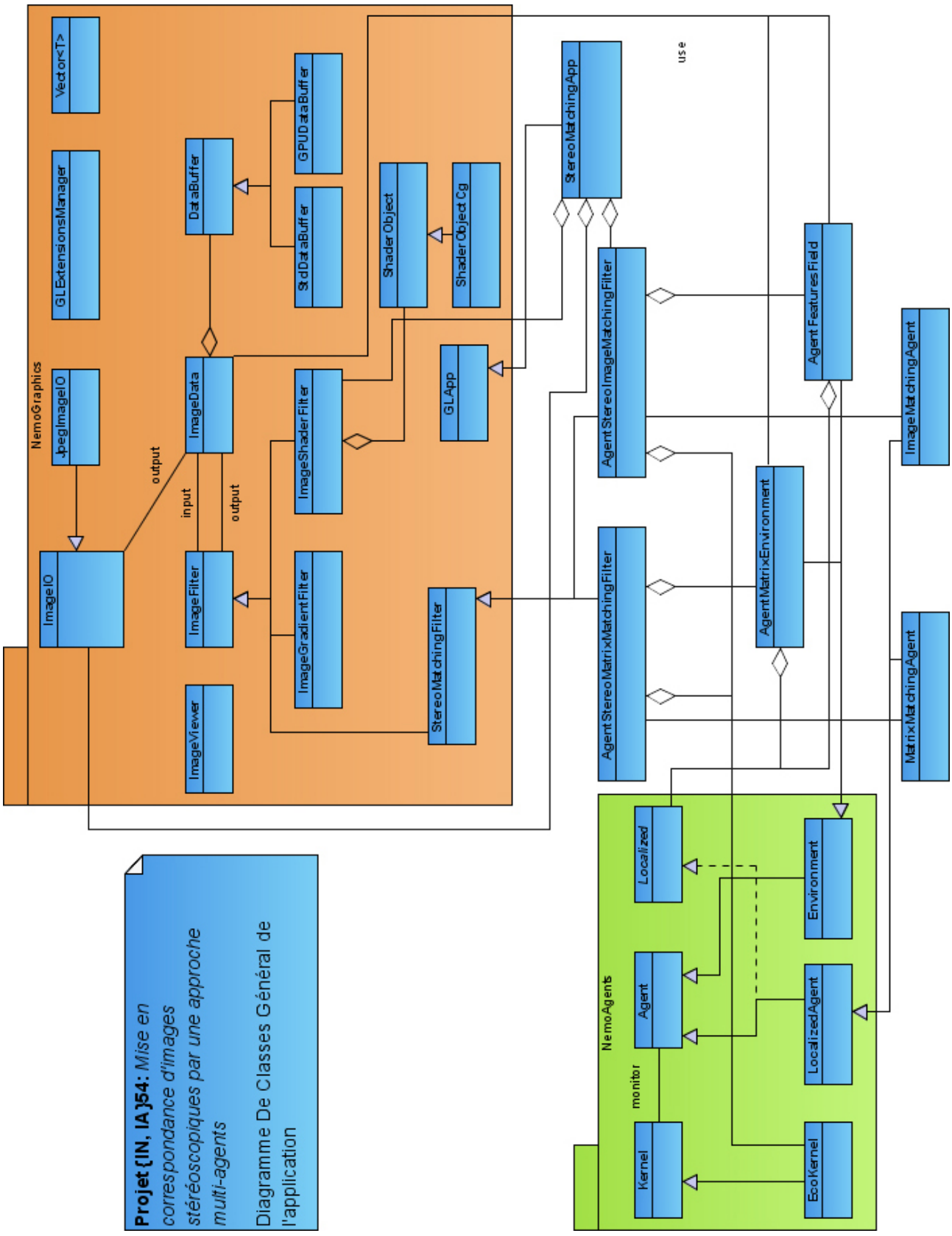


FIG. A.1 – Diagramme de classe complet de l'application et du framework *Nemo*