Contributions pour les applications de réalité augmentée

Christophe Dehais

N° d'ordre :

Institut de Recherche en Informatique de Toulouse — UMR CNRS 5505

Contributions pour les applications de réalité augmentée.

—

Suivi visuel et recalage 2D.
Suivi d'objets 3D représentés par des modèles par points.

MÉMOIRE

présenté et soutenu publiquement le 21 mai 2008

pour l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE
(Spécialité Informatique)

par

Christophe DEHAIS

devant le jury composé de

Président :	Jean-Pierre Jessel (Professeur Université Toulouse III)
Rapporteurs :	Jean-Marc Lavest (Professeur Université Clermont I) Pascal Guitton (Professeur Université Bordeaux I)
Examinateurs :	Vincent Lepetit (CR École Polytechnique Fédérale de Lausanne) Alain Ayache (Professeur INP Toulouse, Directeur de thèse) Vincent Charvillat (MdC HDR INP Toulouse) Géraldine Morin (MdC INP Toulouse) Ariel Choukroun (Ingénieur R&D Société FittingBox)

Résumé : Cette thèse présente plusieurs méthodes de recalage pour les applications de réalité augmentée (R.A.). Nous décrivons d'abord des expériences de R.A. utilisant des recalages et suivis uniquement 2D. Nous nous intéressons ensuite au suivi visuel d'un objet naturel dont on connaît un modèle 3D et dont l'image peut ainsi être augmentée avec une cohérence spatiale et temporelle.

Dans une première partie, nous proposons d'abord d'utiliser un recalage homographique en temps-réel pour augmenter des séquences filmées par une caméra en rotation autour de son centre optique. Dans une autre application, des transformations non rigides sont calculées hors ligne pour augmenter les images naturelles des parois gravées d'une grotte préhistorique. Le recalage géométrique des interprétations graphiques d'un préhistorien permet de créer un logiciel de découverte interactive des parois.

Dans la seconde et majeure partie de ce travail, nous partons des méthodes de suivi 3D de l'état de l'art prises parmi les plus performantes. Ces méthodes consistent à suivre un objet naturel connaissant sa représentation par un maillage 3D. Nous proposons une approche de suivi visuel 3D utilisant quant à elle des modèles par points de l'objet. Ce type de modèle, caractérisé par l'absence de topologie, est encore peu utilisé en vision par ordinateur mais il présente une souplesse intéressante par rapport aux modèles constitués de facettes.

La méthode de suivi que nous proposons consiste à interpréter des mises en correspondances 2D entre points d'intérêt en termes de variations de positions 3D. Le processus d'estimation sous-jacent utilise des champs de mouvements déduits des modèles 3D par points et des reconstructions par «Moving Least Squares» et «splatting». Ces techniques développées par la communauté d'informatique graphique s'attachent à reconstruire localement (explicitement ou implicitement) la surface de l'objet à suivre et certains attributs définis de manière éparse sur le nuage de points. Nous les adaptons à l'interpolation des champs de mouvements. L'avantage de notre approche est d'aboutir à un algorithme enchaînant quelques étapes d'estimation linéaires pour la détermination du mouvement 3D inter-images. Notre technique de résolution est intégrée à une adaptation originale d'un algorithme de suivi visuel de l'état de l'art qui repose sur un suivi hybride, combinant les informations issues de l'image précédente et celles apportées par des images clés acquises hors ligne. Une des particularités de notre implantation vient aussi de l'exploitation des capacités des unités de calcul graphiques (GPU) modernes pour les parties critiques de l'algorithme (extraction de points d'intérêt, appariement et calcul de champs de mouvements).

Title : **Contributions to Augmented Reality applications.**
Visual tracking and 2D registration.
3D object tracking using point-based models.

Abstract : This thesis presents several registration methods for augmented reality applications. We first present two applications which use 2D composition techniques (based on homography estimation or non-rigid image transformation). We then propose a 3D visual tracking approach that makes use of a point-based model of the object. The idea is to explain the motion of feature points matched from frame to frame in terms of the variation of the 3D pose parameters. For that we compute 2D motion fields induced by elementary 3D motions. We adapt the splatting technique developed by the computer graphics community to render the model and approximate the motion fields anywhere on the surface. To avoid the drift of the estimation process, we use a set of keyframes which poses are determined offline. Our implementation also exploits the graphic processing units (GPU) for its most computationally intensive parts.

Remerciements

Je remercie tout d'abord Alain Ayache d'avoir accepté de diriger cette thèse. Je remercie tous les membres de l'équipe VORTEX et en particulier les membres historiques de l'équipe VPCAB, pour leur accueil et les enrichissantes discussions que nous avons pu avoir.

Au début de ma thèse je travaillais dans une grotte. Je remercie Jean-Pierre Jessel qui m'a offert l'opportunité de travailler dans d'autres grottes, plus intéressantes, celles de Gargas dans les Hautes-Pyrénées et qui m'a fait le plaisir de présider mon jury. Je remercie Jean-Marc Lavest et Pascal Guitton pour m'avoir fait l'honneur de rapporter mon travail et Vincent Lepetit pour avoir gentiment accepté de faire le déplacement jusqu'à Toulouse pour m'écouter et partager son expertise.

J'adresse ma chaleureuse reconnaissance à mes encadrants Géraldine Morin et Vincent Charvillat, pour leur foi inébranlable en la stabilité de mon bateau. Je les remercie également de m'avoir confié des charges d'enseignement dans des matières qui m'ont beaucoup intéressé et beaucoup appris.

Je tiens à remercier les personnes qui ont contribué à rendre la vie au laboratoire si agréable. Merci donc à Pascaline, Matthijs, Clovis, Romulus, Cézar, Vincent, Aurélie, Romain, Ahmed, Pierre-Loïc, Sandrine, Nassima, Stéphane, Benoit, Sébastien, Jean-Charles, Marco. Je dirige un clin d'oeil particulier à Ariel en repensant à ces quelques mois passés à construire ensemble ce qu'il a ensuite su transformer en une aventure industrielle dont j'apprécie désormais l'envergure au quotidien.

Enfin je remercie Papa et Maman pour leur soutien constant pendant toutes mes études et les cinq années de ce travail.

Table des matières

Table des figures

Chapitre 1

Introduction

Une piste essentielle pour faciliter l'accès aux systèmes informatiques est de concevoir des mécanismes interactifs plus élaborés et plus adaptés que les périphériques traditionnellement utilisés, clavier et souris en particulier. L'utilisation des modalités vocale, gestuelle ou haptique ou une combinaison de plusieurs d'entre elles a été intensivement étudiée ces dernières années par la communauté d'interaction homme-machine. La communication entre l'utilisateur et la machine est alors «inversée» : la machine doit, de manière autonome, acquérir des informations sur l'utilisateur ou son environnement pour le comprendre. La Réalité Augmentée est l'un des mécanismes d'interaction qui permettent de mettre en place cette communication, d'une part en précisant ce que «compréhension de l'environnement» signifie pour le système et d'autre part en proposant des mécanismes de présentation attestant et exploitant cette compréhension, par l'ajout d'éléments de synthèse à la perception de l'utilisateur.

Sans rentrer dans la présentation des concepts détaillés au chapitre 2, le problème clé de la Réalité Augmentée est la compréhension de la structure géométrique (et éventuellement photométrique) de l'environnement de l'utilisateur (la «scène»). On peut distinguer deux grands types d'approches : d'une part les techniques par apprentissage, qui, sur la base d'exemples, permettent de généraliser un certain savoir à la scène particulière observée ; d'autre part les approches qui modélisent explicitement la scène ou ses objets d'intérêt afin de réaliser les tâches utiles : notamment détecter, reconnaître, suivre et enrichir ces objets. La complexité et la variabilité des scènes à analyser rend la première démarche délicate. De plus, un modèle est dans tous les cas nécessaire pendant la phase d'apprentissage. Notre travail s'inscrit dans la deuxième démarche, c'est-à-dire que nous postulons la connaissance d'un modèle a priori de la scène. La question de fond soulevée dans ce travail concerne la nature du modèle utilisé en suivi visuel 3D pour la Réalité Augmentée.

Idées maitresses de nos travaux

Une application innovante des principes de Réalité Augmentée est un dispositif d'essayage virtuel de lunettes, représenté à droite sur la figure 1.1. À gauche de cette figure

FIG. 1.1 – Suivi 3D du visage avec un maillage pour une application d'essayage de lunettes virtuelles. Société FittingBox.

l'algorithme de suivi du visage employé est illustré : un maillage grossier est utilisé comme connaissance a priori de la scène et des points d'intérêt sont détectés et suivis dans chaque image de la vidéo. Pour relier leur mouvement à la transformation globale tridimensionnelle du visage, il faut remonter à la facette 3D puis évaluer un modèle de mouvement local des points image. Il faut enfin exprimer le mouvement global en fonction des mouvements locaux mesurés. Ces niveaux d'indirection sont-ils nécessaires ? Nous montrons dans ce travail que ce n'est pas le cas.

On peut ensuite se poser la question de la finesse optimale du modèle de l'objet à suivre. En effet, un maillage trop grossier est parfois insuffisant ; pour les maillages très fins, la nécessité de la topologie est discutable. À des fins de suivi, l'échantillonnage du modèle n'a pas besoin d'être plus fin que la résolution de l'image. A l'instar des cartes de profondeur, qui représentent en chaque point de l'image la distance à la caméra du point de la scène correspondant, nous proposons dans nos travaux une représentation dense, à la résolution des images, des mouvements 2D observés en chaque pixel.

Dans cette thèse, nous défendons l'idée que les **modèles utilisés pour le suivi 3D ne requièrent pas nécessairement d'informations topologiques comme celles apportées par un maillage**.

Les modèles par points que nous utilisons présentent par ailleurs plusieurs avantages :

- dans le contexte de la Réalité Augmentée on s'intéresse à des modèles représentatifs des objets réels. Les dispositifs d'acquisition 3D tels que les scanners laser ou les algorithmes de reconstruction à partir d'images fournissent en sortie un ensemble de points. Ce dernier doit être maillé si les algorithmes utilisés par la suite s'appuient sur des facettes. L'utilisation directe de modèles par points permet de s'affranchir de cette étape ;
- ils offrent une représentation compacte et simple de la géométrie de l'objet à suivre, qui n'est pas surchargée par une information topologique inutile ;

– une représentation de surface par facettes et textures suppose un mouvement apparent 2D qui est localement une transformation homographique. Une représentation dense par points colorés, bien que plus coûteuse en mémoire offre également une modélisation plus fine du mouvement dans les images.

Notons également que les méthodes utilisant le suivi par des arêtes échantillonnent les contours en des points, ce qui suggère que l'information topologique n'est qu'une aide algorithmique.

Les travaux sur le suivi visuel 3D présentés dans cette thèse montrent que l'on peut utiliser un modèle par points à toutes les étapes d'un algorithme de suivi : pour la prédiction en tout point du mouvement inter-image, pour la construction d'image clés à pose référencée et pour la synthèse du modèle texturé.

Plan du document

Nous développerons ces idées selon le plan suivant, qui s'organise en trois grandes parties. Les trois premiers chapitres forment un état de l'art couvrant la Réalité Augmentée, le suivi visuel 3D et les représentations par points. Dans une deuxième partie nous décrivons notre approche de suivi visuel utilisant un modèle par points. Enfin, dans une dernière partie, nous présentons un travail de mise en œuvre des tâches bas niveau de notre l'algorithme de suivi exploitant les GPU modernes.

Dans le premier chapitre, nous présentons la Réalité Augmentée et une revue des technologies développées et mises en œuvre ces dernières années dans ce domaine. Comme nous le verrons, le concept de mélange d'objets virtuels et réels à des fins d'interaction s'est concrétisé en une grande variété d'applications. Parallèlement au «noyau dur» de notre travail qui concerne le suivi visuel 3D, nous avons réalisé deux applications de Réalité Augmentée simples s'appuyant sur la composition d'objets réels ou virtuels bidimensionnels que nous présentons dans ce chapitre. Le deuxième chapitre aborde la thématique centrale de ce travail, le suivi visuel 3D temps réel. Il s'agit d'une instance particulière d'un problème clé de vision par ordinateur qui consiste à déterminer la position et l'orientation d'une caméra à partir des images filmées. Nous présentons le formalisme utilisé dans la suite du document et étudions quelques unes des nombreuses approches proposées. Même si des avancées significatives ont été faites récemment, c'est un problème encore ouvert. Le chapitre 3 termine cette première partie d'état de l'art par une présentation des géométries par points introduites et développées par la communauté d'informatique graphique que nous adapterons.

Dans une deuxième partie, nous présentons la contribution majeure de ce travail, en détaillant d'une part les aspects théoriques et d'autre part les aspects pratiques de la mise en œuvre d'un algorithme de suivi visuel utilisant les représentations par points. Le chapitre 4 présente en détail notre approche de suivi utilisant des modèles par points. Nous analysons un des meilleurs algorithmes de l'état de l'art utilisant un modèle par

facettes et surmontons les difficultés d'adaptation à un nuage de points. Nous présentons une solution qui permet, grâce à une mise en œuvre efficace sur GPU, de calculer des champs de mouvement denses, conduisant à une résolution linéaire itérative du problème de suivi de la pose. Pour compenser les effets de dérive inhérents à ce type d'algorithme, nous utilisons un ensemble d'image clés. Le modèle par points de l'objet est ici aussi utilisé pour le référencement de points 3D et pour la synthèse de nouvelles vues. Le chapitre 5 présente les détails de mise en œuvre d'algorithmes de rendu par *splatting*. Nous détaillons notre adaptation de cet algorithme pour l'approximation dense des champs de mouvement 2D. Nous montrons que cette mise en œuvre est compatible avec une exécution en temps réel dans l'algorithme présenté au chapitre 4.

Le chapitre 6 constitue la dernière partie de ce manuscrit. Nous y décrivons plus en détails les problématiques de mise en œuvre des méthodes d'extraction et d'appariement de points d'intérêt. Il s'agit en particulier d'exploiter les capacités de calculs des CPU et des GPU modernes. Nous montrons que les mises en œuvre sur GPU sont d'un ordre de grandeur plus rapides que celles sur CPU.

En conclusion, nous dressons le bilan de ce travail et formulons quelques une des nombreuses perspectives possibles de ce travail.

Contributions

Ces travaux comportent trois principales contributions : la proposition de deux applications de Réalité Augmentée simples, le développement d'un algorithme de suivi 3D à partir d'un modèle et une mise en œuvre temps-réel de cet algorithme exploitant le GPU.

Au tout début de ce travail doctoral, nous avons contribué à la réalisation d'une application de suivi de fond, qui est au cœur de la thèse de Matthijs Douze [Dou04]. Nous avons plus récemment réalisé un dispositif interactif assistant un utilisateur dans son interprétation de gravures préhistoriques. Ces contributions modestes mettent en lumière la diversité des problématiques de Réalité Augmentée, au travers des questions de création des scènes augmentées et d'interaction avec l'utilisateur.

Nous nous sommes placés dans le cadre général de l'application des modèles, des méthodes et des outils développés par la communauté de l'informatique graphique pour le traitement des nuages de points. Notre contribution à la transposition de ces outils au domaine de la vision par ordinateur repose sur trois points :

- le calcul de prédicteurs ponctuels de mouvement et la reconstruction dense dans l'espace image d'un champs de mouvement apparent ;
- une formulation linéaire itérative de la solution du problème d'estimation de la pose sur la base de ces prédicteurs ;

- l'utilisation d'un algorithme de rendu texturé d'un modèle par points pour générer de nouvelles vues à partir d'une image clé.

Nous contribuons enfin par la mise en œuvre en temps-réel de l'algorithme de suivi proposé. Ceci est possible grâce à

- la mise en œuvre sur GPU de la détection et de la mise en correspondance de points d'intérêt entre images consécutives et entre une image et une image clé ;
- la mise en œuvre sur GPU et l'évaluation des performances des algorithmes de *splatting*.

Chapitre 2

Concepts et applications de la Réalité Augmentée

Ce chapitre présente dans une première partie (section 2.1) les concepts de la Réalité Augmentée, qui est le cadre applicatif de nos travaux. Nous abordons les problématiques théoriques et les difficultés de mise en œuvre des systèmes utilisant la Réalité Augmentée. Dans une seconde partie (section 2.2) nous présentons deux applications de Réalité Augmentée que nous avons développées, utilisant des principes de composition 2D 1/2 ou 2D. L'une fait appel à un algorithme de suivi de fond, l'autre utilise un dispositif intératif non standard. Ces deux contributions font l'objet de deux publications [DDCM04, DCC07]. Nous discutons de leurs limitations et justifions l'approche de suivi 3D que nous développerons dans le reste du document.

2.1 Concepts et applications

Les systèmes de Réalité Augmentée ont pour but, dans un environnement réel, de combiner des objets réels et virtuels (c'est-à-dire générés par ordinateur) en une même scène. Contrairement à la Réalité Virtuelle, où l'utilisateur est totalement immergé au sein d'un monde virtuel, la Réalité Augmentée vise à intégrer, par superposition, les objets virtuels dans l'environnement réel de l'utilisateur. Les objets de synthèse enrichissent la perception que les utilisateurs ont de leur environnement en fournissant ou en facilitant l'accès à des informations inaccessibles directement. Ils permettent par exemple de faire apparaître les objets réels cachés, de présenter des annotations graphiques ou textuelles ou encore de superposer des informations issues de capteurs (de température par exemple). On peut en ce sens définir la finalité des applications de Réalité Augmentée comme l'accompagnement de l'utilisateur dans une tâche réelle afin de la rendre plus facile à exécuter.

Bien que les environnements réels et virtuels soient souvent considérés comme étant définis en 3D et que la cohérence spatiale 3D soit recherchée, en général toute forme de composition d'objets visuels réels et virtuels est considérée comme faisant partie du domaine de la Réalité Augmentée, du moment que les deux types d'objets paraissent coexister

de manière cohérente au sein du même environnement. Les expérimentations présentées en section 2.2 montrent des techniques de recalage et de composition 2D 1/2 et 2D. Cette définition est celle formulée par Azuma [Azu97], qui définit la Réalité Augmentée comme le domaine des applications qui :

- combinent le réel et le virtuel,
- fonctionnent en temps réel et
- donnent l'apparence que les objets virtuels et réels cohabitent dans le même monde tridimensionnel.

Milgram [MK94] définit un continuum d'expériences de «Réalité Mixte» allant de l'environnement réel aux mondes virtuels. La Réalité Augmentée est un degré intermédiaire de ce continuum, présentée sur la figure 2.1. On pourrait par exemple classer tout à gauche les systèmes de vidéo-conférence et de visiophonie, tout à droite les dispositifs immersifs tels que les casques de réalité virtuelle ou les systèmes CAVE (Cave Automatic Virtual Environment) [CNSD93].

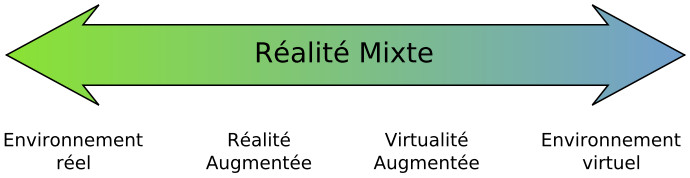


FIG. 2.1 – Continuum réalité-virtualité de Milgram.

2.1.1 Domaines d'application de la Réalité Augmentée

Nous donnons ici un aperçu des domaines d'application des systèmes de Réalité Augmentée. Les revues d'Azuma et al. [Azu97, ABB+01] comportent une riche bibliographie des premiers travaux significatifs réalisés ces dernières années.

Applications médicales Un des domaines les plus prometteurs est celui de l'assistance au geste chirurgical. Les systèmes de Réalité Augmentée peuvent être utilisés lors des opérations pour superposer des données médicales sur le corps du patient. Les données superposées peuvent être directement issues de radiographies ou d'images IRM (on recale alors le plan de coupe de ces prises de vue sur la vue du corps du patient) ou bien être un modèle 3D reconstruit à partir d'une succession de scans (qu'il faut également recaler correctement vis-à-vis de la vue). Le chirurgien possède alors une vision étendue des organes sans avoir recours à des techniques plus intrusives. Sato et al. [SNT+98] proposent un système permettant la superposition sur un flux vidéo d'une tumeur cancéreuse dont un modèle 3D est reconstruit à partir d'images ultrasons.

Une autre application est l'apprentissage et la validation de gestes médicaux, les différentes étapes d'une opération pouvant être superposées à la vue de l'apprenti, qui n'a

plus à se détourner du patient pour consulter un manuel. Les objets virtuels peuvent également servir à la localisation des organes. Sielhorst et al. présentent un simulateur pour l'entraînement des praticiens à l'accouchement [SOB+04]. Dans ce système, un casque de réalité virtuelle est utilisé pour augmenter la vue d'un mannequin avec les représentations tridimensionnelles des os du bassin, du foetus et de forceps.

Maintenance et assemblage industriel En superposant les instructions de montage ou de réparation, présentées par exemple sous forme de modèles 3D animés des pièces à manipuler, il est possible de faciliter ces opérations. Les coûts, parfois importants, de maintenance (stockage et mise à jour) des manuels papiers sont également réduits. Plusieurs évaluations ont par ailleurs montré l'apport des systèmes de Réalité Augmentée pour la productivité dans les chaînes de montages [WOSL01] et pour la maintenance et la réparation [PHMG06] dans les environnements industriels.

Divertissement ARQuake [TCD+00] est l'une des premières expérimentations d'un système de Réalité Augmentée pour le jeu vidéo. Le jeu se déroule en extérieur et les éléments virtuels sont intégrés dans l'environnement physique en combinant différents capteurs (GPS, compas numérique et suivi visuel de marqueurs placés dans l'environnement). La disponibilité de caméras bon marché (*webcam*) et la puissance de calcul grandissante des consoles de jeux et des ordinateurs permettent l'essor de jeux utilisant l'interaction vidéo et la superposition d'objets virtuels. Dans le jeu *The Eye of Judgment* [Inc07], le joueur déplace sur un plateau des cartes localisées et suivies à l'aide d'une caméra. À l'écran (dont deux captures sont montrées en figure 2.2) des créatures sont dessinées à l'emplacement de chaque carte.

FIG. 2.2 – Images issues du jeu *Eye of Judgment*. Tous droits réservés Sony Computer Entertainment Inc.

Dans le domaine de la production audiovisuelle, les techniques de suivi de caméra permettent d'étendre la technique classique du «fond vert» en réintégrant en temps réel un acteur dans un environnement virtuel qui remplace le fond. Contrairement aux bulletins météorologiques, le fond n'est pas statique et s'adapte à la position de la caméra. De nombreux événements sportifs diffusent également des images augmentées, permettant par exemple la visualisation des lignes de hors jeu, des distances à l'embut au football et au

rugby ou de la marque du meilleur saut ou du meilleur lancer en athlétisme. Les panneaux publicitaires peuvent également être modifiés virtuellement afin de les adapter au pays de diffusion.

Des expérimentations ont été menées pour aider la localisation des visiteurs dans un musée et proposer des annotations contextuelles attachées aux œuvres présentées (voir par exemple [WWWC04]).

2.1.2 Dispositifs matériels

Des dispositifs matériels spécifiques sont nécessaires à la mise en œuvre des systèmes de Réalité Augmentée pour résoudre deux problématiques : la première est la présentation de la composition finale des objets réels et virtuels ; la deuxième concerne la détermination des informations géométriques sur la scène permettant un placement cohérent des objets visuels dans la scène réelle (voir plus loin à propos des «modes » de cohérence que l'on peut définir).

Systèmes de visualisation La combinaison du réel et du virtuel peut être réalisée par différents dispositifs, que l'on peut ranger en deux catégories. La première est celle des dispositifs pour lesquels l'utilisateur ne voit le monde que par l'intermédiaire d'un écran : c'est le cas en particulier des casques *Video See-Through*. Une caméra, généralement montée sur le casque, filme la scène réelle et le résultat de la composition avec les objets de synthèse est renvoyé sur l'écran. La deuxième catégorie regroupe les cas où les objets de synthèse sont affichés par superposition avec la vision naturelle. Les casques *Optical See-Through*, équipés d'un écran translucide, mettent en œuvre ce principe.

Lee et al [LHSD05] proposent un dispositif d'affichage original reposant sur un projecteur et une surface plane rectangulaire que l'utilisateur peut déplacer librement devant le cône de projection. Des capteurs de luminosité placés aux quatre coins de la surface permettent de déterminer sa position. L'image projetée est alors adaptée pour n'illuminer que la surface de projection, simulant un dispositif d'affichage portable. Il s'agit d'une extension du principe des *Magic Lenses* [BSP+93].

Capteurs de position et de mouvement Pour assurer le positionnement précis des objets de synthèse, un dispositif permettant de repérer les objets réels de la scène vis-à-vis du point de vue augmenté doit être mis en place. Dans le cas où la vue de l'utilisateur passe par l'intermédiaire d'une caméra et d'un écran, le repère de référence dans lequel il faut déterminer les objets de la scène est celui de la caméra. Dans le cas des systèmes *Video See-Through*, le repère de référence est celui de l'œil de l'utilisateur. Les objets réels à suivre dépendent de l'application. Pour les systèmes de guidage de personnes, il peut s'agir de l'environnement global de l'utilisateur. Dans le cas de la maintenance, généralement seule la position de l'objet à réparer (et éventuellement de ses différentes pièces mobiles) est nécessaire. Pour l'aide aux opérations chirurgicales, une partie seulement du corps

du patient est généralement repérée, ainsi que les instruments du praticien (endoscopes, forceps, etc.).

Pour que la coexistence des objets de synthèse et des objets réels soit convaincante, le système de suivi doit répondre à certaines contraintes. L'objectif est que le système facilite et non perturbe la tâche à accomplir.

On peut distinguer les contraintes suivantes :

- Latence réduite. La latence est le temps qui sépare le déplacement d'un objet réel et celui d'un objet de synthèse associé. À partir de quelques dizaines de millisecondes, la latence introduit des décalages perceptibles entre les objets réels et virtuels.
- Fréquence de mise à jour. La position des objets virtuels doit être mise à jour suffisamment souvent. Ceci est particulièrement important lorsqu'on utilise des lunettes semi-transparentes puisque les changements dans la vue réelle sont perçus instantanément. Dans le cas où la composition passe par un écran, le rafraîchissement de la position est limité par celui de l'écran.
- Précision. La précision du recalage des objets virtuels est cruciale pour des raisons de sécurité dans le cas des applications médicales. Le phénomène de *capture visuelle* ([Wel78]), c'est-à-dire la prépondérance du sens visuel sur les autres sens, rend toute erreur de recalage perturbante pour effectuer des mouvements précis. De nombreux facteurs influent sur la précision finale du recalage, indépendamment des contraintes dynamiques précédentes, notamment les jeux mécaniques, les distorsions optiques et les erreurs propres au système de suivi.
- Volume de déplacement suffisant. Certaines applications, notamment en extérieur requièrent la capacité de suivre des déplacements de grande amplitude.

Parmi les systèmes de suivi proposés dans la littérature, peu satisfont simultanément les contraintes citées ci-dessus. En production audiovisuelle la technologie la plus courante utilise des caméras montées sur des têtes et des chariots mécaniques qui rapportent la position et l'orientation à la régie. La précision et la fréquence de mise à jour sont assez bonnes, mais le volume de suivi est limité par les butées mécaniques. Les systèmes magnétiques sont sensibles à la présence de métaux dans l'environnement. Les systèmes utilisant des ultrasons fournissent un résultat bruité et ne sont pas adaptés aux grands volumes où les variations de température ambiante deviennent sensibles. Les capteurs inertiels ont tendance à diverger au cours du temps, les erreurs de recalage sont alors de plus en plus importantes si l'ont ne recalibre pas fréquemment le système.

Dans nos travaux, nous avons développé un système de suivi s'appuyant sur le traitement du flux vidéo produit par une caméra (voir chapitre 5). À partir des images numérisées, on extrait des indices caractéristiques dans la scène qui sont utilisés pour le recalage.

Interface haptique Certains systèmes de Réalité Augmentée combinent les dispositifs de suivi avec des interfaces à retour d'effort, comme des gants haptiques. Les chercheurs de l'ETH Zurich proposent par exemple une application de ping pong où la raquette est associée à un bras à retour d'effort pour simuler les impacts d'une balle virtuelle [BKSH06].

2.1.3 Aspects logiciels

Les systèmes de Réalité Augmentée conduisent à des problématiques de conception logicielle particulières. Le comportement de ces systèmes est principalement contrôlé par les flux continus de données issus des différents capteurs mis en œuvre. La quantité d'information à traiter est importante et un soin particulier doit être apporté à la réduction des coûts de calcul, particulièrement en environnement mobile. En effet, de l'efficacité des traitements mis en œuvre dépend la qualité de l'interaction. Lorsque le suivi s'appuie sur l'analyse d'un flux vidéo, comme c'est notre cas, une cadence d'images minimale doit être maintenue, afin que le retour visuel induit par une modification du réel soit convaincant. Il est également souhaitable de limiter le décalage temporel (aussi appelé latence) entre l'instant de prise de vue et l'instant d'affichage de la scène augmentée. Un retour d'information rapide et fluide permet également d'accroître la sensation d'immersion lorsque l'utilisateur interagit directement avec un élément réel. Lorsque les conditions pour une interaction satisfaisante avec le système sont remplies, on parle de traitements *en temps interactif*. Cela correspond à une cadence d'image de l'ordre de 10 images par seconde ou plus et une latence inférieure à 100 millisecondes. Dans le chapitre 7, nous nous posons la question de l'optimisation des traitements bas-niveau de notre algorithme (décrit au chapitre 5) afin de satisfaire le budget de temps imparti par image.

La complexité de conception des applications de Réalité Augmentée a amené certains chercheurs à développer des méthodes de conceptions logicielles spécifiques comme la notation ASUR++ [DGN03]. Depuis quelques années le prototypage d'applications de Réalité Augmentée utilisant une caméra pour le recalage a été simplifié grâce à la disponibilité de bibliothèques logicielles gratuites, comme ARToolkit [ART] ou OpenCV [Int08].

2.2 Compositions 2D d'objets visuels

Nous présentons ici deux contributions qui ont en commun de mettre en œuvre un recalage bidimensionnel de différents objets visuels qui peuvent être des images, des vues de synthèse statiques ou dynamiques d'une scène virtuelle, des annotations (graphiques ou textes) ou encore des vidéos. Ces applications suivent le même principe que les approches tridimensionnelles, en particulier on peut distinguer deux grandes étapes : la première consiste en le recalage des objets visuels afin de garantir la cohérence de la scène finale. Cette étape est la plus cruciale et implique de déformer tout ou partie des objets visuels utilisés. La seconde consiste en la composition des objets recalés, qui correspond généralement à un mélange pixel à pixel des différentes couches obtenues dans la première étape.

2.2.1 Réalité augmentée 2D par suivi d'arrière plan

Dans cette application qui est au cœur du doctorat de Matthijs Douze et pour laquelle nous avons contribué [DDCM04], nous supposons disposer d'une ou plusieurs vues panoramiques d'une scène (partie gauche de la figure 2.3), ramenées à des projections centrales planes de centres optiques communs. Ces vues panoramiques constituent une

connaissance a priori du fond de la scène que nous supposons figé. L'éclairage est également supposé constant. L'intérêt principal dans ce contexte est de faciliter la réalisation de scènes augmentées géométriquement cohérentes, puisqu'une simple image panoramique (ou éventuellement un rendu panoramique dynamique) est nécessaire.

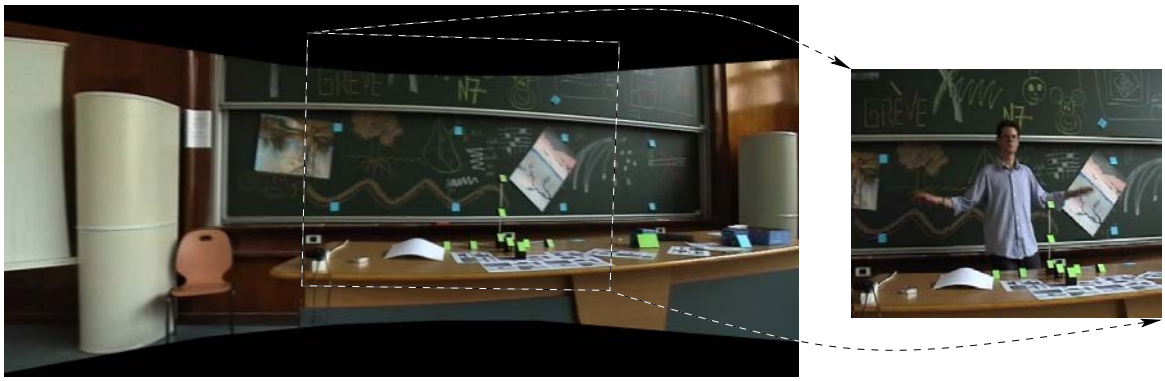


FIG. 2.3 – Mise en correspondance, basée sur une homographie, d'une image de la séquence vidéo avec la vue panoramique de référence.

Des objets et/ou des personnages s'ajoutent alors à la scène et une caméra filme ce qui se passe. La caméra dont le centre optique est invariant, peut changer de direction de visée et de distance focale. Si la caméra est montée sur un pied, le nombre de degrés de liberté peut être limité à 3 (deux angles de rotation –inclinaison et azimut– et un facteur de zoom).

Un algorithme permet alors d'analyser le mouvement apparent du fond de la scène. Il s'agit de mettre en correspondance les images de la séquence avec l'image panoramique de référence. Cette tâche est un problème de suivi (deux images successives sont « proches ») à base d'images géométriquement liées par des homographies 2D. Cela est illustré par la figure 2.3.

En considérant **l'image panoramique en entier comme la cible 2D**, nous retrouvons le contexte du suivi d'un motif plan en présence d'occultations. En conséquence nous savons «extraire» l'objet visuel «fond» sur chaque image de la séquence d'entrée (figure 2.4).

Par un processus de «soustraction», nous savons également détecter les intrus, c'est-à-dire les objets visuels qui se rajoutent à la scène figée initiale. Cette «soustraction» nécessite des procédés d'alignement et de correction photométrique que nous ne développons pas ici, mais qui sont illustrés sur la figure 2.5.

Ces fonctionnalités possibles nous permettent également d'opérer des compositions «fond,forme» nombreuses dont certaines se retrouvent chez d'autres auteurs [IAB+96, DM96], ainsi que dans le contexte du codage MPEG-4 [PE02]. Nous décrivons ici deux scénarios de réalité augmentée.

Ajout d'un objet de synthèse Il s'agit d'insérer un objet de synthèse dans la séquence vidéo pour aboutir à une intégration cohérente spatiotemporellement avec le « décor »

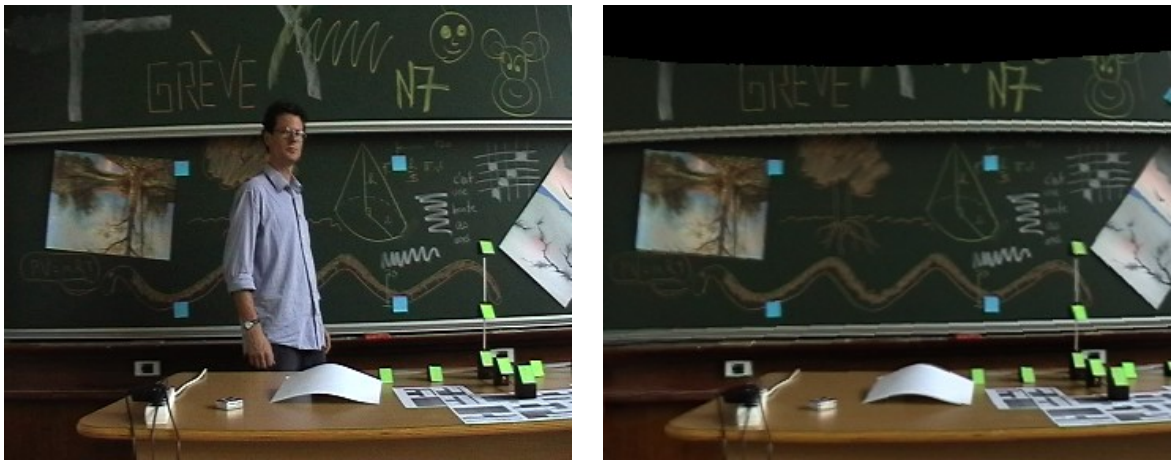


FIG. 2.4 – Image de la séquence vidéo et portion de l'image panoramique de référence après alignement géométrique.

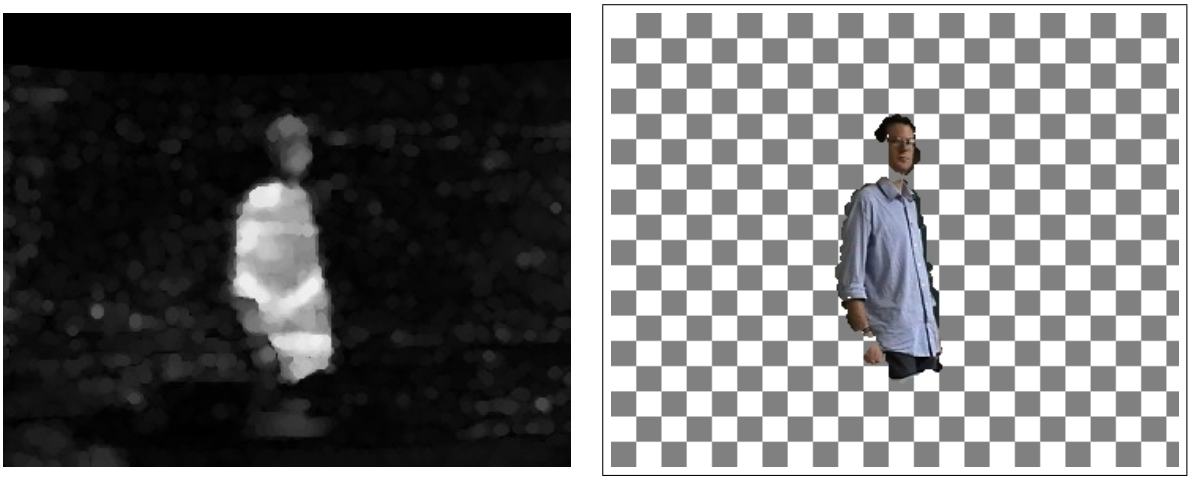


FIG. 2.5 – Masque d'occultation et premier plan extrait (les parties transparentes sont représentées en échiquier gris et blanc).

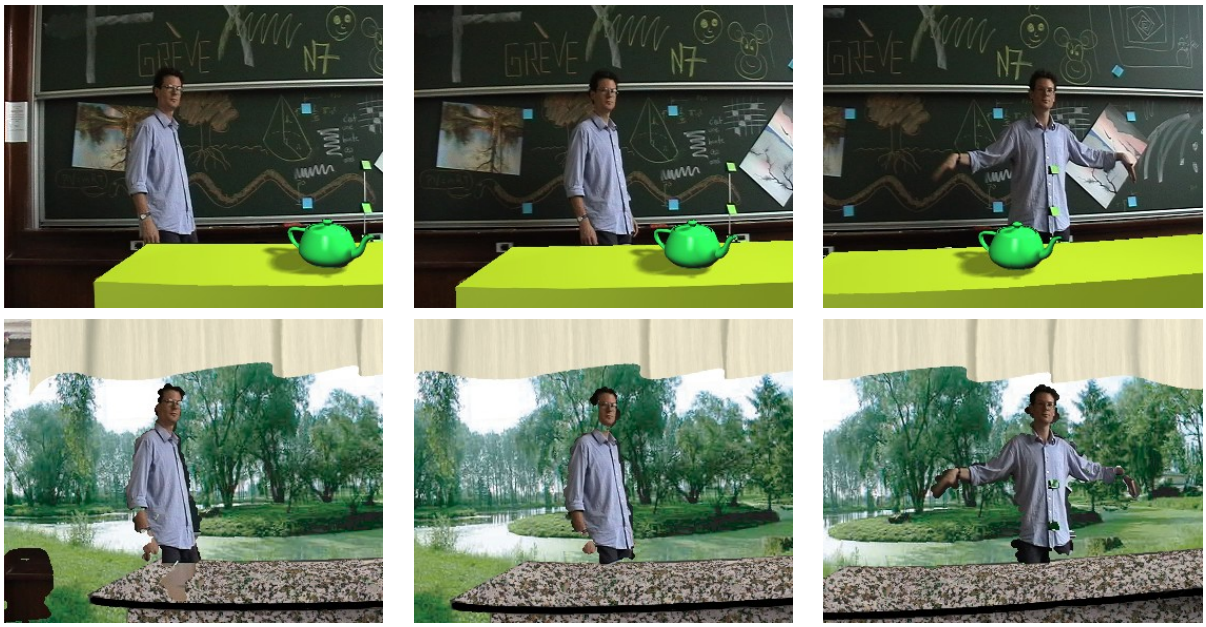


FIG. 2.6 – Première ligne : Réalité Augmentée avec insertion d'un objet de synthèse (bureau virtuel). Deuxième ligne : Réalité Augmentée avec extraction de l'objet occultant.

existant. Cet objet est en premier plan et ne subit aucune occultation en provenance des composantes réelles.

Pour cela, il suffit de superposer les objets visuels suivants : l'image courante de la séquence vidéo et un calque contenant l'image artificielle d'une augmentation. Ce peut être un objet 3D virtuel dont l'image est calculée en temps réel ou une image panoramique précalculée porteuse de l'augmentation.

La première ligne de la figure 2.6 illustre ce scénario. Le mouvement du bureau virtuel est cohérent avec le reste de la scène : il reste fixe par rapport au tableau.

Détection d'un objet occultant Il s'agit d'extraire de la scène les objets/personnages occultants et les représenter au premier plan d'un décor virtuel.

Pour cela, il suffit de superposer les objets visuels suivants : une image panoramique modifiée. Un dessinateur peut la préparer à partir de photographies ou d'éléments synthétiques et le calque contenant l'objet occultant extrait de la séquence vidéo. La deuxième ligne de la figure 2.6 illustre ce scénario. L'image panoramique modifiée est un mélange artistique d'éléments synthétiques et de photographies. Ce «faux fond» a le même mouvement apparent que la scène réelle. Le déplacement du personnage ne permet pas de déceler le trucage !

FIG. 2.7 – Une paroi gravée. L'interprétation des formes visibles est difficile sans l'aide d'un expert.

2.2.2 Augmentation interactive de gravures préhistoriques

La contribution présentée ici se distingue de la précédente essentiellement sur deux aspects : tout d'abord le recalage entre les différents objets visuels utilisés est effectué hors ligne et non en temps réel. Ensuite la caractéristique principale de cette application est sa dimension interactive [DCC07].

Notre équipe travaille depuis un an avec les organisations et les collectivités territoriales en charge de l'exploitation et de la mise en valeur de la grotte de Gargas située dans les Hautes Pyrénées (voir le site internet de présentation des grottes de Gargas, `http://grottesdegargas.free.fr/`). Les grottes de Gargas sont surtout connues pour leurs nombreuses « mains négatives » de l'époque Gravettienne (-27000 ans) mais elles possèdent aussi de magnifiques gravures. Nous avons conçu le prototype d'une application touristique de découverte interactive de ces gravures.

Le dispositif réalisé [DCC07] consiste à aider l'utilisateur dans sa lecture ou son interprétation des gravures. Le principe est d'augmenter des images naturelles des parois en donnant au visiteur un moyen de découvrir interactivement les interprétations des gravures émises par les préhistoriens. On part de photographies de parois gravées difficilement interprétables sans l'aide d'un expert, comme celle montrée en figure 2.7.

Pour aider cette lecture, le visiteur peut aussi observer dans l'entrée du site des relevés précis et annotés par un préhistorien (Pr. C. Barrière) qui permettent différentes interprétations du fouillis apparent formé par les incisions et les gravures observables sur les parois. Les figures 2.8 (a) et (b) montrent un relevé (issu d'une publication de C. Barrière) et une version annotée en couleur permettant d'interpréter l'ensemble de la paroi dîte «de la Conque».

Notre travail a d'abord consisté à recaler ces relevés sur des images naturelles de la paroi comme le montre la figure 2.9. Les relevés graphiques ont pour cela été numérisés, prétraités et segmentés avant le recalage de portions de relevés sur les images numériques acquises par nos soins. Le recalage basé sur des amers manuels utilise un modèle non-rigide dont on contrôle la complexité avec un processus de sélection de modèle [CB05].

(a) (b)

FIG. 2.8 – (a) Relevés et interprétation du préhistorien C. Barrière. (b) Relevés annotés faisant apparaître différents animaux.

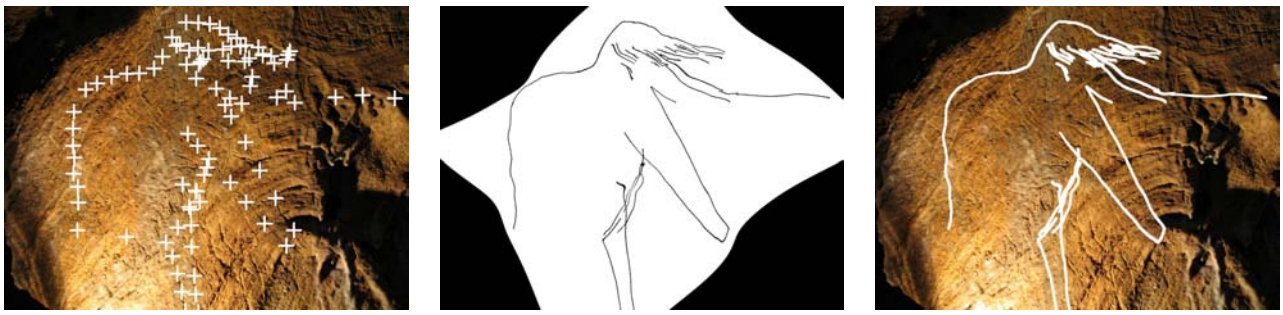


FIG. 2.9 – Recalage non-rigide des relevés du préhistorien.

Dispositif interactif et mise à disposition au public

Le système développé exploite un tableau magnétique qui rapporte à un ordinateur les coordonnées d'un stylo en contact avec le tableau. La surface est également utilisée comme écran sur lequel les images composées sont reprojetées. Le schéma de la figure 2.10 (a) montre ce dispositif.

Un logiciel interactif permet à l'utilisateur de sélectionner la gravure particulière (c'est-à-dire la combinaison d'une photo et d'une forme) qu'il souhaite «explorer». La découverte se fait en partant de la projection de la photographie seule, puis l'utilisateur se sert du stylo pour désigner les zones de son choix. Les zones correspondant aux tracés recalés (et seulement elles) sont ainsi révélées progressivement. Le principe de cette composition est décrit en figure 2.11.

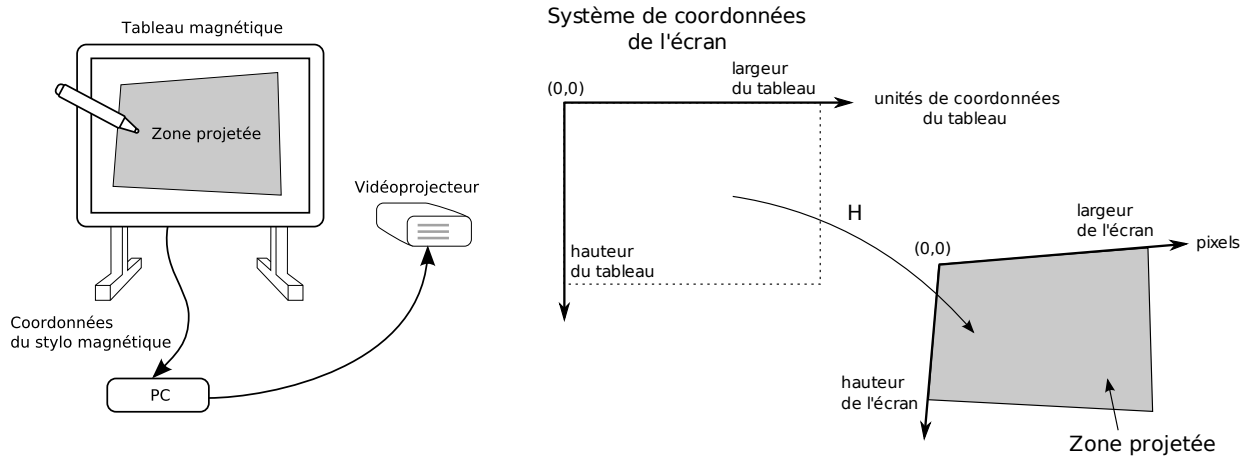
Un module permet également de corriger la déformation de l'image projetée due principalement au non parallélisme du plan de projection et du plan du tableau. Cette déformation, modélisée par une homographie est retrouvée par une procédure de calibrage consistant à cliquer les quatre coins de l'image avec le stylo (voir figure 2.10 (b)). Ce processus est réalisé une fois pour toutes au moment de l'installation.

Cette interaction très simple permet d'envisager plusieurs scénarios d'utilisation :

- le premier est un fonctionnement de type borne interactive, pour lequel un visiteur peut explorer seul une base de données photographiques annotées et ainsi enrichir et/ou préparer une visite avec un guide.
- dans le second, un guide utilise le dispositif comme support de ses commentaires. Ce scénario est complémentaire de la visite réelle puisque qu'il permet de présenter des gravures parfois inaccessibles au public pour des raisons de sécurité ou de conservation de la paroi.
- on peut également imaginer des scénario multi-utilisateurs, par exemple impliquant un enseignant et un élève. En effet le dispositif reconnait l'utilisation de plusieurs stylos et présente les annotations dans une couleur différente, comme on le voit sur la photographie de gauche sur la figure 2.12.

Nous avons pu constater l'intérêt de chacun de ces scénarios lors de différentes manifestations et séances de démonstration publiques du dispositif : pour la fête du centenaire des grottes et pour la fête de la science à Aventignan (65), pour des séances scolaires à la maison du savoir de St Laurent de Neste (65), à la maison de Midi-Pyrénées (31) et lors du salon ScientiLivre (Toulouse 2007). Les enfants sont parmi les utilisateurs les plus enthousiastes du dispositif (figure 2.12). Nous pensons que l'accès très intuitif et une grande tolérance invitant les utilisateurs novices à apprendre par l'essai sont les forces de notre système.

Des démarches sont en cours en vue d'une version « professionnalisée » de notre prototype de laboratoire dans le cadre du financement d'un pôle d'excellence rural auquel nous participons. Une des améliorations que nous envisageons est d'utiliser un système permettant une projection par l'arrière de l'écran, afin d'éviter les ombres (actuellement nous atténuons ce phénomène en désaxant fortement le projecteur et en utilisant d'une part une correction optique et d'autre part la procédure de calibrage de la figure 2.10 (b)).



(a) (b)

FIG. 2.10 – (a) Principe du dispositif : le tableau magnétique renvoie à un ordinateur les coordonnées du stylet magnétique. (b) Correction des coordonnées rapportées par le tableau pour les replacer dans les coordonnées de l'écran.

Cette application d'augmentation d'images naturelles est donc une démonstration prometteuse des possibilités offertes par des mécanismes de composition simple pour la mise en œuvre pratique de systèmes interactifs ludiques et pédagogiques.

2.3 Réalité Augmentée 3D

Les deux applications présentées précédemment utilisent des principes de composition 2D par couches, pour lesquels la position des objets réels dans l'espace n'est pas nécessaire. Ceci limite la généralisation de ces techniques à des mécanismes d'interaction relativement simple. Nous présentons ici les principes de la Réalité Augmentée 3D, qui s'appuient sur le recalage dans l'espace 3D des objets et la détermination des paramètres de la caméra. Le principe de l'ajout d'objets de synthèse 3D repose sur la mise en coïncidence des caméras réelle et virtuelle.

2.3.1 Cohérence géométrique

L'une des caractéristiques principales garantissant l'illusion que les objets de synthèse cohabitent dans le même monde que les objets réels est la cohérence géométrique. Cela revient à aligner les caméras virtuelle et réelle de sorte que les objets virtuels semblent avoir été filmés avec la caméra réelle. Plusieurs techniques permettent de retrouver les paramètres de la caméra, c'est-à-dire la déformation perspective d'une part et sa position dans la scène, par rapport à un repère global (appelé aussi repère monde). Certaines s'appuyent sur le

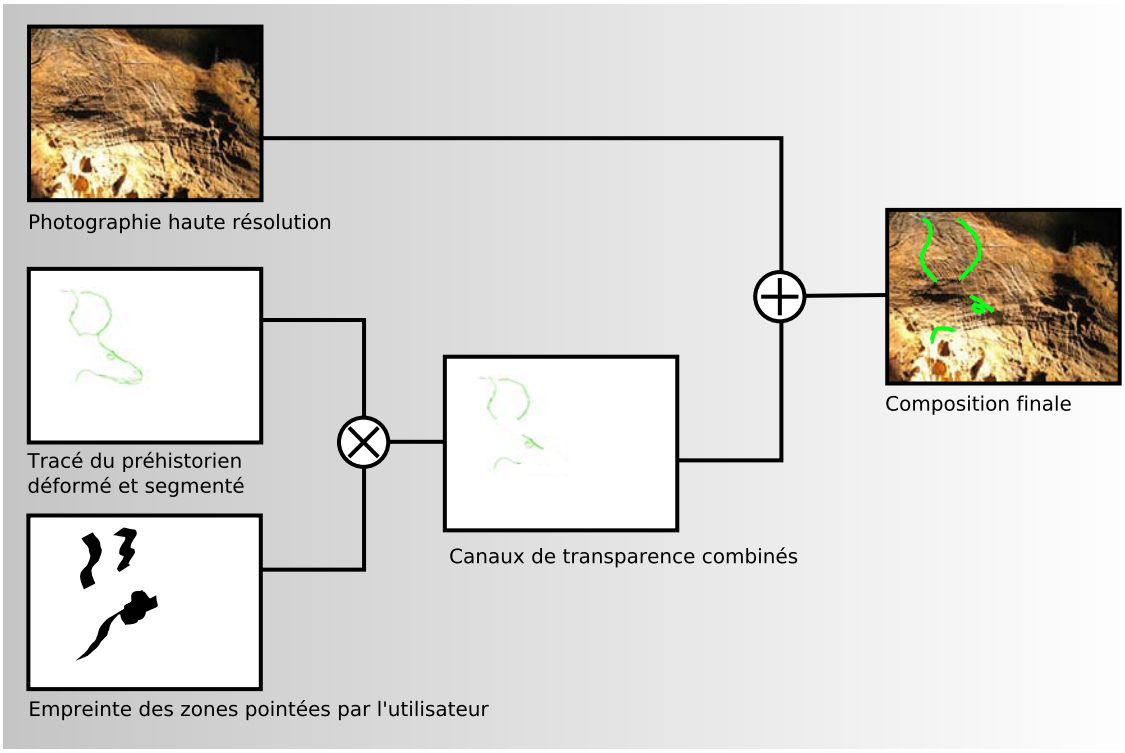


FIG. 2.11 – Composition de l'image haute résolution de la paroi avec le tracé déformé et segmenté du préhistorien. La zone composée est limitée aux régions sélectionnées par l'utilisateur.

FIG. 2.12 – Quelques photos prises lors de démonstrations en public du système.

placement de marqueurs dans la scène, d'autres sur la connaissance du modèle de certains objets.

L'alignement cohérent des caméras permet de placer les objets au bon endroit dans l'image finale. Cependant la cohérence géométrique peut être cassée si l'on ne considère pas les occultations des objets réels par les objets virtuels. La solution consiste généralement à modéliser les objets réels occultants et à les prendre en compte dans l'étape de gestion de la visibilité du moteur de rendu.

2.3.2 Cohérence photométrique

Pour augmenter encore le réalisme de l'intégration des objets de synthèse, il faut considérer la cohérence photométrique, c'est-à-dire simuler sur les objets de synthèse l'éclairage de la scène réelle dans laquelle ils sont plongés. C'est un problème difficile, car il nécessite l'estimation de l'illuminance de la scène, dont la modélisation fait intervenir la position et la nature des sources lumineuses et la réflectance des surfaces des objets. Pour traiter les ombres portées des objets réels, il est nécessaire de les modéliser et éventuellement de les suivre individuellement s'ils ne sont pas statiques. Debevec [Deb98] a initié les travaux autour de l'estimation de l'illumination d'une scène réelle à partir de photographies de sphères réfléchissantes (*light probes*). Boivin et Gagalowicz [BG01] proposent un programme permettant l'estimation des réflectances des surfaces des objets réels d'une scène à partir de plusieurs photographies. Les données acquises permettent un rendu de la scène augmentée prenant en compte l'illumination globale. Loscos et al. [LDR00] proposent un outil interactif permettant de changer les paramètres des lumières d'une scène photographiée, en prenant en compte les ombres portées. Ces procédés restent très coûteux en calculs et ne peuvent être envisagés pour des applications temps réel. Le niveau d'interaction est cependant suffisant pour des applications d'aménagement intérieur et de prototypage architectural.

2.3.3 Cohérence spatio-temporelle

La cohérence spatiotemporelle concerne les occultations entre les objets de synthèse et les objets réels. Là encore un traitement correct implique généralement de modéliser et de suivre les objets réels potentiellement occultants. C'est un problème difficile, chaque objet supplémentaire introduisant 6 paramètres à estimer. Les techniques les plus adaptées actuellement permettent la localisation simultanée de plusieurs marqueurs plans, auxquels les objets virtuels sont attachés. Notre système de suivi 3D (voir chapitre 5) suit la position d'un seul objet (mais de forme quelconque).

2.4 Conclusion

Le domaine de la Réalité Augmentée touche de nombreuses problématiques matérielles et logicielles. Aucune solution de suivi n'est actuellement satisfaisante dans tous les cas et

peu de systèmes sont suffisamment robustes pour sortir de cadres applicatifs très spécifiques où l'utilisateur est au préalable entraîné ou supervisé lors de la manipulation du système. Nous pensons que les approches s'appuyant sur la vision par ordinateur offrent la souplesse nécessaire pour, à terme, permettre aux concepts de Réalité Augmentée d'atteindre le grand public.

Les deux réalisations présentées, utilisant des approches de composition 2D, sont selon nous des contributions intéressantes : la première, en se plaçant dans le cadre restreint des scènes panoramiques, facilite grandement la création d'environnement augmentés ; la seconde montre l'intérêt de mécanismes d'interaction originaux pour la découverte par le grand public de sites protégés.

Pour des applications de Réalité Augmentée plus complexes et moins restreintes, les mécanismes de composition en trois dimensions restent les plus prometteurs. L'algorithme que nous avons développé s'appuie sur l'analyse en temps réel d'un flux vidéo pour le suivi de la pose d'un objet réel de forme quelconque, dont on connaît un modèle 3D. C'est un problème de suivi visuel 3D, abondamment exploré ces dernières années. Le chapitre suivant est une revue de la littérature sur ce sujet. Il nous permettra de positionner nos contributions vis-à-vis de l'état de l'art.

Chapitre 3

État de l'art du suivi visuel 3D

Dans ce chapitre, nous présentons des méthodes permettant, à l'aide des images successives produites par une seule caméra, de retrouver la position et l'orientation d'un objet de la scène par rapport à la caméra. Nous nous intéressons en particulier aux méthodes pouvant être mises en œuvre en temps réel ou interactif, c'est-à-dire exploitables pour des applications de Réalité Augmentée. Souvent les méthodes de suivi temps réel sont des extensions d'algorithmes étudiés initialement sans contrainte temporelle. L'étude de ces algorithmes est donc également d'intérêt ici.

Le problème de l'estimation de la pose 3D d'un objet à partir d'un flux d'images est un problème difficile, en particulier dans le cadre des applications interactives où la caméra est manipulée par l'utilisateur (tenue à la main ou attachée à sa tête). Dans ces conditions les objets de la scène subissent des mouvements quelconques, parfois très rapides et qu'il est difficile de prédire. Comme dans ce type de systèmes l'interaction repose par essence sur la connaissance de la pose à chaque instant, le mécanisme d'estimation doit être le plus fiable possible ou capable de détecter les échecs d'estimation pour se réinitialiser. À ces difficultés s'ajoutent des contraintes de temps d'exécution très fortes, en particulier lorsque la plateforme cible est un système embarqué (par exemple un PDA). Ceci explique la rareté des solutions commerciales de Réalité Augmentée s'appuyant sur la vision par ordinateur.

Le chapitre pose tout d'abord le formalisme mathématique de formation d'une image par une caméra et formule le problème d'estimation de la pose (section 3.1). Dans une deuxième partie (section 3.2), un panorama des solutions temps réel à ce problème est dressé.

3.1 Formalisme mathématique

Nous rappelons ici le formalisme décrivant le processus de formation d'image par une caméra. Les concepts et les outils de base sont issus de la géométrie projective [HZ03, Fau93, FL01].

Modèle de caméra Le modèle de caméra le plus couramment utilisé, car il s'applique à la plupart des dispositifs d'acquisition vidéo[1], est le modèle de caméra dit sténopé ou *trou d'épingle*.

La formation de l'image dans un tel modèle est mathématiquement régie par la projection perspective d'un point 3D de l'espace sur le plan image, telle que présentée sur le schéma de la figure 3.1.

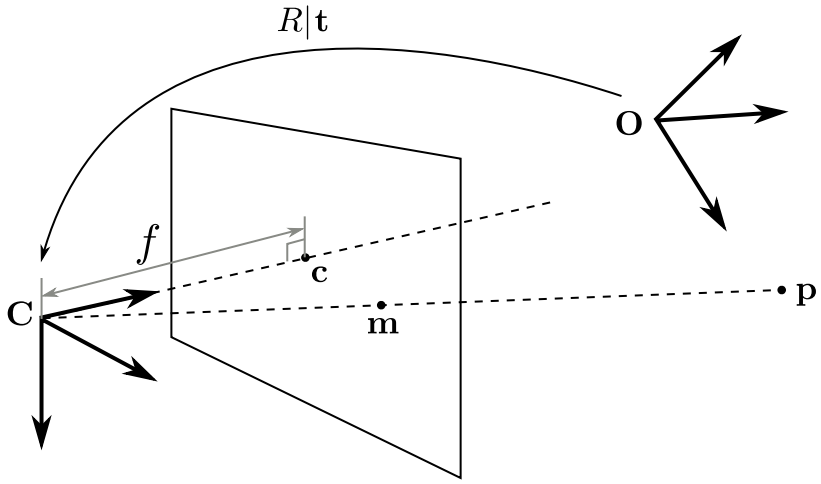


FIG. 3.1 – Projection perspective d'un point 3D $\mathbf{p}$ en un point 2D $\mathbf{m}$ dans le plan image.

Un point $\mathbf{p} = (x, y, z)^\top$ de l'espace 3D euclidien se projette en un point 2D $\mathbf{m} = (u, v)^\top$ de l'image tel que

$$\lambda \widetilde{\mathbf{m}} = P \widetilde{\mathbf{p}} \tag{3.1}$$

où $\widetilde{\mathbf{m}} = (u, v, 1)^\top$ et $\widetilde{\mathbf{p}} = (x, y, z, 1)^\top$ sont les coordonnées des points $\mathbf{m}$ et $\mathbf{p}$ dans les espaces projectifs $\mathcal{P}_2$ et $\mathcal{P}_3$. La matrice P est une matrice de projection 3×4 définie à un facteur d'échelle λ près et dépend donc de 11 paramètres. Une telle matrice peut se décomposer selon la forme

$$P = K[R|\mathbf{t}] \tag{3.2}$$

avec :

- K la matrice de calibrage, de taille 3×3 qui dépend des paramètres internes de la caméra telle que la distance focale.
- R et $\mathbf{t}$ sont les composantes de rotation et de translation de la transformation euclidienne permettant de passer du repère monde (de centre $\mathbf{O}$) au repère caméra (de centre le centre de projection $\mathbf{C}$). La matrice R est une matrice de rotation 3×3

[1] hormis les optiques *fish eyes* et les caméras omnidirectionnelles utilisant par exemple des miroirs paraboliques.

(c'est-à-dire orthogonale et de déterminant 1). Le vecteur $\mathbf{t}$ est un vecteur de $\mathbb{R}^3$. Ces deux entités peuvent être décrites par 6 paramètres (3 de rotation et 3 de translation), rassemblés en vecteur $\beta \in \mathbb{R}^6$, qui forment la *pose de la caméra*.

Calibrage La matrice de calibrage K dépend de 5 paramètres (11−6), appelés paramètres internes de la caméra. On peut écrire :

$$K = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

où :

- α_u et α_v sont les facteurs d'échelle dans la direction u et v du repère image. Ces facteurs sont proportionnels à la longueur focale f de la caméra : on a $\alpha_u = k_u f$ et $\alpha_v = k_v f$ avec k_u et k_v les densités de pixels du capteur de la caméra dans les directions u et v.
- $(u_0, v_0)^\top = \mathbf{c}$ est le *point principal*, défini comme la projection du centre optique sur le plan image (voir figure 3.1).
- s correspond à l'obliquité (ou coefficient de cisaillement horizontal) des cellules du capteur. On a $s = 0$ lorsque celles-ci sont carrées, ce qui est le cas de la majorité des caméras modernes[2].

Les hypothèses simplificatrices mais raisonnables que l'on peut formuler sont de prendre $\mathbf{c}$ au centre du capteur et de considérer des pixels carrés, c'est-à-dire $s = 0$ et $\alpha_u = \alpha_v$. Lorsque la matrice K est déterminée, on dit que la caméra est calibrée.

Distorsion radiale L'optique d'une caméra induit généralement des déformations que le modèle sténopé présenté plus haut ne modélise pas. Ces déformations sont d'autant moins négligeables que la caméra est bon marché et que sa focale est petite. On modélise généralement ces phénomènes par une distorsion radiale, c'est-à-dire un déplacement des points de l'image radialement autour du centre de l'image. Les points en coordonnées non déformées $(u', v')^\top$ sont obtenus à partir des points observés $(u, v)^\top$ selon le modèle

$$\begin{aligned} u' &= u + (u - u_0)L(r) \\ v' &= v + (v - v_0)L(r) \end{aligned} \tag{3.4}$$

avec L le polynôme

$$L(r) = k_1 r^2 + k_2 r^4 \ , \ r = \sqrt{(u - u_0)^2 + (v - v_0)^2}. \tag{3.5}$$

Les facteurs k_1 et k_2 sont retrouvés par la procédure de calibrage (voir ci-après), ou par une méthode distincte s'appuyant par exemple sur la mise en correspondance des lignes de la scène.

[2] s peut être non nul dans le cas où l'on compose plusieurs systèmes d'acquisition, par exemple lorsqu'on photographie une photographie.

Méthodes de calibrage pour l'estimation des paramètres internes La plupart des méthodes d'estimation de pose en temps-réel supposent que les paramètres internes de la caméra sont connus et constants. Ceci signifie en particulier que la caméra ne zoome pas ; un changement de distance focale est en effet difficile à distinguer d'une translation le long de l'axe optique. Si les paramètres internes de la caméra ne sont pas connus, il est nécessaire de les déterminer expérimentalement, hors ligne.

Ce problème a été largement abordé par les chercheurs en vision par ordinateur. Deux types de solutions ont été proposées : les premières utilisent une ou plusieurs vues d'un objet tridimensionnel appelé mire de calibrage, dont on connait précisément la structure. Les images des points caractéristiques de la mire (dont la position 3D est connue) peuvent être déterminées à la main ou automatiquement, ce qui fournit des correspondances 2D-3D à partir desquelles la matrice P peut être estimée [LVD98]. On peut ensuite remonter à K par exemple en calculant la décomposition QR de la sous-matrice 3×3 KR (voir équation 3.2). Parmi les méthodes entièrement automatiques, citons celle de Zhang [Zha00], qui utilise une simple grille plane apparaissant dans plusieurs vues (au minimum deux).

Les secondes méthodes analysent une séquence d'images d'une scène statique générale et déduisent les paramètres internes à partir de la mise en correspondance (éventuellement automatique) d'indices visuels dans les images (points, ellipses, plans, etc.) [GS03]. Ces dernières méthodes sont appelés méthodes d'autocalibrage.

Formulation du problème Nous formulons ici le problème d'estimation de la pose auquel les méthodes présentées ci-après proposent une solution.

On suppose que les paramètres internes (K, k_1, k_2) de la caméra sont connus et fixes, et on souhaite déterminer la pose β_t correspondant à l'image I_t acquise par la caméra à l'instant t. L'historique des images précédentes $\{I_0, \ldots, I_t\}$ est accessible, mais généralement seules I_{t-1} et I_t sont utilisées pour des raisons d'efficacité.

3.1.1 Suivi de caméra sans modèle

Le suivi de caméra sans modèle a priori de la scène a toujours été un des problèmes clé en vision par ordinateur. A partir des fondements mathématiques présentés plus haut et ceux issus de la géométrie des images multiples, deux grands types d'approches ont été développées : le SLaM (*Simultaneous Localization and Mapping*) et la SfM (*Structure from Motion*). Dans les deux cas on cherche à retrouver d'une part les poses de la caméra au cours d'une séquence d'images et d'autre part la structure 3D de la scène observée. La plupart des travaux en SfM ne prennent pas en compte la contrainte de temps réel. À partir d'une estimation initiale grossière utilisant peu de données (quelques vues, quelques amers), la solution est raffinée lors d'une étape d'optimisation finale d'ajustement de faisceaux, prenant en compte toutes les données à la fois. Les approches SLaM ont leurs origines dans le domaine de la robotique mobile, elles intègrent donc certaines contraintes de temps d'exécution et la notion de causalité : l'estimation est récursive et les incertitudes sont propagées au fur et à mesure que les données sont acquises par le robot. Notons cependant que lorsque la précision de la carte est nécessaire, les appproches SfM sont préférables.

Royer et al. proposent ainsi un système de navigation par vision monoculaire dans lequel une carte 3D de l'environnement est calculée hors ligne lors d'une phase d'apprentissage où le robot est commandé manuellement [RLDL07]. Pendant la navigation, le robot se localise en temps réel dans la carte.

Produit commerciaux De nombreux produits commerciaux ont été développés pour résoudre le problème du suivi de caméra dans le cadre de la post-production audiovisuelle. Ces solutions visent la qualité et la généralité du suivi et plutôt que le temps réel. Citons les logiciels PFTrack de la société PixelFarm [Pix], RealViz Match Mover [Rea] et Boujou [Bou]. Ces outils proposent des procédures fortement automatisées et aboutissent à des résultats très précis.

3.2 Taxonomie des méthodes temps réel

L'estimation de la pose peut être décomposée en deux étapes :
- une étape de traitement d'images, visant à obtenir des mesures du mouvement de la scène ou des indices de la position d'un objet ;
- une étape d'estimation proprement dite utilisant les mesures précédentes pour déterminer la pose.

Dans cette section, nous avons classé les différentes approches de la littérature en fonction du type d'indices visuels recherchés et appariés dans les images. Le choix de ces indices est central lors de la conception d'un algorithme de suivi visuel car il a un impact important sur ses conditions d'utilisation (en particulier le type d'objet auquel il sera adapté).

Les indices visuels utilisés peuvent être des marqueurs ajoutés à la scène, choisis pour leur facilité de détection, d'identification et de suivi ou bien des indices naturellement présents dans les images, par exemple les contours des objets, les points d'intérêt ou les régions texturées.

3.2.1 Méthodes utilisant des marqueurs

On peut distinguer des marqueurs de deux natures différentes : les marqueurs ponctuels, qui fournissent une correspondance entre un point 3D de la scène et son image 2D et les marqueurs planaires qui apportent une information plus riche, typiquement la transformation reliant le plan du marqueur à sa projection dans l'image. La position dans l'espace des marqueurs est supposée connue avec précision ; elle peut être déterminée à la main, à l'aide de télémètres laser ou encore par des algorithmes de reconstruction multivues. Chaque correspondance fournit deux contraintes linéaires, il est donc en général possible d'estimer la pose de la caméra à partir de la localisation de 6 marqueurs [HZ03, chapitre 7].

Les marqueurs ponctuels proposés utilisent le plus souvent des formes circulaires ou elliptiques pour représenter le point 3D de l'espace correspondant à leur centre. Hoff et al utilisent par exemple des marqueurs formés de deux disques concentriques noir et blanc [HLN96]. Les marqueurs sont localisés par seuillage de l'image et détection des régions

noires et blanches : un marqueur est détecté lorsque le barycentre d'une région noire coïncide avec celui d'une région blanche.

Les marqueurs planaires apportent à la fois plus d'information géométrique et permettent l'identification des marqueurs. Dans le cas de la bibliothèque ARToolkit [ART], un marqueur est formé d'une carte blanche sur laquelle un carré à bordure noire épaisse est imprimé. La détection des quatre coins du carré suffit à retrouver la pose du marqueur (lorsque la caméra est calibrée). On peut également calculer, grâce à ces mesures, la déformation homographique du marqueur et rectifier l'image d'une forme binaire dessinée à l'intérieur du carré, afin de la comparer à une base de donnée. Ainsi différentes annotations peuvent être attachées et maintenues sur chaque marqueur. Fiala [Fia04] propose une amélioration de ce principe avec des marqueurs spécialement conçus pour leur distinctivité, chacun encodant 36 bits d'information.

Outre la nécessité d'instrumenter soigneusement l'environnement, l'inconvénient principal des méthodes s'appuyant sur des marqueurs est d'être restreint à leur zone de visibilité. En effet un marqueur peut devenir indétectable lorsque la caméra s'éloigne.

3.2.2 Méthodes utilisant les contours

Les contours sont sûrement les primitives qui ont reçu le plus d'attention, et ce dès les premières approches de suivi sans marqueur. Si l'on exclut l'approche de Lowe [Low92], qui utilise une détection de contour préalable sur une large portion de l'image, toutes les méthodes mettent en œuvre une recherche locale des contours de l'image depuis des points de contrôle répartis sur les contours d'un modèle 3D de l'objet projeté dans l'image. Cette recherche est monodimensionnelle, le long d'une ligne orientée dans la direction de la normale du contour projeté.

Dans la méthode RAPiD [Har92], un modèle CAO de l'objet est projeté dans une pose prédite à l'aide d'un filtre de Kalman puis les contours sont localisés par une recherche linéaire locale à partir de points de contrôle échantillonnés sur les contours projetés. Les contours visibles sont déterminés grâce à une structure de donnée précalculée, indexée par la pose courante. La pose est ensuite mise à jour en minimisant la distance entre les contours projetés (prédits) et les contours détectés.

De nombreuses variations ont été proposées autour de ce schéma initial notamment pour améliorer la robustesse vis-à-vis des mesures erronnées : Armstrong et Zissermann [AZ95] utilisent pour cela l'algorithme RANSAC, Drummond et Cippolla [DC99] formulent la résolution avec un M-estimateur.

Comport et al. utilisent la mesure du déplacement de points échantillonnés sur des primitives segment et cercle [CMPC06]. L'estimation de la pose est formulée comme un problème d'asservissement visuel virtuel et utilise un M-estimateur pour résister aux mesures aberrantes.

La détection des points de contours le long de la ligne de recherche peut prendre en compte les frontières de régions texturées au lieu de détecter les fortes valeurs de gradient, comme le propose Shahrokni et al. [SDF05]. La probabilité de changement de texture est

modélisée le long de la ligne de recherche par un modèle de Markov caché, mis à jour pendant le suivi.

Récemment Klein et Murray ont proposé un suivi temps réel robuste à partir des contours utilisant intensivement l'accélération matérielle des cartes graphiques modernes [KM06]. Un filtre à particules similaire à CONDENSATION [IB98] est utilisé. Chaque particule représente une pose probable de l'objet. La vraisemblance d'une particule est entièrement calculée sur la carte graphique, ce qui permet d'en utiliser plusieurs centaines. Les arêtes non visibles sont éliminées grâce aux fonctions standards du matériel graphique et la fonction de vraisemblance est calculée par un *fragment shader*[3]. Contrairement aux approches précédentes, la totalité des pixels des contours visibles est utilisée.

3.2.3 Méthodes utilisant des points d'intérêt

Les méthodes précédentes sont bien adaptées aux objets comportant des arêtes saillantes : maquettes, salles, pièces industrielles, etc. Pour les objets « naturels », trop peu d'information de contour est généralement disponible ou les contours s'apparentent à du fouilli. Il est alors nécessaire de prendre en compte la texture locale de l'objet, typiquement échantillonnée en des points d'intérêt[4].

Vacchetti et al. obtiennent des résultats très convaincants en utilisant un maillage surfacique de l'objet suivi et la mise en correspondance de points d'intérêt [VLF04a]. La pose initiale est donnée, puis les points sont projetés sur le modèle. Il est alors possible de relier le mouvement 2D des points dans l'image au changement de pose 3D. Les déplacements mesurés, obtenus à partir des correspondants de points peuvent ainsi être comparés aux déplacements prédits. Les poses courantes et précédentes sont conjointement estimées par minimisation de la distance entre les positions mesurées et prédites des points d'intérêt. Comme la mise à jour de la pose repose sur la projection des points selon la pose précédente, le processus est *en boucle ouverte* et accumule les erreurs d'estimation. Pour contrecarrer cet effet, les auteurs utilisent un ensemble d'image clés. La pose d'une image clé est précisément déterminée hors ligne et le nuage 3D formé des points d'intérêt reprojetés sur le modèle fournit une représentation statique de l'apparence de l'objet (sous la forme de fenêtres autour des points). La déformation de l'apparence de l'image clé la plus proche vers la pose courante estimée permet la mise en correspondance des points entre la vue courante et l'image clé (potentiellement assez éloignée). Des correspondances 3D-2D sont ainsi établies et viennent compléter le critère d'optimisation utilisant les correspondances entre images successives. Le problème d'optimisation non linéaire obtenu est résolu avec un M-estimateur afin de minimiser l'impact des appariements erronés.

Rosten et Drummond [RD05] utilisent un détecteur de points très efficace et présentent une méthode d'estimation de la pose robuste à un nombre élevé d'appariements erronés,

[3] Un fragment shader est un petit programme exécuté sur la carte graphique permettant de manipuler les pixels rendus par la carte graphique. Le chapitre 6 aborde plus en détails la programmation des cartes graphiques modernes.

[4] Une description détaillée des méthodes d'extraction et d'appariemment des points d'intérêt est faite au chapitre 7.

une situation notamment observée lors des déplacements très rapides de la caméra. Cette solution repose sur l'apprentissage au cours du suivi d'une fonction reliant la valeur du critère d'appariement entre points (en l'occurrence SDC, voir chapitre 7) à la probabilité que l'appariement soit correct. Les faux appariements sont rejetés sur la base de cette distribution en utilisant l'algorithme *Expectation Maximisation* pour prendre en compte sa forte multimodalité.

Les premières approches performantes de type SLaM monoculaire et temps réel ont vu le jour il y a quelques années [NNB04, DRMS07]. Depuis, grâce à des puissances de calcul croissantes et des algorithmes plus astucieux, il est devenu possible de rajouter des fonctionnalités tout en gardant l'aspect temps réel. Ceci a notamment permis d'effectuer, en temps réel, des ajustement de faisceaux locaux sur un sous-ensemble de la séquence vidéo, ce qui permet une propagation moins approximative des incertitudes que dans le SLaM traditionnel et accroît de manière significative la stabilité et la précision des résultats [ESN06, MLD+07, KM07].

Notre approche s'appuye également sur la mise en correspondance de points d'intérêt, entre images consécutives et avec une image clé. La particularité de notre mise en œuvre est d'exploiter la carte graphique pour l'extraction et la mise en correspondance des points. Notre algorithme de suivi visuel est présenté au chapitre 5 et les détails de mise en œuvre concernant les points d'intérêt au chapitre 7.

3.2.4 Méthodes utilisant des motifs texturés

Pour extraire plus d'information sur l'apparence de l'objet, des indices s'appuyant sur la texture globale ou locale de l'objet sont utilisés. La formulation la plus générale est celle du suivi de motifs (ou *template matching*) qui utilise un ou plusieurs motifs de référence, par exemple donnés sur la première image et cherche le déplacement successif de ces motifs entre deux images consécutives.

L'algorithme de suivi de motifs de Lucas et Kanade [LK81, BM04] utilise des motifs rectangulaires (*patchs*) et un modèle de déformation de ce motif (par exemple une transformation affine 2D). Les paramètres de déformation sont ajustés itérativement afin de minimiser l'erreur quadratique entre le motif déformé et l'image. Puisque la transformation est arbitraire cette méthode peut être utilisée pour suivre aussi bien des régions 2D [HB98, BJ98] que la pose 3D [CSA00].

Ce principe général de suivi de motif est également utilisé dans les approches populaires utilisant des modèles d'apparence comme les AAM (*Active Appearance Model*) [CET01]. Dans ces approches, l'objet est représenté par un maillage 2D texturé ; des modes de déformation géométrique et d'apparence sont appris à partir de séquences d'apprentissage par analyse en composantes principales. Ainsi l'espace des apparences de l'objet est décrit de manière minimale par des combinaisons linéaires de vecteurs de forme (déformation du maillage) et de vecteurs d'apparence (variation de la texture). Des techniques similaires étendent ces approches à des modèles 3D [BV99].

Masson et al. utilisent un modèle constitué d'une multitude de *patchs* planaires dont le déplacement inter-image est régi par des homographies [MDJ04]. L'information de tex-

ture au sein des *patchs* est échantillonnée en des points d'intérêt. Les auteurs utilisent la technique de suivi de motifs développée par Jurie et Dhome [JD02] puis combinent les estimations individuelles en utilisant l'algorithme d'estimation de pose de Lowe [Low85].

Lepetit et Fua proposent l'utilisation de motifs distinctifs et formulent de manière originale le problème de leur mise en correspondance comme un problème de classification [LF06]. Dans une phase d'apprentissage, les motifs centrés sur des points d'intérêt sont reprojetés sur un modèle 3D de l'objet à suivre, et un grand nombre de vues de ces motifs sont générées automatiquement. La dimensionnalité de l'espace des motifs est réduite par analyse en composantes principales et les ensembles de vues sont groupés par la technique $k-$means. La mise en correspondance des motifs extraits dans l'image courante avec ceux construits pendant la phase d'apprentissage est effectuée à l'aide d'un arbre de décision aléatoire, ce qui aboutit à un algorithme rapide (en temps interactif) et robuste.

3.2.5 Méthodes de suivi hybrides

Chacune des approches présentées précedemment a des intérêts et des limites spécifiques. Les approches s'appuyant sur les contours ne sont pas sensibles à l'accumulation d'erreurs, mais peuvent être perturbées par les contours des autres objets de la scène. Les méthodes utilisant des correspondances successives de points sont généralement plus robustes aux occultations partielles de l'objet, mais le processus d'estimation étant en *boucle ouverte* (car les points ne sont que temporairement localisés sur le modèle), l'accumulation d'erreurs et l'effet de dérive sont inévitables. Les approches par détection (telle que celle de Lepetit et Fua citée plus haut) sont encore un peu coûteuses et difficiles à généraliser pour les environnements très vastes et changeants. Une démarche naturelle consiste alors à développer des méthodes hybrides combinant plusieurs approches, typiquement en stabilisant un suivi sensible à la dérive avec des mesures plus stables.

Certaines approches combinent le suivi visuel avec les mesures d'un ou plusieurs capteurs. Par exemple, Klein et Drummond combinent un suivi de contours avec trois capteurs inertiels (gyroscopes) mesurant la vitesse angulaire d'un casque de Réalité Virtuelle [KD03]. L'intégration temporelle des mesures des gyroscopes permet d'initialiser le suivi visuel près de la pose réelle, même lors des mouvements rapides. Les mesures de vitesse fournissent également une estimation du flou de bougé, ce qui permet d'adapter les filtres utilisés pour le calcul du gradient de l'image. Pour une application de Réalité Augmentée embarquée, les mêmes auteurs fusionnent les estimations issues d'un suivi visuel à partir des contours, fonctionnant sur un tabletPC avec le suivi de marqueurs constitués de LEDs attachés à la tablette [KD04]. La fusion des mesures utilise un filtre de Kalman étendu.

Vacchetti et al. rajoutent des informations a priori sur la scène sous forme d'images clés. Dans [VLF04b] ils combinent les mesures issues de contours et de points d'intérêt. Pressigout et Marchand [PM06] utilisent le formalisme d'asservissement visuel virtuel présenté plus haut et incorporent au suivi des arêtes d'un modèle CAO un critère portant sur la texture des faces du modèle. À chaque face est associée une image de texture de référence et le transfert des points d'intérêt situés à la surface du modèle est exprimé en fonction de la pose cible. Le critère de texture repose sur la différence des niveaux de gris sur une fenêtre

autour des points d'intérêt dans l'image courante et des points transférés dans l'image de texture.

3.3 Conclusion

De nombreuses avancées ont été réalisées ces dernières années dans le domaine du suivi visuel 3D. Les premières approches temps réel et robustes ouvrent la voie à l'utilisation de ces techniques dans le cadre de la Réalité Augmentée. Les méthodes les plus efficaces utilisent des informations a priori sur la scène ou l'objet suivi et requièrent par exemple un modèle 3D. Peu de réflexions ont été menées sur la nature de ce modèle.

L'approche que nous développons au chapitre 5 a les caractéristiques suivantes :

- l'objet à suivre est modélisé par un nuage de points,
- elle s'appuye sur la mise en correspondance de points d'intérêt entre images successives pour la mise à jour de la pose,
- elle exploite des images clés pour éviter le phénomène de dérive.
- elle utilise les capacités des cartes graphiques modernes pour la manipulation du modèle et pour l'extraction et la mise en correspondance des points d'intérêt.

Le chapitre suivant présente un état de l'art des représentations géométriques d'objets 3D par des nuages de points et des techniques de rendu et de manipulation de ces représentations, qui nous seront utiles dans le cadre du suivi.

Chapitre 4

Représentations et traitements des géométries à base de points

L'approche de suivi visuel 3D que nous présenterons au chapitre 5 a été développée avec l'objectif d'utiliser, comme représentation 3D des objets suivis, des modèles à base de points[1]. Nous avons en particulier identifié le besoin de deux procédures couramment utilisées dans les systèmes de rendu d'objets géométriques : la procédure de lancer de rayon, c'est-à-dire la capacité d'intersecter la surface avec une demi-droite issue du centre de la caméra et passant par un point image donné, et la procédure de projection de la totalité de la surface sur le plan image. Cette projection s'accompagne généralement d'un mécanisme d'interpolation des caractéristiques de la surface : la couleur, les normales ou toute autre propriété géométrique (au chapitre 5 nous interpolerons des bases de vecteurs de mouvement).

Ces deux processus permettent d'utiliser les représentations par points comme modèles d'objets non déformables. Nous complétons cette présentation générale par les différentes techniques permettant le traitement des représentations par points, de l'acquisition à l'édition. Si notre cadre de travail se limite aux objets rigides nous pensons que les nuages de points sont une représentation intéressante pour la gestion des objets déformables. Ainsi, le problème de l'acquisition ou de l'enrichissement du modèle au cours du suivi pourrait bénéficier d'une représentation qui ne maintient pas explicitement d'information topologique (on entend par là des relations de voisinages codés par un maillage d'arêtes ou de facettes).

Ces deux procédures, lancer de rayon et projection sont maintenant classiques lorsque le modèle est un maillage de facettes triangulaires ou une surface définie de manière analytique (surface de Bézier, surface implicite, etc.), et peuvent être considérées comme des techniques élémentaires. Ce n'est pas le cas lorsque le modèle est un nuage de points, même si ces représentations sont depuis une dizaine d'années extensivement étudiées par les communautés de modélisation géométrique et de synthèse d'images.

Ce chapitre est une présentation des géométries par points dans leur contexte original.

[1] Nous utiliserons dans ce document les termes génériques de *géométries à base de points*, de *représentations à base de points* ou encore de *nuages de points* pour désigner les modèles constitués de points définissant un objet 3D.

L'accent est mis sur les outils et techniques que nous adapterons dans le cadre du suivi visuel, qui impose des contraintes propres. Ainsi, si les nuages de points laissent envisager la possibilité de manipuler des objets (géométriques) très complexes, nous les confrontons à une «réalité» acquise par des caméras numériques standard, de résolution faible en comparaison du rendu haute qualité recherché pour la synthèse d'images. Les modèles que nous considérerons auront donc une précision limitée, en deçà de ce que suggère l'état de l'art de techniques d'acquisition (près d'un milliard de points) et du rendu (quelques dizaines de millions de points par seconde). Par ailleurs, les manipulations interactives de l'objet sont des problématiques annexes dans le cadre du suivi vidéo, où les déformations de l'objet sont essentiellement contraintes par le flux vidéo, et non par un opérateur.

Le chapitre s'articule comme suit : nous présentons d'abord les différentes représentations associées à la notion de «nuage de points». Nous montrons comment ces représentations définissent la surface sous-jacente de l'objet (section 4.1). Ensuite nous détaillons les différentes étapes d'une chaîne de traitement des géométries par points, de l'acquisition au rendu (section 4.2). Le rendu se verra accorder un développement particulier (section 4.3) car elle nous fournira la base des techniques utilisées au chapitre 5.

4.1 Les représentations à base de points

4.1.1 Historique et motivation

Systèmes de particules L'idée d'utiliser des points pour la représentation, manipulation et le rendu des objets géométriques n'est pas récente. Les points ont été initialement privilégiés pour les représentations volumiques de phénomènes naturels notamment (fumée, feu, nuage, etc.). Les objets naturellement «particulaires» (les liquides, les fluides, le sable, etc.) se prêtent bien à des représentations ne décrivant pas explicitement l'information topologique. Puisqu'il n'est pas nécessaire de maintenir cette information, il est en effet aisé de traiter les caractéristiques dynamiques de ces objets. Les systèmes de particules introduits par Reeves [Ree83], sont un exemple classique de telles représentations. Une topologie est alors calculée localement afin de définir les interactions entre une particule et ses voisines. Les représentations par points sont aussi utilisées conjointement à un maillage dans les systèmes masses-ressorts, utilisés par exemple pour la modélisation et l'animation des vêtements [RC02].

Représentation par maillage Historiquement les représentations continues ont été privilégiées par rapport aux représentations à base de points pour la modélisation de la surface des objets. Pour le rendu en particulier, les représentations à base de facettes sont largement les plus utilisées. Projeter et discrétiser des triangles est relativement aisé et efficace grâce à l'existence d'algorithmes de type *scanline*. Par ailleurs, les bibliothèques et interfaces de programmation standards comme OpenGL [Khr07a] proposent une chaîne de rendu adaptée à ce type de modèles. La disponibilité grandissante de matériels implantant cette chaine de rendu, poussée par les besoins de l'industrie du loisir audiovisuel (jeu vidéo,

cinéma et télévision), a rendu les géométries à base de facettes incontournables dans ce domaine. Pour l'édition des objets de synthèse, d'autres types de surfaces ont été développés, plus adaptés à la manipulation interactive, notamment d'objets lisses. Citons par exemple les surfaces de Bézier (voir par exemple [Far02]), les surfaces splines [Far02] et les surfaces implicites [BBB+97]. Pour chacune de ces représentations des algorithmes de facétisation existent : méthodes d'échantillonnage pour les surfaces paramétriques et, par exemple, la méthode des *marching-cube* [LC87] pour les surfaces implicites. Ces méthodes permettent d'obtenir des maillages pour la phase de rendu.

Grâce aux processeurs graphiques modernes, il est désormais possible de visualiser en temps réel des scènes de très grande complexité. Pour augmenter la richesse visuelle des images sans augmenter la complexité géométrique des objets, on utilise la technique classique consistant à plaquer une «texture» sur le maillage. Les informations supplémentaires ainsi apportées (sous la forme de couleurs ou de normales) permettent un rendu de grande qualité même avec des modèles de faible complexité géométrique.

Cependant les modèles géométriques créés et manipulés sont de plus en plus complexes, grâce au développement de procédés d'acquisition d'objets réels plus précis et à la capacité mémoire croissante des stations de travail. Parallèlement, la taille des images générées n'a pas augmenté en proportion similaire : un écran d'une résolution standard 1600×1200 affiche moins de deux millions de pixels. Une carte graphique récente permet d'afficher plusieurs dizaines de millions de triangles par seconde. Il est alors envisageable de visualiser en temps interactif des scènes d'une finesse telle que les triangles se projettent sur moins d'un pixel, rendant l'assemblage et la discrétisation des triangles inutilement complexe. Dans ce contexte, les textures sont fortement sous-échantillonnées ; pour éviter des artefacts, un filtrage coûteux doit leur être appliqué et la technique perd son intérêt initial (c'est-à-dire ajouter efficacement des détails à une géométrie simple).

Intérêt des représentations par points De ce constat, on peut tirer l'idée de remplacer les triangles par des primitives plus simples que sont les points. Levoy et Whitted [LW85] ont les premiers proposé d'utiliser le point comme une primitive de rendu universelle : tout objet peut en effet être représenté en échantillonnant de façon suffisamment dense sa surface par des points. L'attrait des points est également important dans une optique de manipulation des modèles. En effet de nombreux algorithmes opérant sur les maillages de triangles nécessitent de conserver la cohérence topologique à toutes les étapes, ce qui les rend plus complexes et peut nécessiter des phases de «nettoyage» du maillage [ABK98] ou des techniques de re-maillage, en particulier lorsque l'objet subit des déformations [BK04].

Depuis quelques années et notamment les travaux réalisés pour la visualisation des données issues du *Digital Michaelangelo Project* [LPC+00], de nombreuses recherches ont été effectuées sur les géométries à base de points, à toutes les étapes de la chaine de traitement : simplification [PGK02], édition [ZPKG02], rendu [BHZK05, AKP+05] et animation [PKA+05]. Nous renvoyons à [KB04] pour un panorama détaillé de l'utilisation des représentations par points dans le contexte de la synthèse d'images.

4.1.2 Différentes primitives «points»

Nous présentons maintenant les différentes représentations possibles de la «primitive point» dans un ordre de complexité croissante.

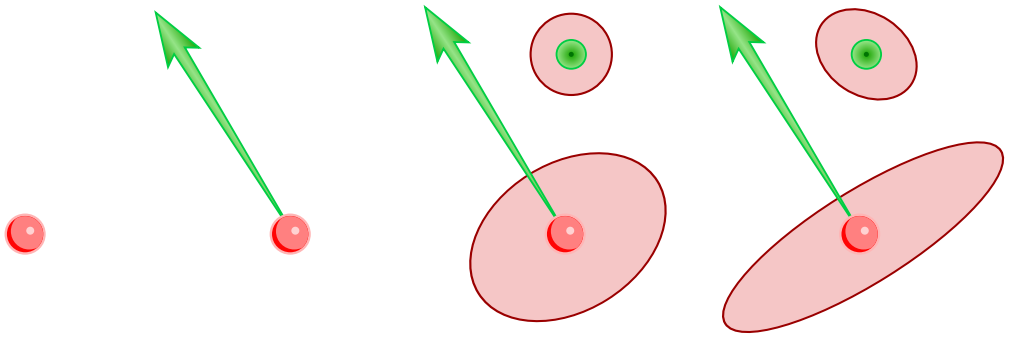


FIG. 4.1 – Différentes représentations de la primitive points ; de gauche à droite : point pur, point orienté, splat, splat non isotrope.

Un type de géométrie est désigné comme «à base de points», lorsqu'il est constitué d'un ensemble de points non organisés topologiquement, c'est-à-dire sans relation explicite de voisinage. L'ensemble des points peut être organisé en mémoire selon une structure de données particulière, de nature éventuellement géométrique, comme les *octree*, les partitions binaires (*kd-tree*) (voir par exemple [dBvKO00]) ou des structures hiérarchiques. Les hiérarchies de sphères englobantes sont aussi fréquemment utilisés par exemple dans [RL00] pour la gestion des niveaux de détails ou dans [AA03a] pour accélérer le lancer de rayon. Indépendamment de la structure de données utilisée, des mécanismes d'optimisation, comme le stockage temporaire (*cache*) d'informations topologiques locales sont fréquemment utilisées (par exemple des ensembles de points voisins).

Nous noterons de façon abstraite un modèle par points sous la forme de l'ensemble $P = \{\mathbf{p}^i\}_{i=[1,\dots,N]}$, avec $\mathbf{p}^i$ un point 3D vu comme un échantillon de surface, contenant au minimum une position et éventuellement accompagné d'autres informations (voir figure 4.1). Un nuage de points définit une surface sous-jacente notée $\mathcal{S}(P)$.

Les points purs

Ici les points $\mathbf{p}^i$ sont assimilés à leur position dans $\mathbb{R}^3$. C'est la représentation la plus compacte possible. Elle correspond généralement aux données brutes générées par les appareils d'acquisition (scanners 3D) ou par les techniques de reconstruction 3D à partir d'images. En pratique cette représentation est rapidement enrichie dès l'étape de reconstruction vers une représentation associant au point au moins la normale à la surface

sous-jacente $\mathcal{S}(P)$. En effet cette information est souvent requise, par exemple pour le calcul de l'éclairage lors du rendu. Connaître les normales simplifie également l'estimation de la courbure locale de la surface, à partir de laquelle on peut déduire la densité de points nécessaire à une reconstruction satisfaisante.

Les points orientés

Ici on oriente chaque point en fournissant une normale $\mathbf{n}^i$. On possède donc un échantillonnage de la surface sous-jacente et de sa dérivée. La normale peut-être obtenue directement lorsque le modèle par points est le produit de la conversion d'un autre type de modèle géométrique comme les surfaces implicites ou les surfaces de Bézier. En effet en dehors des éventuelles singularités on connaît dans ce cas la forme analytique de la normale en chaque point de la surface.

Lorsque le modèle «source» est un nuage de points purs, il est possible d'estimer la normale en un point à partir de son voisinage, comme nous le détaillons ici.

Dans le domaine discret, un voisinage de $\mathbf{p}^i$ est un sous-ensemble $V_i = \{\mathbf{p}^{i_1}, \ldots, \mathbf{p}^{i_k}\}$ de P constitué des points satisfaisant un certain *critère d'appartenance* au voisinage, défini par des relations spatiales entre les points. Par exemple, l'ensemble des k-plus proches voisins de $\mathbf{p}^i$ s'appuie sur un réordonnancement des points de P défini par une permutation Π de $\sigma(N)$ telle que

$$\|\mathbf{p}^{\Pi(1)} - \mathbf{p}^i\| \leqslant \|\mathbf{p}^{\Pi(2)} - \mathbf{p}^i\| \leqslant \ldots \leqslant \|\mathbf{p}^{\Pi(N)} - \mathbf{p}^i\|$$

On a alors

$$V_i^{k-pp} = \{\mathbf{p}^{\Pi(1)}, \ldots, \mathbf{p}^{\Pi(k)}\}$$

D'autres relations de voisinage peuvent être définies, par exemple l'ensemble des points appartenant à la boule $\mathcal{B}(\mathbf{p}^i, \varepsilon)$ pour une distance ε donnée. Des voisinages définis à l'aide de structures géométriques (partitions binaires, diagrammes de Voronoï) [Pau03] permettent d'obtenir un nombre réduit de voisins tout en conservant un échantillonnage isotrope du morceau de surface autour de $\mathbf{p}^i$.

L'analyse en composantes principales de V_i permet de déterminer les propriétés locales de la surface en $\mathbf{p}^i$. En effet soit $\overline{\mathbf{p}}$ le barycentre de V_i et soit C la matrice de variance-covariance définie par :

$$C = \frac{1}{k} \begin{bmatrix} \mathbf{p}^{i_1} - \overline{\mathbf{p}} \\ \vdots \\ \mathbf{p}^{i_k} - \overline{\mathbf{p}} \end{bmatrix}^T \cdot \begin{bmatrix} \mathbf{p}^{i_1} - \overline{\mathbf{p}} \\ \vdots \\ \mathbf{p}^{i_k} - \overline{\mathbf{p}} \end{bmatrix}$$

Alors le vecteur propre associé à la plus petite valeur propre C définit une approximation de la normale au point $\mathbf{p}^i$. La qualité de cette estimation dépend de la densité d'échantillonnage du nuage vis à vis de la taille caractéristique des détails du modèle. Cette taille peut se définir comme le rayon h de la plus grande sphère qui intersecte la surface en une composante connexe au plus.

Les normales ainsi estimées sont déterminées au signe près et donc peuvent avoir des orientations incohérentes et ne pas refléter la continuité $\mathcal{C}^1$ de la surface $\mathcal{S}(P)$. Hoppe

[HDD+92] propose une procédure pour lever l'ambiguïté sur l'orientation de la surface. Elle consiste à parcourir un graphe de couverture minimale du nuage, en orientant la normale de chaque point de sorte que l'angle entre deux points adjacents soit inférieur à $\frac{\pi}{2}$. Le point de départ peut être choisi comme le point ayant une coordonnée arbitraire maximale ; il est alors orienté dans la direction opposée au barycentre du nuage P. Lorsque la surface est orientable et lorsque l'échantillonnage est suffisamment dense, cette propagation conduit à une orientation cohérente pour l'ensemble des normales du nuage.

Par ailleurs les deux autres valeurs propres de C définissent un espace orthogonal à $Vect(\mathbf{n}^i)$ dans $\mathbb{R}^3$ et ainsi permettent de définir un plan T_i localement tangent au nuage (voir la figure 4.2).

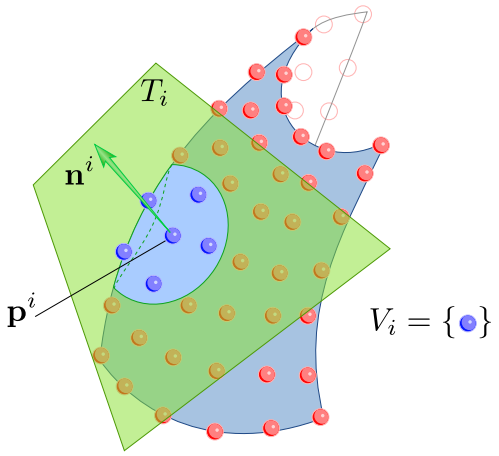


FIG. 4.2 – Analyse de la covariance du voisinage pour déterminer les propriétés différentielles locales de la surface (normale et plan tangent).

La connaissance de la normale n'est pas toujours suffisante, notamment pour la visualisation par projection des points sur le plan image. Pour garantir une image "sans trou" quelle que soit la résolution et la densité du nuage, il est nécessaire de connaître la distance d'un point à ses voisins. Cette distance peut-être définie globalement dans le cas où la surface est échantillonnée de manière homogène. Cependant un échantillonnage homogène est difficile à conserver lors des traitements locaux du nuage. Il est de plus inefficace : les zones de faible courbure ont la même densité de points que les zones de forte courbure. On peut améliorer le compromis entre le nombre de points et l'erreur d'approximation en associant un rayon à chaque point. Ce type de structure, proposé initialement par Zwicker [ZPvBG01] est appelé *splat* dans la littérature.

Les *splats*

Un *splat* est un point orientés auquel est associé le rayon d'un disque localement tangent à la surface (voir figure 4.1). Pour couvrir l'espace entre les points lors de la visualisation, le rayon doit correspondre à la distance du point à ses voisins. Une détermination précise est délicate, par exemple la présence d'un trou considéré comme une caractéristique de l'objet à préserver ne doit pas influencer la taille des splats de son bord. À partir d'un maillage une méthode possible consiste à définir le rayon d'un point comme la demi-longueur de la plus grande arête reliée au point. La surface $\mathcal{S}(P)$ est alors approximée par l'ensemble des disques, ce qui donne une approximation linéaire par morceaux, tout comme un maillage triangulaire.

Pour prendre en compte l'anisotropie locale de la courbure de la surface, des *splats* elliptiques peuvent être défini [BHZK05]. Le plan tangent T_i est paramétré par

$$T_i = \{\mathbf{x} \in \mathbb{R}^3, \mathbf{x} = \mathbf{p}^i + x_u.\mathbf{u_i} + x_v.\mathbf{v_i}, (x_u, y_v) \in \mathbb{R}^2\}$$

où $\mathbf{u_i}$ et $\mathbf{v_i}$ correspondent aux axes de courbure principaux de la surface.

On peut les approcher par une analyse statistique du nuage des normales d'un voisinage du point $\mathbf{p}^i$, de manière similaire à la méthode présentée plus haut. Le splat est alors défini par l'ellipse d'axes $\mathbf{u_i}$ et $\mathbf{v_i}$ et de grand rayon (resp. petit rayon) inversement proportionnel à la courbure minimale (resp. maximale) ([Gar99], p.69). Les vecteurs $\mathbf{u_i}$ et $\mathbf{v_i}$ définissent également un plan T_i' qui approxime le plan tangent à la surface.

On peut représenter le *splat* par la donnée du triplet $\{\mathbf{p}^i, \mathbf{u_i'}, \mathbf{v_i'}\}$ où $\mathbf{u_i'}$ et $\mathbf{v_i'}$ sont les axes de l'ellipse mis à l'échelle en fonction du rayon de l'ellipse, de sorte qu'un point $\mathbf{q}$ appartient au *splat* s'il satisfait :

$$(\mathbf{u_i'}^T.(\mathbf{q} - \mathbf{p}^i))^2 + (\mathbf{v_i'}^T.(\mathbf{q} - \mathbf{p}^i))^2 \leqslant 1$$

Les splats elliptiques fournissent la meilleure approximation linéaire locale à la surface, en particulier ils ont un pouvoir d'approximation supérieur aux triangles. Un nuage de splats elliptiques est cependant plus facile à manipuler qu'un maillage de triangle, puisqu'il n'est pas nécessaire de conserver la continuité lors des traitements.

Il est possible de représenter les discontinuités comme les arêtes et les coins en utilisant des splats découpés selon une ou deux droites définies dans le plan du splat [ZRB+04] (voir figure 4.3).

4.1.3 Reconstruction d'une surface continue à partir d'un nuage de points

Une question fondamentale liée à l'utilisation des modèles géométriques à base de points est celle de la reconstruction d'une surface continue. Cette étape est nécesaire d'une part pour l'édition du modèle (par exemple pour le raffinage par ajout de points) et d'autre part pour obtenir un rendu sans trou. Une des approches possibles est de définir une

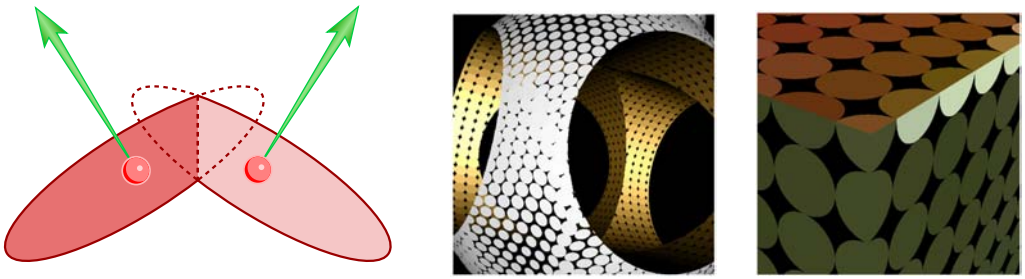


FIG. 4.3 – Rendu des discontinuités par découpe des splats. Les images de droites sont issues de [ZRB+04]

surface implicite à partir du nuage de points : la surface $\mathcal{S}$ reconstruite est alors définie comme l'image réciproque par une fonction globale de potentiel f d'une valeur donnée (0 par convention), c'est-à-dire $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^3, f(\mathbf{x}) = 0\}$. Reuter et al. [RTSD03] proposent une représentation s'appuyant sur les fonctions à base radiale de Wendland. Le degré de continuité de la surface reconstruite dépend de la forme de la fonction choisie. Cette surface permet ensuite de générer, en temps intéractif, de nouveaux points pour le rendu et la manipulation de la surface.

Dans cette section, nous présentons un autre formalisme permettant de définir une surface continue à partir d'un ensemble de points, s'appuyant sur la méthode d'approximation fonctionnelle MLS. Cette méthode a été initialement proposée par Lancaster [LS81] dans le contexte de l'approximation ou de l'interpolation de données fonctionnelles dans $\mathbb{R}^d$. Tout comme les méthodes utilisant des fonctions à base radiale, la définition de l'approximation MLS s'appuie sur le voisinage local et ne nécessite pas de topologie explicite, ce qui la rend intéressante pour les géométries à base de points.

Dans le contexte fonctionnel, l'approximation MLS $\widetilde{f}$ d'une fonction f est définie comme suit. Soit une fonction $f : \mathbb{R}^d \mapsto \mathbb{R}^n$ et soit un ensemble $D = \{(\mathbf{x_i}, f_i),\ f_i = f(\mathbf{x_i})\}$ de points d'échantillonnage ; en chaque point $\mathbf{x_i}$ de $\mathbb{R}^d$ est associée la valeur f_i de la fonction à reconstruire. L'approximation MLS de $f(\mathbf{x})$ de degré m est donnée par la valeur $\widetilde{p}(\mathbf{x})$ du polynôme $\widetilde{p}$ de degré m défini par :

$$\widetilde{p} = \underset{p \in P^m[\mathbb{R}^d]}{\operatorname{argmin}} \sum_i (p(\mathbf{x_i}) - f_i)^2\ \Phi(\mathbf{x_i}, \mathbf{x}). \tag{4.1}$$

La fonction Φ est une fonction de pondération dépendant uniquement de la distance euclidienne entre les points arguments, c'est-à-dire $\Phi(\mathbf{x_i}, \mathbf{x}) = \phi(\|\mathbf{x_i} - \mathbf{x}\|)$, où $\phi : \mathbb{R}^+ \mapsto \mathbb{R}^+$ est une fonction de classe $\mathcal{C}^k$, positive et monotone décroissante. La fonction $\widetilde{f}$ est donc définie par :

$$\widetilde{f} : \mathbf{x} \to \widetilde{p}(\mathbf{x}) \tag{4.2}$$

Si de plus $\lim_{x\to 0}\phi(x) = \infty$, alors la reconstruction $\widetilde{f}$ MLS est interpolante, c'est-à-dire $\forall i, \widetilde{f}(\mathbf{x_i}) = f_i$. L'approximation peut être rendue locale si ϕ décroît rapidement vers 0 lorsque x tend vers l'infini, en particulier lorsque ϕ est à support compact. Le degré de continuité de l'approximation ainsi obtenue dépend de la continuité de la fonction ϕ : si ϕ est de classe $\mathcal{C}^k$ alors $\widetilde{f}$ est aussi de classe $\mathcal{C}^k$.

L'adaptation de cette technique à l'approximation de surfaces (plus précisément des variétés de dimension 2) est problématique car il n'existe généralement pas de référentiel global permettant de définir une paramétrisation fonctionnelle de la surface. Levin a proposé un algorithme itératif permettant l'évaluation d'un opérateur de projection d'un point sur la surface MLS [Lev03]. La surface est alors définie par l'ensemble des points fixes de l'opérateur Ψ_P, c'est-à-dire :

$$\mathcal{S}(P) = \{\mathbf{x} \in \mathbb{R}^3, \Psi_P(\mathbf{x}) = \mathbf{x}\}. \tag{4.3}$$

L'évaluation de l'opérateur de projection en un point $\mathbf{x}$ proche de la surface comporte trois étapes : le calcul d'un repère local (1) dans lequel on estimera une approximation polynômiale locale de la surface (2) ; enfin l'évaluation de la projection d'un point sur la surface et de la normale en cette projection (3). Ces trois étapes sont détaillées ci-dessous.

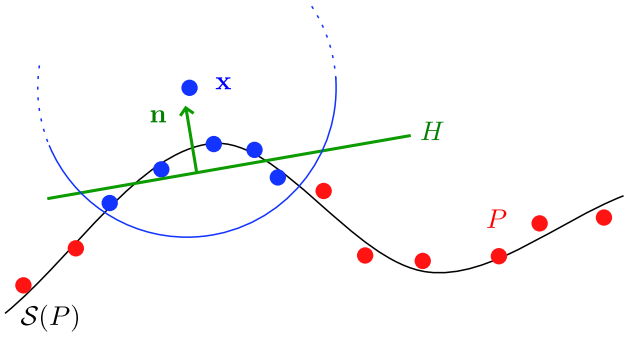


FIG. 4.4 – Détermination du plan de référence local.

(1) Calcul du repère local (voir figure 4.4). Un plan d'un repère de référence local H est d'abord déterminé :

$$H = \{\, \mathbf{y} \in \mathbb{R}^3,\ \mathbf{y}^T.\mathbf{n} - d = 0 \,\},\ \mathbf{n} \in \mathbb{R}^3,\ \|n\| = 1,\ d \in \mathbb{R} \tag{4.4}$$

Pour cela on cherche $\mathbf{n} \in \mathbb{R}^3$ et $d \in \mathbb{R}$ qui minimisent la somme de carrés pondérés :

$$\sum_i (\mathbf{p}^{i\top}.\mathbf{n} - d)^2.\phi(\|\mathbf{p}^i - \mathbf{q}\|) \tag{4.5}$$

où $\mathbf{q}$ est le projeté de $\mathbf{x}$ sur H (voir figure 4.5). Le plan de référence définit un repère local dans lequel un point $\mathbf{p}^i$ du nuage P a pour coordonnées (u_i, v_i, f_i). Le plan H peut

être considéré comme l'espace des paramètres de la donnée fonctionnelle f_i (*hauteur* du point $\mathbf{p}^i$).

Il est important de noter que les poids sont calculés relativement à $\mathbf{q}$ et non à $\mathbf{x}$. Cette condition est cruciale pour que la procédure soit un opérateur de projection.

(2) Approximation locale par un polynôme. Une fois le plan de référence local déterminé, le problème se formule comme le calcul de l'approximation MLS dans le cadre fonctionnel. Il s'agit donc de calculer le polynôme $\widehat{p}$ qui minimise :

$$\widehat{p} = \underset{p \in P^m[\mathbb{R}^2]}{\operatorname{argmin}} \sum_i (p(u_i, v_i) - f_i)^2 \, \phi(\|\mathbf{p}^i - \mathbf{q}\|). \tag{4.6}$$

(3) Évaluation de la projection de $\mathbf{x}$. La projection de x sur la surface est alors définie par :

$$\Psi_P(\mathbf{x}) = \mathbf{q} + \widehat{p}(0,0).\mathbf{n}. \tag{4.7}$$

La normale à la surface MLS est alors définie grâce aux dérivées $\frac{\partial \widehat{p}}{\partial u}(0,0)$ et $\frac{\partial \widehat{p}}{\partial v}(0,0)$ du polynôme $\widehat{p}$.

La définition MLS de la surface sous-jacente d'un nuage de points permet d'obtenir une reconstruction aussi fine que souhaitée, puisque chaque point de l'espace permet d'obtenir un point sur la surface par projection. Le rendu par lancer de rayons présenté en section 4.3.1 profite de cette propriété pour produire un résultat de grande qualité. Une fois la représentation MLS connue, il est possible d'obtenir directement les propriétés locales de la surface, comme la courbure ou le plan tangent, à partir du polynôme approximant $\widehat{p}$. L'inconvénient principal des surfaces MLS est le coût de calcul, notamment du plan de référence, qui limite l'utilisation de cette représentation dans les applications interactives ou temps-réel. Dans notre algorithme de suivi, nous utilisons une procédure de lancer de rayons pour les surfaces MLS, mais pour des calculs effectués hors-ligne (voir le chapitre 5).

4.2 Manipulations des géométries à base de points

Une des motivations principales pour définir des manipulations sur les surfaces de points est le développement récent de dispositifs d'acquisition de modèle produisant un nombre d'échantillons de la surface très important [LPC+00]. Ces données sont évidemment bruitées et peuvent comporter des trous. Définir un maillage à partir de tels ensembles de points s'avère délicat car la topologie sous-jacente est ambiguë. Par ailleurs le temps de calcul nécessaire à la génération du maillage et le stockage de la topologie deviennent importants.

Il est alors intéressant de définir une chaîne de traitement qui ne nécessite aucune triangulation, depuis la reconstruction de la surface, jusqu'au rendu, selon le schéma de la figure 4.6.

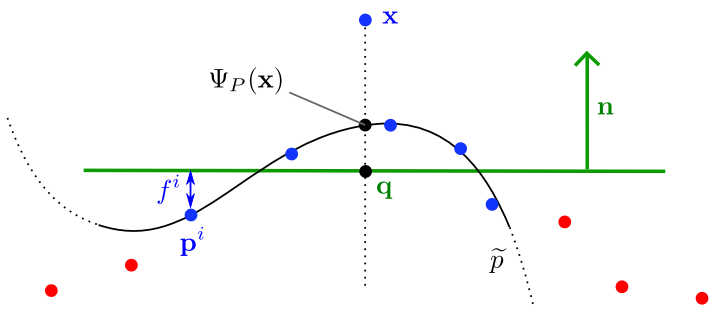


FIG. 4.5 – Polynôme approximant localement le nuage de point

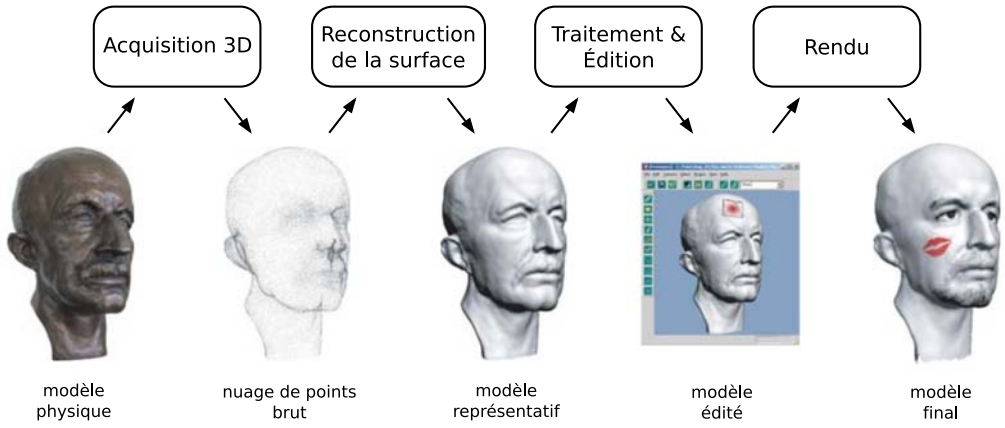


FIG. 4.6 – Chaîne de traitement pour les géométries par points. Traduction d'un schéma issu de [Pau03]

.

4.3 Rendu des modèles à base de points

Toute la difficulté du rendu des géométries à base de points réside dans la production d'une image continue, autrement dit sans trou, à partir d'une représentation discrète. L'approche à ce jour la plus classique pour le rendu des objets échantillonnés par des points (par exemple issus d'un dispositif d'acquisition) consiste à construire un maillage, dans une étape préalable de triangulation, et à représenter ce maillage. La continuité étant préservée par la projection, on obtient une image continue.

Lorsque le nuage de points est suffisamment dense au regard de la résolution de l'image, une solution simple consiste à représenter chaque point par un pixel image. On peut également dessiner les points comme des petits carrés ou des petites ellipses, et adapter la taille de ces primitives de base dans l'espace image en fonction du niveau de zoom, afin d'obtenir une représentation continue. Ceci est relativement facile à mettre en oeuvre, et peut suffire pour des applications de prévisualisation rapide, comme celles décrites dans [LPC+00] et [RHHL02]. L'algorithme *QSplat* [RL00] décrit la sélection automatique d'un niveau de détails adapté, en fonction du niveau de zoom courant. À partir d'un nuage de *splats* (c'est-à-dire position, normale et rayon), on construit une hiérarchie de sphères englobantes : les feuilles de l'arbre sont constitutées des sphères centrées en les points du nuage et de rayon celui du splat. Chaque niveau est ensuite construit récursivement : on partage le nuage en deux et on construit une sphère englobant les sphères englobantes construites pour les deux sous-nuages. Chaque sphère conserve des informations interpolées depuis ses sphères filles, comme la normale ou la couleur. La première ligne de la figure 4.7 montre plusieurs niveaux de détails. Un même niveau de qualité est obtenu en adaptant le niveau de détails de sorte que chaque splat soit représentée par une primitive de taille inférieure à une taille fixée. Pour les applications de visualisation interactives, la sélection du niveau de détails peut être intégrée à la boucle de rendu pour garantir par exemple un rafraîchissement d'affichage minimum désiré.

La qualité de ce type de rendu est cependant médiocre, puisque chaque élément de surface est représenté de manière grossière. Le rendu de haute qualité des géométries par points a été abordé selon deux approches : le lancer de rayons et la projection de noyaux de reconstruction (en anglais, cette technique est appelée *splatting*).

Les techniques de lancer de rayons exploitent la possibilité de définir une surface sous-jacente au nuage (*Point-Set Surfaces*, dont les surfaces MLS sont une instance). Cette technique offre la meilleure qualité d'image, et permet le rendu d'effets optiques complexes tels que réflexion et réfraction. Les techniques de lancer de rayons sont cependant très coûteuses : dans [AA03a], les auteurs évoquent "plusieurs heures" de calculs. Dans [AA03b], les mêmes auteurs rapportent quelques dizaines de secondes pour de petites images (la même approche est utilisée dans [AKP+05] pour le rendu de surface déformable). Les résultats que nous obtenons avec notre mise en œuvre de ces méthodes sont de cet ordre de grandeur (voir section 4.3.1). Quelques travaux proposent des techniques plus performantes. L'approche hautement optimisée proposée dans [WS05] permet un rendu interactif. Récemment, Guennebaud et Gross [GG07] ont proposé un nouveau modèle de surface pour les nuages de points, et ont développés une mise en œuvre de l'opérateur de projection sur le GPU. Ils

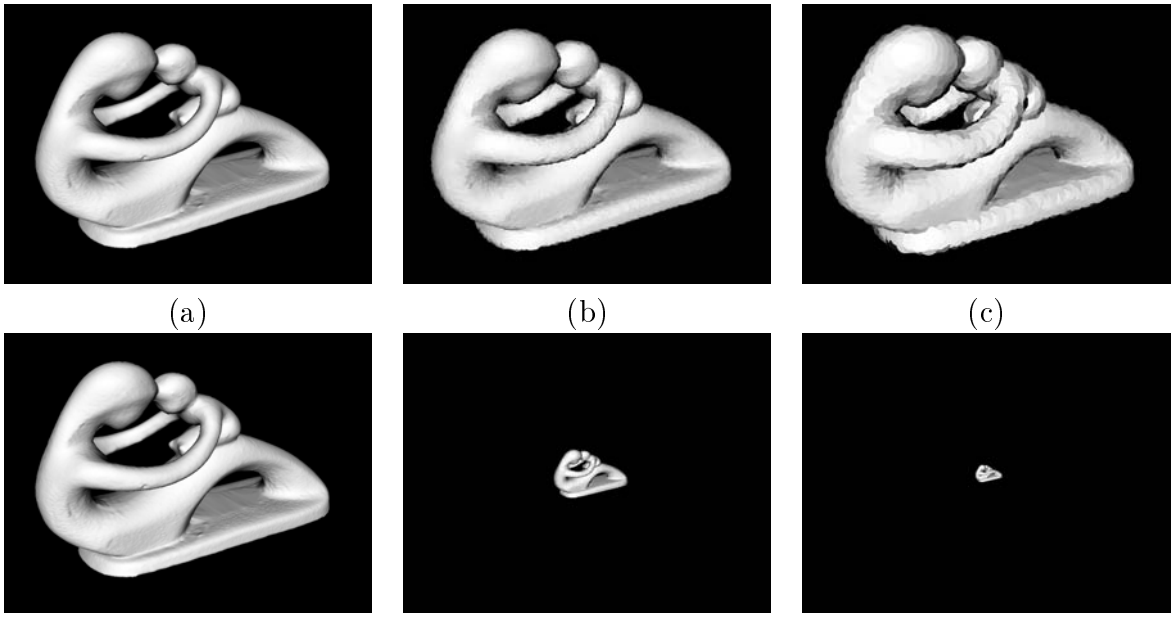


FIG. 4.7 – Gestion du niveau de détails grâce à une hiérarchie de sphères englobantes. La première ligne montre un objet à différents niveaux de détails. Si la taille apparente d'une sphère est inférieure à un seuil, elle est affichée (par un disque parallèle au plan image), sinon ses sphères filles sont considérées, récursivement. Le nombre de sphères dessinées est 241 604 (a), 26 301 (b) et 6 593 (c). La deuxième ligne montre les mêmes niveaux de détails, mais à un niveau de zoom pour lequel la qualité du rendu est satisfaisante.

annoncent 40 millions de projections par seconde. On peut ainsi envisager une adaptation au lancer de rayons qui conduise à un rendu en temps interactif. Au delà de ces progrès, notre contexte particulier nous autorise malgré tout à nous intéresser à cette technique. En effet notre objectif n'est pas le rendu de la totalité d'une image, mais le calcul des antécédents 3D des points d'intérêt calculés dans les images. Ce nombre est relativement petit : quelques centaines de points sur la surface doivent être déterminés, à mettre en regard des quelques 300 000 pixels d'une image en 640×480. Par ailleurs, contrairement aux algorithmes de rendu, nous n'avons pas besoin de la normale à la surface. De même, nous ne calculons que les rayons primaires (ceux issus du centre de projection), puisque nous ne cherchons pas le rendu des effets optiques (réflexion, réfraction, ombres dures, etc.).

4.3.1 Approche par lancer de rayons

Nous présentons ici une méthode de rendu d'un nuage de points s'appuyant sur les surfaces MLS présentées précédemment et une procédure de lancer de rayons introduite par Adamson et Alexa [AA03a].

Intersection d'un rayon et d'une surface MLS

L'idée est d'exploiter l'opérateur de projection Ψ_P définissant la surface MLS $\mathcal{S}_P$ (voir équation 4.7 en section 4.1.3). En effet le calcul de cet opérateur implique la détermination d'un polynôme approximant localement la surface. Les intersections successives du rayon avec ces polynômes permettent d'avancer le long du rayon jusqu'à l'intersection avec $\mathcal{S}_P$. La figure 4.8 montre deux itérations de la procédure, à partir d'un point initial $\mathbf{r}_0$ sur le rayon à intersecter.

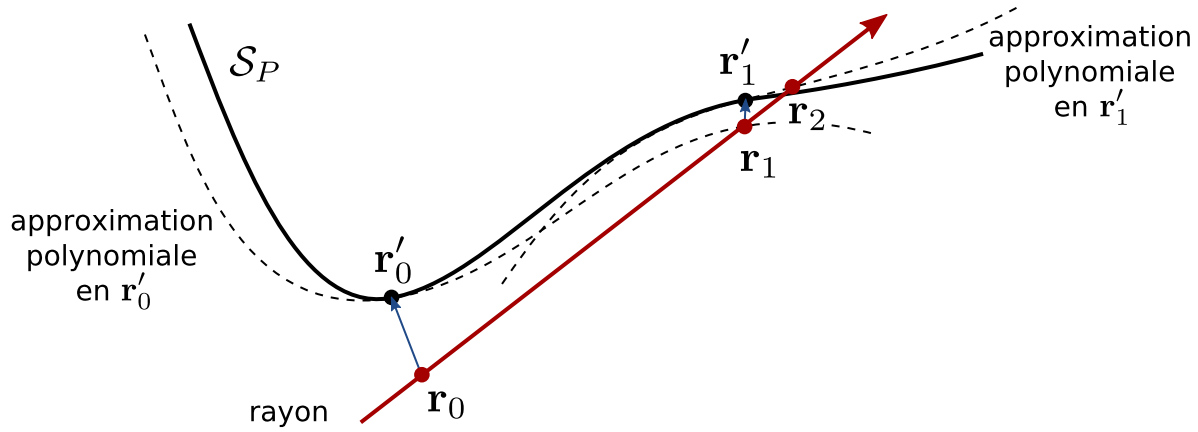


FIG. 4.8 – Convergence vers la surface MLS par intersections successives avec les approximations polynomiales locales.

A l'étape i, le point $\mathbf{r}_i$ est projeté sur la surface en $\mathbf{r}'_i = \Psi_P(\mathbf{r}_i)$, ce qui fournit également une approximation polynômiale de la surface (représentée en pointillé sur la figure) valide dans le voisinage de $\mathbf{r}'_i$. Le point $\mathbf{r}_{i+1}$ est alors obtenu par intersection du rayon avec ce polynôme. On répète la procédure jusqu'à ce qu'une des conditions suivantes soit satisfaite :

- le rayon n'a pas d'intersection avec le polynôme courant, ou bien cette intersection n'est pas dans le voisinage de $\mathbf{r}'_i$. Dans ce cas le rayon n'intersecte pas la surface.
- $\mathbf{r}_i$ est suffisamment proche de sa projection $\mathbf{r}'_i$, c'est-à-dire $d(\mathbf{r}_i) = ||\mathbf{r}'_i - \mathbf{r}_i|| < \varepsilon$. L'intersection du rayon avec la surface est donnée par le point $\mathbf{r}'_i$

L'opérateur de projection Ψ_P ne projette sur la surface $\mathcal{S}_P$ que les points proches, par conséquent la procédure itérative décrite précedemment doit partir d'un point $\mathbf{r}_0$ proche de la surface. Pour obtenir un tel point, on construit un ensemble de sphères $\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ centrées en les points du nuage et de taille suffisante pour garantir l'absence de trous, c'est-à-dire $\mathcal{S}_P \subseteq \bigcup_{k=0}^{n} \mathcal{B}_k$. Le point $\mathbf{r}_0$ peut-être alors être choisi comme l'intersection du rayon avec une sphère $\mathcal{B}_k$.

Cependant, comme le montre la figure 4.9, un rayon intersecte généralement plusieurs sphères, conduisant à plusieurs points initiaux ; la plupart ne permettent pas à la procédure interactive décrite plus haut de converger vers un point de la surface. Pour chaque rayon lancé, on établit donc la liste des sphères touchées par le rayon. En partant de la sphère la plus proche du point de vue, on recherche l'éventuelle intersection rayon-surface avec comme point initial l'intersection du rayon avec cette sphère. En cas d'échec, on relance la recherche avec la sphère suivante dans la liste.

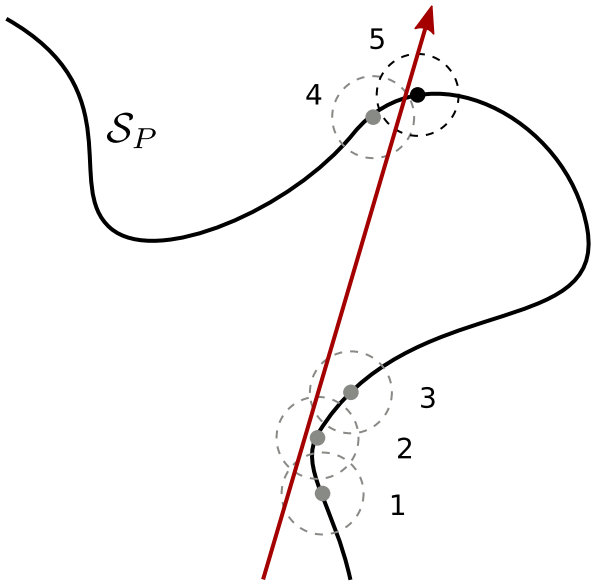


FIG. 4.9 – Intersections d'un rayon avec l'ensemble des sphères. Les chiffres représentent l'ordre de parcours des sphères.

Quelques détails de mise en œuvre permettent de rendre la procédure plus rapide. Tout

d'abord, pour minimiser le nombre d'intersections rayon-sphère, on utilise une hiérarchie de sphères englobantes similaire à celle décrite en introduction de cette section. Les sphères $\mathcal{B}_k$ constituent les feuilles de la hiérarchie.
Ensuite, la partie la plus coûteuse de l'algorithme est le calcul des projections MLS $\Psi_P(\mathbf{r}_i)$ et des approximations polynomiales locales. Pour minimiser le nombre de projection à effectuer le point $\mathbf{r}_0$ peut être remplacé par le centre de la sphère $\mathcal{B}_k$ courante. Il est alors possible de sauvegarder les coefficients de l'approximation polynômiale dans les données associées à la sphère et ainsi d'éviter le calcul de la première projection pour les rayons voisins (plus précisément ceux qui intersectent la même sphère $\mathcal{B}_k$).
Enfin, les approximations locales sont des polynômes de degré 2, ce qui simplifie le calcul des intersections avec le rayon (qui correspond à la recherche des racines d'un polynôme).

Images obtenues

Nous avons implanté la méthode décrite précédemment et l'avons intégrée dans une application permettant le rendu d'un objet défini par un nuage de points 3D. Le système construit les structures de données accélératrices présentées plus haut et effectue un rendu par lancer de rayons. La fonction de pondération ϕ utilisée pour calculer les repères de référence locaux (voir l'équation 4.5) est la gaussienne

$$\phi(x) = e^{-\frac{x^2}{h^2}}, \tag{4.8}$$

dont le paramètre h détermine le degré de lissage de la reconstruction MLS. Le support de ϕ est tronqué aux valeurs significatives, de sorte que les sommes 4.5 et 4.6 sont finies.

Puisque la technique permet d'intersecter un rayon quelconque avec le modèle, il est possible d'obtenir des rendus très réalistes, incorporant par exemple des ombres portées. Les zones d'ombre sont déterminées en lançant un rayon secondaire depuis l'intersection du rayon primaire avec l'objet vers la source lumineuse. La figure 4.10 montre deux exemples de rendus haute qualité. Plusieurs heures de calculs sont nécessaires pour générer ces images (la taille originale est 1024×768 pixels, nous les avons découpées pour les besoins de la mise en page).

Utilisation dans le cadre du suivi

Les temps de calcul de l'approche par lancer de rayons limitent son utilisation dans le cadre du rendu temps-réel. En effet, reprojeter ne serait-ce que quelques centaines de points nécessite plusieurs secondes. Cependant notre algorithme de suivi (présenté au chapitre 5) utilise également des informations acquises hors ligne sous la forme d'images clés dans lesquelles le modèle est référencé et des points d'intérêt sont reprojetés sur le modèle. Pour cette reprojection nous utilisons l'algorithme de lancer de rayons décrit précédemment, car il fournit une reconstruction précise et non dépendante du point de vue. Les détails concernant l'utilisation du lancer de rayons dans notre système de suivi sont présentés au chapitre 5.

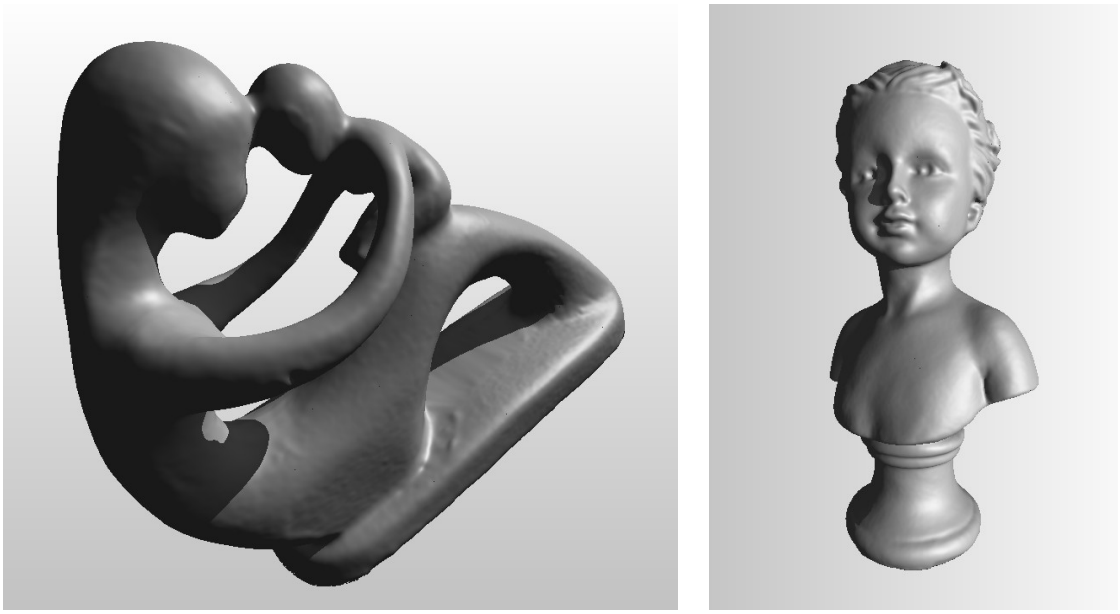


FIG. 4.10 – Deux autres rendu de surface MLS par lancer de rayons, utilisant un modèle d'éclairage plus complexe (prise en compte des spécularités et simulation des ombres portées).

4.3.2 Reconstruction par projection

Une autre approche pour reconstruire une surface continue à partir d'un nuage de points consiste à associer à chaque point une primitive «plus large» comme une sphère, une ellipse ou un disque afin de remplir les trous lors de la projection de ces primitives. Nous présentons ici l'approche que nous avons suivi, à partir de la représentation par splats circulaires définie en section 4.1.2. La technique de reconstruction de surfaces définies par splats qui fait référence à ce jour a été introduite et développée par Zwicker [ZPBG02, ZRB+04]. L'idée est d'associer à chaque splat un noyau de reconstruction local défini dans le plan du splat, qui coïncide avec le plan localement tangent à la surface. Ces noyaux servent ensuite à reconstruire les attributs de la surface (position, normale, couleur, etc.) à partir de leurs valeurs en chaque point. Les attributs d'un splat sont approchés par des fonctions définies dans un repère attaché au splat. La reconstruction de la surface nécessite une paramétrisation globale du nuage de points dans laquelle exprimer le mélange des noyaux de reconstruction locaux. Dans le cadre du rendu, cette paramétrisation est naturellement définie par l'espace 2D de l'image calculée. Les noyaux de reconstruction locaux sont ainsi projetés dans l'image, d'où l'appelation de rendu par *splatting* (c'est-à-dire par *écrasement*).

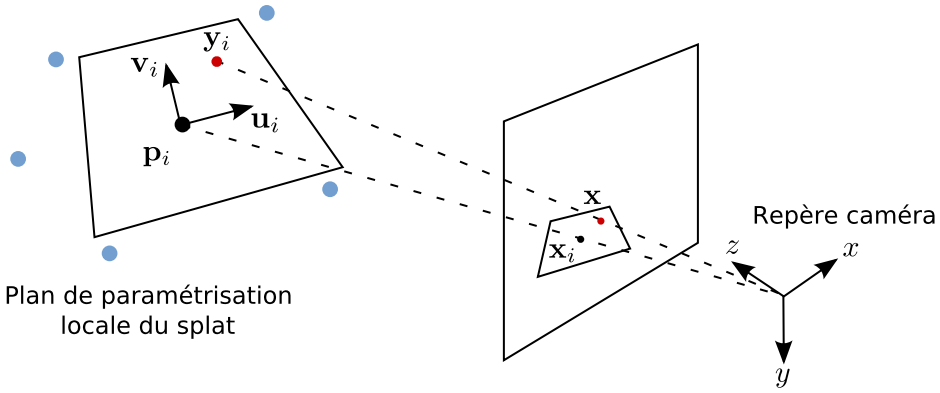


FIG. 4.11 – Espace de paramétrisation local autour d'un splat $\mathbf{p}_i$ et projection dans le plan image.

Expression de la surface reconstruite

La surface $\mathcal{S}$ (inconnue) est définie en un splat $\mathbf{p}_i$ par un ensemble d'attributs $\{A_1^i, A_2^i, \ldots\}$. La paramétrisation locale du plan du splat est donnée par deux vecteurs $(\mathbf{u}_i, \mathbf{v}_i)$, comme illustrée par la figure 4.11. Un attribut A est approché en un point de coordonnées locales $\mathbf{y}_i = (y_{i,u}, y_{i,v})$ par une fonction approximante $\mathcal{P}_i^A : \mathbb{R}^2 \to \mathbb{R}^{n_A}$ définie dans le plan du splat (n_A est la dimension de l'espace de définition de l'attribut, $n_A = 1$ pour un attribut scalaire). Cette fonction peut être par exemple un polynome obtenu par ajustement des valeurs A^j de l'attribut en les splats du voisinage de $\mathbf{p}_i$.

On définit ensuite les noyaux de reconstruction r_i, qui dépendent uniquement de la densité locale d'échantillonnage en $\mathbf{p}_i$. La valeur $r_i(\mathbf{y}_i)$ peut être interprétée comme le degré de confiance associé à l'approximation $\mathcal{P}_i^A(\mathbf{y}_i)$. Un choix classique est d'utiliser un noyau gaussien dont la variance est adaptée à la distance des voisins de $\mathbf{p}_i$.

On définit la transformation permettant de passer de l'espace 2D local à la paramétrisation globale par un morphisme inversible $\mathcal{M}_i : \mathbf{y}_i \to \mathbf{x}$. On peut alors exprimer les versions transformées r'_i et ${\mathcal{P}'}_i^A$ des noyaux de reconstruction et des fonctions approximantes :

$$r'_i(\mathbf{x}) = r_i(\mathcal{M}_i^{-1}(\mathbf{x})) \quad \text{et} \quad {\mathcal{P}'}_i^A(\mathbf{x}) = \mathcal{P}_i^A(\mathcal{M}_i^{-1}(\mathbf{x})). \tag{4.9}$$

Avec ces notations la reconstruction de l'attribut A dans la paramétrisation globale est définie comme la moyenne pondérée

$$\mathcal{S}^A(\mathbf{x}) = \frac{\sum_i r'_i(\mathbf{x}) {\mathcal{P}'}_i^A(\mathbf{x})}{\sum_i r'_i(\mathbf{x})}. \tag{4.10}$$

Notons que cette reconstruction n'est pas nécessairement interpolante, c'est-à-dire $\mathcal{S}^A(\mathcal{M}_i(0,0)) \neq A^i$. En pratique les supports des noyaux sont généralement tels que les centres des *splats* ne sont recouverts que par le noyau défini pour le splat. Lorsque ce n'est

pas le cas, les contributions des noyaux voisins sont très faibles par rapport à celle du noyau défini pour le splat. On peut ainsi considérer la reconstruction comme *quasi-interpolante*, et, par la suite, nous abuserons du verbe interpoler pour désigner l'action de cette reconstruction (en particulier pour le calcul des champs de mouvement denses (chapitres 5 et 6)).

On peut voir le problème de la reconstruction comme un problème de rééchantillonnage. Les splats définissent un échantillonnage (non régulier) de la surface. Dans le cas du rendu, la surface continue est rééchantillonnée selon la grille régulière des pixels de l'image (qui constitue l'espace de paramétrisation globale). Lorsque la grille de pixel n'est pas assez dense, ce rééchantillonnage ne permet pas une reconstruction exacte, ce qui se traduit par des effets d'aliassage (motifs moirés, arêtes en escalier ou «bruit» dans le cas de scènes animées). En effet, considèrant la surface $\mathcal{S}$ comme un signal continu, la théorie du signal indique que la fréquence d'échantillonnage doit être supérieure à la plus haute fréquente présente dans le signal d'origine. En pratique, pour des raisons de place mémoire et de coût de calcul, il n'est pas raisonnable d'augmenter indéfiniment la résolution de l'image calculée. L'approche théorique valide consiste à filtrer le signal original afin de supprimer les fréquences supérieures à la fréquence d'échantillonnage. Pour cela on calcule la convolution de la reconstruction $\mathcal{S}^A(\mathbf{x})$ avec un filtre passe-bas $h(x)$. Zwicker propose d'approximer ce filtrage en n'appliquant la convolution qu'aux noyaux de reconstruction, c'est-à-dire en écrivant :

$$\widetilde{\mathcal{S}}^A(\mathbf{x}) = (\mathcal{S}^A \otimes h)(\mathbf{x}) \approx \frac{\sum_i (r'_i \otimes h)(\mathbf{x}) \mathcal{P}'^A_i(\mathbf{x})}{\sum_i (r'_i \otimes h)(\mathbf{x})} = \frac{\sum_i \rho_i(\mathbf{x}) \mathcal{P}'^A_i(\mathbf{x})}{\sum_i \rho_i(\mathbf{x})}. \tag{4.11}$$

Le terme $\rho_i(\mathbf{x}) = (r'_i \otimes h)(\mathbf{x})$ correspond à un filtre de rééchantillonnage dans l'espace image, qui peut être exprimé assez simplement dans les conditions que nous détaillons ci-après.

Expression des filtres de rééchantillonnage dans l'espace image

Pour simplifier les calculs, l'attribut de position est approximé par un polynôme de degré 1 (c'est-à-dire la surface est localement approximée par le plan du splat) et les autres attributs (comme la couleur par exemple) sont approchés par des constantes, autrement dit $\forall \mathbf{y}_i \ \mathcal{P}^A_i(\mathbf{y}_i) = A^i$. Les noyaux de reconstruction r_i et le filtre passe-bas h sont définis comme des fonctions gaussiennes, car alors le filtre de rééchantillonnage dans l'espace image $\rho_i(\mathbf{x})$ peut également s'exprimer comme une gaussienne lorsque les morphismes $\mathcal{M}_i$ sont affines. Les $\mathcal{M}_i$ doivent incorporer toutes les transformations nécessaires à la projection d'un point défini dans l'espace objet en un pixel de l'image : transformation vers les coordonnées caméra, projection dans l'espace image puis transformation vers les coordonnées pixeliques. Cette chaine de transformation n'est généralement pas linéaire (à cause de la projection perspective). On utilise donc l'approximation affine locale

$$\overline{\mathcal{M}_i}(\mathbf{y_i}) = \mathbf{x}_i + J_i \mathbf{y}_i \tag{4.12}$$

de $\mathcal{M}_i$, où $\mathbf{x}_i$ est la projection de $\mathbf{p}_i$ dans l'image et

$$J_i = \frac{\partial \mathcal{M}_i}{\partial y_i}(0,0) \tag{4.13}$$

est le jacobien de $\mathcal{M}_i$ calculé au point $(0,0)$. En pratique J_i est calculé en transformant les vecteurs de base $\mathbf{u}_i$ et $\mathbf{v}_i$ dans l'espace image (voir [ZPvBG01] pour les détails).

En notant $g_V(\mathbf{x})$ la gaussienne 2D de matrice de covariance $V \in \mathbb{R}^{2\times 2}$, définie par

$$g_V(\mathbf{x}) = \frac{1}{2\pi|V|^{\frac{1}{2}}} e^{-\frac{1}{2}\mathbf{x}^T V^{-1}\mathbf{x}}, \tag{4.14}$$

on utilise $r_i(\mathbf{y_i}) = g_{R_i}(\mathbf{y}_i)$ et $h(\mathbf{x}) = g_H(\mathbf{x})$ pour respectivement le noyau de reconstruction dans l'espace source (objet) et le filtre passe-bas.

Le filtre passe-bas h a généralement une variance unitaire (c'est-à-dire $H = I$), pour s'adapter à l'échantillonnage pixelique de l'image. La variance des noyaux de reconstruction dépend quant à elle de la densité locale au niveau du splat considéré, puisque l'échantillonnage en 3D n'est pas uniforme. Si on note d_i l'espacement moyen entre le splat $\mathbf{p}_i$ et ses k plus proches voisins, on peut définir :

$$R_i = \begin{bmatrix} d_i^2 & 0 \\ 0 & d_i^2 \end{bmatrix}. \tag{4.15}$$

Dans ces conditions, la projection du noyau de reconstruction dans l'espace image est également une gaussienne :

$$r'_i(\mathbf{x}) = r_i(J_i^{-1}(\mathbf{x}) - \mathbf{x}_i) = \frac{1}{|J_i^{-1}|} g_{J_i R_i J_i^T}(\mathbf{x} - \mathbf{x}_i) = |J_i|\ g_{J_i R_i J_i^T}(\mathbf{x} - \mathbf{x}_i). \tag{4.16}$$

Enfin, le filtre de rééchantillonnage, obtenu par convolution avec h est lui encore une gaussienne :

$$\rho_i(\mathbf{x}) = (r'_i \otimes h)(\mathbf{x}) = |J_i|\ g_{J_i R_i J_i^T + H}(\mathbf{x} - \mathbf{x}_i) = |J_i|\ g_{V_{\rho_i}}(\mathbf{x} - \mathbf{x}_i), \tag{4.17}$$

avec

$$V_{\rho_i} = J_i R_i J_i^T + H. \tag{4.18}$$

Rendu

Le rendu consiste, pour chaque splat $\mathbf{p}_i$, à calculer la variance V_{ρ_i} du filtre de rééchantillonnage ρ_i. Afin de limiter le recouvrement des splats, le filtre est tronqué à un support compact, c'est-à-dire à l'ellipse $\mathcal{E} = \{\mathbf{x},\ \rho_i(\mathbf{x}) > s\}$, où s est une valeur seuil petite. Pour chaque attribut A reconstruit (autre que la position $A = p$), un tampon est utilisé pour les contributions $\rho_i(\mathbf{x})\mathcal{P}'^A_i$. Les poids $\rho_i(\mathbf{x})$ sont également accumulés dans un tampon séparé.

Naturellement, seuls les pixels des splats correspondant à une même portion de surface doivent contribuer à l'équation 4.11. Pour déterminer si un pixel donné contribue au

mélange final, on utilise un algorithme s'appuyant sur un tampon de profondeur «flou» : pour chaque pixel $\mathbf{x}_k$ de l'ellipse, la profondeur $z_{i,k} = \mathcal{P}'^{\mathbf{p}}_i(\mathbf{x}_k)$ du point reconstruit est comparée à ε près à la valeur $z(\mathbf{x}_k)$ actuellement dans le tampon de profondeur. Trois cas se présentent :

- si $|z_{i,k} - z(\mathbf{x}_k)| < \varepsilon$, le pixel est considéré comme appartenant à la même portion de surface que le pixel courant. La contribution est accumulée.
- si $z_{i,k} - z(\mathbf{x}_k) > \varepsilon$, le pixel est caché par la contribution courante, il n'est pas pris en compte.
- si $z_{i,k} - z(\mathbf{x}_k) < -\varepsilon$, le pixel est visible mais ne fait pas partie de la portion de surface déjà projetée. La contribution du pixel remplace toutes les valeurs courantes dans les tampons d'attribut.

Une fois tous les splats projetés, l'équation 4.11 est évaluée en divisant chaque tampon d'attributs par le tampon accumulant la somme des poids en chaque pixel. L'image finale est obtenue en combinant les différents attributs reconstruits en chaque pixel. Par exemple, si la couleur et la normale ont été reconstruite, on peut calculer un modèle d'éclairage de Phong en chaque pixel.

Le rendu par splatting est une méthode intermédiaire entre les approches de reconstruction entièrement dans l'espace objet (comme les surfaces MLS présentées en section 4.1.3) et les approches entièrement dans l'espace image comme QSplat. Elle a pour inconvénient de produire une reconstruction dépendante du point de vue et d'être plus sensible à la densité d'échantillonnage de la surface que les approches par lancer de rayons. En effet lorsque la densité est très faible les splats deviennent très gros, ce qui entraine des artefacts au niveau de la silouhette des objets. Cette méthode est cependant celle offrant le meilleur compromis entre qualité et rapidité. La reconstruction est lisse et le filtrage anistrope élimine les effets d'alliasage. Le fait de reconstruire les attributs de surface séparément autorise le calcul efficace de modèles d'éclairage complexes puisque celui n'est évalué qu'en les pixels visibles (ce principe de rendu est appelé «deferred shading» [GBP04]). La figure 4.12 présente deux images obtenues par rendu de modèles définis par splats.

Utilisation dans le cadre du suivi

Sur la base des travaux de Guennebaud [Gue05] et de Botsch et al. [BHZK05] notamment, nous avons mis en œuvre un algorithme de rendu par splatting fonctionnant sur les processeurs graphiques modernes. Cette mise en œuvre, détaillée au chapitre 6 est très rapide. Nous utilisons cette technique dans deux étapes de notre algorithme de suivi présenté au chapitre suivant :

- tout d'abord pour le calcul de bases de vecteurs de mouvements en tout point de l'image. Les vecteurs de base sont calculés en chaque point du nuage puis interpolés en chaque pixel par la technique de splatting,
- ensuite pour la déformation d'une image clé, utilisée pour faciliter l'appariement d'amers. La déformation correspond à un rendu dans une pose cible donnée.

(a) (b)

FIG. 4.12 – Exemples de rendu par splats. Dans (a) La couleur de chaque splat est déterminée par un modèle d'éclairage de Phong (à partir d'un matériau de couleur uniforme). Dans (b) la couleur est prédéterminée à partir d'une texture.

4.4 Conclusion

Une grande variété d'outils sont désormais disponibles pour le traitement des objets dont le modèle initial est un nuage de points. Même si la plupart des techniques présentées dans ce chapitre ont pour applications le rendu et l'édition de ce type de modèles, elles s'avèrent suffisamment générales pour être utilisées dans le cadre de la vision par ordinateur et en particulier pour le suivi visuel. Le chapitre suivant présente l'algorithme que nous avons développé pour suivre en temps réel un objet dont un modèle par points est donné.

Chapitre 5

Un cadre pour le suivi visuel 3D utilisant un nuage de points

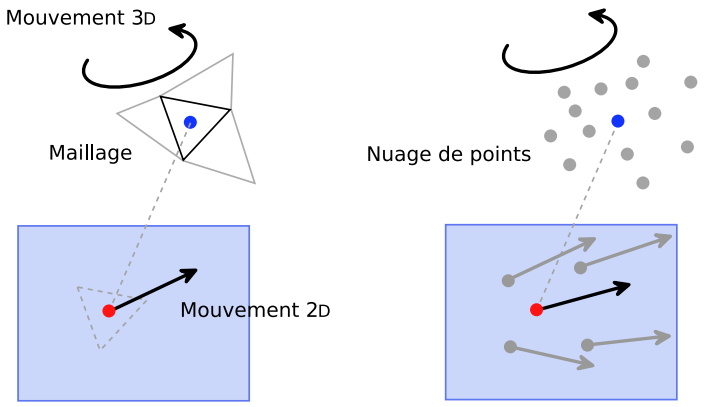


FIG. 5.1 – Inférer le mouvement 2D à partir du mouvement 3D de l'objet : à gauche, cas d'un maillage ; à droite, cas d'un nuage de points. Le modèle est représenté en gris, le point suivi dans l'image en rouge et son antécédent 3D en bleu.

Ce chapitre présente la méthode de suivi visuel 3D exploitant les nuages de points que nous avons développée. Nous avons transposé et adapté aux nuages de points l'approche utilisant un ensemble de facettes proposée par Vacchetti et al. dans [VLF04a]. Cet algorithme est reconnu comme l'un des plus efficaces de l'état de l'art. Comme nous l'avons vu au chapitre 3, il s'agit d'un suivi itératif intégrant des connaissances acquises hors ligne sous la forme d'*images clés*. Les mesures effectuées dans les images de la séquence vidéo sont les déplacements 2D de points d'intérêt, déduits d'appariements formés grâce à un critère de corrélation locale. Ces points permettent un échantillonnage des images bien adapté à la détection du mouvement apparent.

La connaissance du modèle permet d'inférer le mouvement 3D de l'objet par rapport à la caméra. L'expression du mouvement apparent d'un point de l'image en fonction du mouvement 3D de l'objet peut être obtenue par deux approches : la première consiste à considérer l'antécédent 3D, par une opération de *projection arrière*. La seconde consiste au contraire à projeter le mouvement 3D dans les images. Ce dernier «paradigme» est illustré par la figure 5.1.

Après avoir précisé nos notations, nous présenterons comment, grâce à la notion de fonction de transfert, la connaissance du modèle 3D permet de s'affranchir de la triangulation de points 3D, réduisant ainsi considérablement le nombre d'inconnues du problème. Lorsque le modèle est un maillage, deux bonnes propriétés sont exploitables : tout d'abord il est facile, à partir d'un point de l'image, de déterminer la facette sur laquelle se trouve son antécédent 3D. Ensuite la connaissance de cette facette permet d'exprimer le déplacement de ce point en fonction d'un mouvement 3D hypothétique. Ces deux propriétés n'existent pas dans le cas d'un nuage de points. Comme nous l'avons annoncé en introduction de ce document, nous défendons l'idée que les modèles utilisés pour le suivi 3D ne requièrent pas nécessairement d'informations topologiques comme celles apportées par un maillage. Nous proposons une adaptation de la technique de suivi de [VLF04a] qui conduit à un problème d'optimisation linéaire grâce à l'utilisation de l'algèbre de Lie, comme proposé dans [DC02].

Le chapitre s'organise ainsi : la section 5.1 décrit brièvement le problème du suivi déjà présenté au chapitre 3. Dans la section 5.2 nous présentons une analyse du suivi itératif utilisant un modèle de l'objet. Nous introduisons la notion importante de fonction de transfert qui nous permet, en section 5.3, de proposer une formulation du problème du suivi adaptée aux modèles par points. Ceci est en particulier possible grâce à l'utilisation d'un algorithme d'interpolation de champs de mouvement 2D par *splatting*, mis en œuvre sur le GPU. L'approche itérative (c'est-à-dire s'appuyant sur le calcul d'une mise à jour de la pose précédente pour obtenir la pose courante) est sujète au problème de dérive. En section 5.4 nous proposons une solution permettant d'éliminer ce problème, grâce à l'utilisation, conjointement au suivi itératif, d'un suivi à partir d'images clés. Nous présentons en section 5.5 les bonnes performances de cette approche hybride, en termes de précision du suivi et de rapidité d'exécution. Nous montrons enfin quelques captures issues du logiciel mettant en œuvre notre approche.

5.1 Modélisation du problème

5.1.1 Notations

Séquence d'images et poses On note la séquence d'image $\mathcal{S}_t = \{I_0, ..., I_{t-1}, I_t\}$, I_t étant l'*image courante* ou *image à l'instant* t et I_{t-1} l'*image précédente*.

On cherche à déterminer à chaque instant t, la pose d'un objet rigide visible au moins partiellement dans l'image courante. Suivant les notations du chapitre 3, la pose est car-

actérisée par une matrice de projection 3×4 $P_t = K_t[R_t|\mathbf{t}_t]$, avec K_t la matrice des paramètres intrinsèques et $[R_t|\mathbf{t}_t] = E_t$ la transformation euclidienne exprimant le passage d'un repère lié à l'objet au repère de la caméra. Par la suite on considère que les paramètres intrinsèques sont constants (ce qui exclu notamment les variations de la focale au cours de la séquence), c'est-à-dire $P_t = K[R_t|\mathbf{t}_t]$. La pose est alors définie par 3 paramètres de rotation et 3 paramètres de translation, que l'on regroupe dans un *vecteur de paramètres* β_t [1]. Il existe une fonction h telle que $P_t = h(\beta_t)$, de sorte que retrouver la pose de l'objet à l'instant t signifie estimer le vecteur β_t. Dans un soucis de concision, on fera l'abus d'assimiler *la pose de l'objet à l'instant t* indistinctement à β_t ou à P_t.

Par ailleurs on suppose β_0 connu.

Mesures dans les images Dans chaque image nous extrayons un ensemble de points d'intérêt, qui sont ensuite appariés avec les points extraits de l'image précédente. La mise en œuvre concrète de ces procédures, qui repose sur le GPU, sera développée en détails au chapitre 7.

Parmi les points appariés de l'image I_{t-1}, on a :

- des points qui sont l'image d'un point de la surface de l'objet suivi,
- des points du fond.

On note $\mathcal{F}_{t-1} = \{m^1_{t-1}, \ldots, m^k_{t-1}\}$ l'ensemble des points extraits et appariés de l'image I_{t-1} qui sont du premier type. De même on note $\mathcal{F}_t = \{m^1_t, \ldots, m^k_t\}$ l'ensemble des points extraits et appariés de l'image I_t. On suppose que ces ensembles sont ordonnés de sorte que m^i_{t-1} est apparié à m^i_t.

Le nombre k de ces appariements est généralement beaucoup plus petit que le nombre de points effectivement extraits et des appariements erronés subsistent.

5.2 Suivi 3D itératif à partir d'un modèle

L'approche que nous présentons ici est une approche classique de suivi itératif, pour laquelle l'estimation $\widehat{P_{t-1}} = h(\widehat{\beta_{t-1}})$ de la pose précédente sert de point de départ à l'estimation de la pose courante $\widehat{P_t}$. L'idée consiste à établir des correspondances 2D/3D, en maintenant des correspondances 2D/2D entre les points d'intérêt extraits des images.

Dans les sections suivantes (5.2.1 et 5.2.2) nous étudions le cas général, indépendamment de la nature du modèle, qui conduit à un problème d'optimisation non-linéaire. En section 5.2.3 nous détaillons comment s'exprime le problème dans le cas particulier d'une représentation par un maillage. Enfin l'étude de la résolution du problème d'optimisation en section 5.2.4 nous fournira des pistes pour une adaptation de la méthode aux nuages de points. Cette adaptation sera détaillée en section 5.3.

[1] Notons que cette paramétrisation n'est ni unique, ni anodine. Il peut être préférable d'utiliser une paramétrisation qui ne soit pas minimale (par exemple en utilisant les quaternions) et qui ne présente pas de singularité (comme en ont par exemple les angles d'Euler).

5.2.1 Fonction de transfert

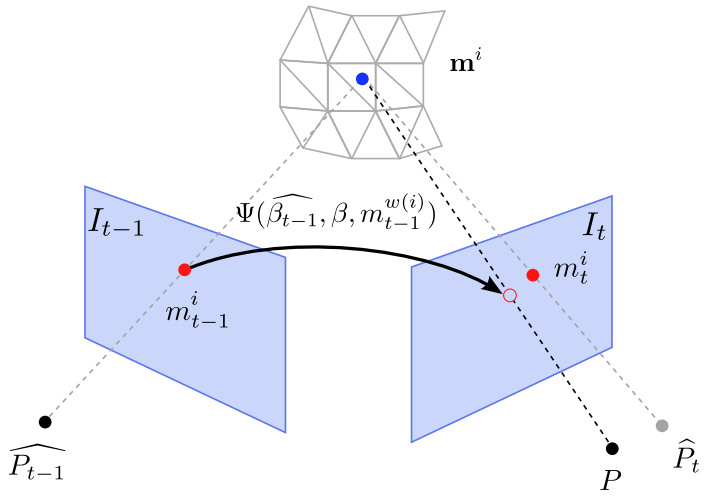


FIG. 5.2 – Fonction de transfert d'un point vers une autre vue.

La connaissance du modèle 3D est exploitée de la manière suivante : si le point m_{t-1}^i de l'image I_{t-1} est sur l'image de l'objet, alors il est l'image d'un point $\mathbf{m}^i$ situé à sa surface[2]. La projection selon une pose P du point $\mathbf{m}^i$ donne un déplacement pour m_{t-1}^i.

Si m_{t-1}^i et m_t^i sont correctement appariés, alors ils sont les projections selon les poses exactes P_{t-1} et P_t du point physique $\mathbf{m}^i$, autrement dit $m_{t-1}^i = \Phi(\beta_{t-1}, \mathbf{m}^i)$ et $m_t^i = \Phi(\beta_t, \mathbf{m}^i)$ où $\Phi(\beta, \cdot)$ est l'opérateur de projection perspective selon la pose β.

Faute de connaître la valeur de β_{t-1}, on utilise son estimation $\widehat{\beta_{t-1}}$. On n'aura donc également qu'une estimation pour le point $\mathbf{m}^i$ (et cette notation désigne cette approximation dans la suite). Pour résumer on a donc $m_{t-1}^i = \Phi(\widehat{\beta_{t-1}}, \mathbf{m}^i)$ et $m_t^i = \Phi(\widehat{\beta}_t, \mathbf{m}^i)$ (voir la figure 5.2).

On en déduit l'estimateur de la pose à l'instant t suivant :

$$\widehat{\beta}_t = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{k} \|\Phi(\beta, \mathbf{m}^i) - m_i^t\|^2 \tag{5.1}$$

Le point $\mathbf{m}^i$ est déterminé de manière non ambiguë uniquement grâce à l'estimation de la pose précédente et au modèle. Cette projection arrière consiste dans sa forme la plus générale à intersecter le rayon issu du centre de la caméra dans la pose $\widehat{\beta_{t-1}}$ avec le modèle.

[2]Plus précisément m_{t-1}^i est la mesure erronée d'un point $(m_{t-1}^i)^\star$ image d'un point $(\mathbf{m}^i)^\star$ a priori inaccessible.

On peut modéliser ce processus (projection arrière puis projection) par une fonction Ψ appelée *fonction de transfert* qui déplace un point m^i_{t-1} de l'image I_{t-1} vers l'image I_t. On a naturellement :

$$\Psi(\widehat{\beta_{t-1}}, \beta, m^i_{t-1}) = \Phi(\beta, \mathbf{m}^i)$$

Ainsi l'équation 5.1 devient :

$$\widehat{\beta}_t = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{k} \|\Psi(\widehat{\beta_{t-1}}, \beta, m^i_{t-1}) - m^i_t\|^2 \tag{5.2}$$

On obtient une formulation aux moindres carrés ordinaires classique, avec un modèle non linéaire en les inconnues β.

Cette formulation simple est cependant problématique puisqu'elle correspond à un modèle d'erreur portant uniquement sur la localisation des points m^i_t. En effet les points m^i_{t-1} sont des constantes du système ici. Or des erreurs apparaissent bien évidemment également sur les points m^i_{t-1}, et nous allons montrer comment en tenir compte.

5.2.2 Formulation symétrique

La première approche pour considérer également les erreurs sur les points de l'image I_{t-1} est d'utiliser la régression en distance orthogonale (Orthogonal Distance Regression, ODR ou moindres carrés orthogonaux).

La formulation ODR est :

$$\min_{\substack{\beta \in \mathbb{R}^6, \\ \delta_i \in \mathbb{R}^2, i \in [\![1,k]\!]}} \sum_{i=1}^{k} \|W(\Psi(\widehat{\beta_{t-1}}, \beta, m^i_{t-1} + \delta_i) - m^i_t)\|^2 + \|\Omega\delta_i\|^2 \tag{5.3}$$

où $W \in \mathcal{M}_2(\mathbb{R})$ et $\Omega \in \mathcal{M}_2(\mathbb{R})$ sont des matrices de pondération. Le problème de cette formulation, toujours non linéaire, est qu'elle possède maintenant $2k + 6$ paramètres. k étant de l'ordre de quelques centaines, ceci rend cette approche difficilement exploitable pour des applications temps-réels.

Une autre possibilité consiste à considérer le problème d'ajustements de faisceaux [TMHF00] restreints aux images I_{t-1} et I_t, qui consiste à minimiser les distances géométriques entre la projection des points 3D *triangulés* et les points d'intérêt détectés :

$$\min_{\substack{\beta_{t-1}, \beta_t, \\ \mathbf{m}^i, i \in [\![1,k]\!]}} \sum_{i=1}^{k} \|\Phi(\beta_t, \mathbf{m}^i) - m^i_t\|^2 + \|\Phi(\beta_{t-1}, \mathbf{m}^i) - m^i_{t-1}\|^2$$

C'est un problème d'optimisation non-linéaire à $2 \times 6 + 3k$ paramètres, que l'on peut également réduire aux 12 paramètres de pose à l'aide de la fonction de transfert Ψ. On symétrise ainsi le critère en introduisant le transfert des points de l'image I_t vers l'image I_{t-1}. On obtient le critère suivant :

$$\min_{\beta_{t-1},\beta_t} \sum_{i=1}^{k} \|\Psi(\beta_{t-1},\beta_t,m_{t-1}^i) - m_t^i\|^2 + \|\Psi(\beta_t,\beta_{t-1},m_t^i) - m_{t-1}^i\|^2 \tag{5.4}$$

C'est la formulation obtenue par Vacchetti et al. ([VLF04a], équation (4)), adaptée de celle proposée dans [SLZ01].

La résolution de ce problème non-linéaire ne peut être efficace que si :

1. On connaît une forme analytique pour la fonction Ψ et pour son jacobien. Dans le cas contraire le jacobien peut-être approché par différences finies, ce qui est toujours dommageable à la rapidité de la convergence et à la précision obtenue sur l'optimum.
2. Cette forme analytique a une validité locale suffisante. Plus précisément, la forme analytique de Ψ doit être la même dans un voisinage comprenant l'estimation initiale et la solution. En effet la forme analytique de la fonction Ψ dépend des valeurs particulières des poses β_{t-1} et β_t et sa détermination peut impliquer un calcul procédural (cf. section suivante), éventuellement coûteux. Limiter le nombre d'évaluation de ce calcul est crucial pour la rapidité du suivi.

5.2.3 Cas où l'objet est représenté par un maillage

Dans le cas d'un modèle défini par un maillage, $\mathbf{m}^i$ appartient à une facette du modèle, que l'on identifie au plan $\pi = \{\mathbf{x} \in \mathbb{R}^3 / \mathbf{x}^\top.\mathbf{n} + d = 0,\ \mathbf{n} \in \mathbb{R}^3, d \in \mathbb{R}\}$ ($\mathbf{n}$ est la normale de la facette). Or les projections dans deux vues d'un point appartenant à un plan sont liées par une relation homographique [HZ03].

Si $m_t^i = (x,y)^\top$ et $m_{t-1}^i = (x',y')^\top$:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim H_{P_{t-1},P_t,\pi} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

où $\sim$ est la relation d'équivalence des points dans l'espace projectif $\mathcal{P}_2$.

$H_{P_{t-1},P_t,\pi}$ est-elle que :

$$H_{P_{t-1},P_t,\pi} = K_{t-1}(\delta R - \frac{\delta\mathbf{t}.\mathbf{n}'^\top}{d'})K_t^{-1}$$

avec

$$\delta R = R_t R_{t-1}^\top,\ \delta\mathbf{t} = -R_t R_{t-1}^\top \mathbf{t}_{t-1} + \mathbf{t}_t,\ \mathbf{n}' = R_{t-1}\mathbf{n},\ d' = d - \mathbf{t}_{t-1}^\top(R_{t-1}\mathbf{n})$$

Nous sommes donc dans un cadre qui satisfait les conditions énoncées plus haut (5.2.2) : la forme analytique de la fonction Ψ est disponible, ainsi que celle de son jacobien. De plus $H_{P_{t-1},P_t,\pi}$ ne dépend que de la facette π, et par conséquent la fonction de transfert $\Psi(\beta_{t-1},\beta_t,\cdot)$ a la même forme analytique pour tout point m appartenant à l'image d'une facette π. La figure 5.3 illustre par un codage couleur des facettes d'un modèle de visage les domaines de validité locale des formes analytiques de Ψ.

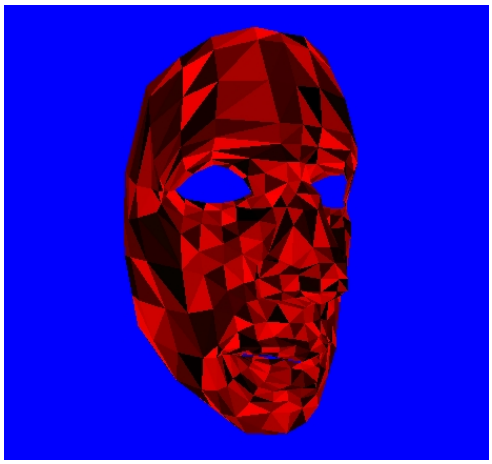


FIG. 5.3 – Domaines de validité des formes analytiques de Ψ

On peut donc, à chaque nouvelle image, après la mise en correspondance, associer à chaque couple de points appariés (m^i_{t-1}, m^i_t) la forme analytique de Ψ correspondante, que l'on supposera valide dans l'espace exploré par l'algorithme de résolution (cf. section suivante 5.2.4).

5.2.4 Résolution

L'introduction d'une fonction de transfert n'a d'intérêt que si elle dispense de la détermination du point 3D $\mathbf{m}_i$. En effet, calculer l'antécédent 3D est une opération trop coûteuse pour être répétée à chaque évaluation de Ψ (c'est-à-dire quelques centaines de fois par point et pour quelques centaines de points par image). Par exemple, dans le cas d'un maillage, comme nous l'avons vu précédemment (section 5.2.3), la détermination de Ψ nécessite seulement la connaissance de π et donc seulement la détermination de la facette à laquelle appartient $\mathbf{m}^i$.

Mais revenons à la distance géométrique $\|\Psi(\widehat{\beta_{t-1}}, \beta, m) - m'\|^2$ issue de l'équation 5.1 (nous raisonnons pour des raisons de clarté sur la formulation non symétrique). Ici $\widehat{\beta_{t-1}}$ est connue et nous notons pour simplifier m le point de l'image précédente et m' son correspondant dans l'image courante.

Dans une approche de résolution itérative du problème d'optimisation 5.2, si $\beta_t^{(k)}$ est l'estimation courante de la pose β_t, on recherchera $\beta_t^{(k+1)} = \beta_t^{(k)} + \Delta\beta^{(k)}$ grâce l'approximation au premier ordre du développement de Taylor de la fonction Ψ (par rapport au $2^{\text{ième}}$ paramètre) autour de $\beta_t^{(k)}$:

$$\|\Psi(\widehat{\beta_{t-1}}, \beta_t^{(k+1)}, m) - m'\|^2 \approx \|\Psi(\widehat{\beta_{t-1}}, \beta_t^{(k)}, m) + \frac{\partial \Psi}{\partial \beta_2}(\widehat{\beta_{t-1}}, \beta_t^{(k)}, m)\Delta\beta^{(k)} - m'\|^2 \quad (5.5)$$

où l'on note $\frac{\partial \Psi}{\partial \beta_2}$ la dérivation par rapport au deuxième paramètre de Ψ.

Puisque $\beta_t^{(0)} = \widehat{\beta_{t-1}}$, on a en particulier :

$$\begin{aligned} \|\Psi(\widehat{\beta_{t-1}}, \beta_t^{(1)}, m) - m'\|^2 &\approx \|\Psi(\widehat{\beta_{t-1}}, \widehat{\beta_{t-1}}, m) + \frac{\partial \Psi}{\partial \beta_2}(\widehat{\beta_{t-1}}, \widehat{\beta_{t-1}}, m)\Delta\beta^{(1)} - m'\|^2 \quad (5.6) \\ &= \|m + \frac{\partial \Psi}{\partial \beta_2}(\widehat{\beta_{t-1}}, \widehat{\beta_{t-1}}, m)\Delta\beta^{(1)} - m'\|^2 \quad (5.7) \\ &= \|\frac{\partial \Psi}{\partial \beta_2}(\widehat{\beta_{t-1}}, \widehat{\beta_{t-1}}, m)\Delta\beta^{(1)} + (m - m')\|^2 \quad (5.8) \end{aligned}$$

On peut interpréter les colonnes de la matrice jacobienne $\frac{\partial \Psi}{\partial \beta_2}(\widehat{\beta_{t-1}}, \widehat{\beta_{t-1}}, m) \in \mathcal{M}_{2\times 6}(\mathbb{R})$ comme les directions des projections dans l'image des déplacements élémentaires 3D. Le vecteur $\Delta\beta^{(1)}$ contient alors les amplitudes correspondantes à chacun de ces déplacements.

Ou encore, on peut dire que le vecteur $\frac{\partial \Psi}{\partial \beta_2}(\widehat{\beta_{t-1}}, \widehat{\beta_{t-1}}, m)\Delta\beta^{(1)}$ *explique* le petit déplacement apparent $d = m - m'$ du point m.

A chaque pas, il est nécessaire de réévaluer la matrice $\Psi(\widehat{\beta_{t-1}}, \beta_t^{(k)}, m)$. Dans le cas des maillages, comme nous l'avons montré en section 5.2.3, une fois la facette π déterminée, le calcul de $\Psi(\widehat{\beta_{t-1}}, \beta, \cdot)$, c'est-à-dire de $H_{\widehat{\beta_{t-1}}, \beta_t, \pi}$ ne nécessite plus de calcul géométrique, puisque le premier argument de Ψ est fixé. Dans la formulation symétrique (équation 5.4), ceci n'est plus valide car l'évolution du premier paramètre peut conduire à un «changement de facette». Vachetti et al. supposent cependant que les erreurs commises en fixant la facette en début d'optimisation sont prises en charge par leur formulation robuste du problème d'optimisation (cet aspect est détaillé dans la section 5.3.5).

Dans le cas où l'on ne possède qu'un nuage de points, la triangulation directe n'est évidemment pas possible. Pour obtenir une expression analytique de la fonction de transfert, nous proposons de linéariser le problème plus tôt, grâce au cadre théorique de l'algèbre de Lie. Ainsi nous obtenons un problème d'optimisation linéaire et ne recourons plus à un algorithme de type Gauss-Newton. Nous détaillons cette approche dans la section suivante.

5.3 Adaptation aux nuages de points

En l'absence de facettes, la transformation d'un point d'une vue à un autre n'a pas la forme compacte précédente. Il reste la possibilité de calculer le point 3D intersection de la surface et du rayon issu d'un point dans une des vues, puis de reprojeter cet antécédent dans la seconde vue. Cette approche est très coûteuse. Une autre idée consiste à constater que les positions des points du modèle lui-même sont connues et que par conséquent la projection d'un mouvement 3D de ces points par rapport à la caméra permet d'évaluer de manière éparse un champ de mouvement dans l'image. Lorsque le nuage est suffisamment dense, on peut interpoler cet échantillonnage en les points d'intérêt extraits et obtenir une approximation du champ en ces points. La mise en regard de cette *hypothèse* de mouvement avec les mesures de déplacements des points d'interêt d'une image à l'autre permet de retrouver le mouvement 3D inter-image.

Nous montrons d'abord que, lorsque le mouvement de l'objet entre deux images est petit, on peut établir une relation linéaire entre le mouvement 3D d'un point de l'objet et le mouvement 2D de son image.

Ensuite nous détaillons les différentes techniques utilisables pour «densifier» le champ de mouvement évalué de manière éparse en les points du modèle.

5.3.1 Approximation linéaire du mouvement rigide

Le mouvement rigide entre les deux images successives I_{t-1} et I_t est décrit par une transformation euclidienne dont la matrice 4×4 s'écrit :

$$\delta E = \begin{bmatrix} \delta R & \delta \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

avec δR et $\delta \mathbf{t}$ la rotation et la translation subies par l'objet entre les poses P_{t-1} et P_t. On a donc :

$$P_t = P_{t-1} \delta E$$

Pour obtenir un modèle de mouvement 2D adapté à l'utilisation d'un nuage de points sans topologie, nous suivons la méthode proposée par Drummond [DC02]. L'idée est d'exprimer la projection dans le plan image des mouvements élémentaires 3D. L'approche s'appuie sur l'algèbre de Lie [Var74, Tom97] et sur l'expression sous forme exponentielle de la matrice δE :

$$\delta E = \exp(\sum_{j=1}^{6} \alpha_j G_j)$$

Les matrices G_j sont les matrices génératrices des mouvements élémentaires 3D. Elles correspondent aux translations dans la direction des axes du repère objet :

$$G_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} G_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

et aux rotations autour de ces axes :

$$G_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} G_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} G_6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(on reconnait les approximations à l'ordre 1 autour de 0 des matrices de rotations classiques).

On regroupe dans le vecteur $\alpha = (\alpha_1, \cdots, \alpha_6)^\top$ les paramètres de translations et de rotations correspondants.

Sous l'hypothèse de petits mouvements inter-image, nous pouvons remplacer δE par l'approximation suivante :

$$\delta E \approx I + \sum_{j=1}^{6} \alpha_j G_j \tag{5.9}$$

Comment relier un petit mouvement, d'amplitude α_j, selon la transformation élémentaire G_j, d'un point 3D $\mathbf{m}$ et le mouvement induit de son projeté $m = \varphi(P\mathbf{m})$ dans l'image ?

Soit φ la fonction de *division perspective* :

$$\begin{array}{rccc} \varphi : & \mathcal{P}^2 & \longmapsto & \mathbb{R}^2 \\ & \begin{bmatrix} u \\ v \\ w \end{bmatrix} & \longrightarrow & \begin{bmatrix} u/w \\ v/w \end{bmatrix} \end{array}$$

On note $P\mathbf{m} = (u, v, w)^\top$ et $PG_j\mathbf{m} = (u'_j, v_j,' w'_j)^\top$.

La fonction $f : \mathcal{P}^3 \mapsto \mathbb{R}^2$ de projection d'un point sur le plan image s'écrit :

$$\begin{array}{rccc} f : & \mathcal{P}^3 & \longmapsto & \mathbb{R}^2 \\ & X & \longrightarrow & m = \varphi(P\mathbf{m}) \end{array}$$

On a l'approximation à l'ordre 1 suivante :

$$f(\mathbf{m} + \alpha_i G_i \mathbf{m}) \approx f(\mathbf{m}) + \sum_{j=1}^{6} \alpha_j f'(\mathbf{m}) G_j \mathbf{m} \tag{5.10}$$

où $f'(\mathbf{m}) \in \mathcal{L}(\mathcal{P}^3, \mathbb{R}^2)$ est la dérivée de f au point $\mathbf{m}$. De plus :

$$f'(\mathbf{m}) \equiv \varphi'(P(\mathbf{m})).P$$

et

$$\varphi'((u, v, w)^\top) = \begin{bmatrix} 1/w & 0 & -u/w^2 \\ 0 & 1/w & -v/w^2 \end{bmatrix}$$

L'equation 5.10 s'écrit alors :

$$m' = m + \sum_{j=1}^{6} \alpha_j l_j$$

avec

$$l_j = \begin{bmatrix} \frac{u'_j w - u w'_j}{w^2} \\ \frac{v'_j w - v w'_j}{w^2} \end{bmatrix}, \ j \in [1, 6]. \tag{5.11}$$

Le vecteur l_j s'interprète comme l'approximation à l'ordre 1 du mouvement subi par le point m projeté de $\mathbf{m}$ lorsque $\mathbf{m}$ subit une transformation 3D élémentaire d'amplitude α_j (petite) selon une translation ($j \in \{1, 2, 3\}$) ou une rotation ($j \in \{4, 5, 6\}$) élémentaire.

5.3.2 Fonction de transfert

Il découle de ce qui précède une autre formulation pour la fonction de transfert Ψ faisant intervenir la base des l_j :

$$\Psi(\beta_{t-1}, \beta_t, m) = m + \sum_{j=1}^{6} \alpha_j l_j. \tag{5.12}$$

Remarquons qu'en notant $l = (l_1, \cdots, l_6)^\top$, on retrouve une relation similaire au cas non linéaire (equation 5.8) :

$$\|\Psi(\beta_{t-1}, \beta_t, m) - m'\|^2 = \|l^\top \alpha + (m - m')\|^2. \tag{5.13}$$

Notant ensuite $\{l_j^i = (u_j^i, v_j^i)^\top\}_{j \in [\![1,6]\!]}$ la base des vecteurs de mouvements élémentaires calculée au point m_{t-1}^i, le problème de la détermination de la pose s'écrit dans ce contexte :

$$\begin{cases} \min & \sum_{i=1}^{k} \| \sum_{j=1}^{6} \alpha_j l_j^i + d^i \|^2 \\ (\alpha_1, \ldots, \alpha_6)^\top & \\ d^i = m_{t-1}^i - m_t^i = (d_x^i, d_y^i)^\top & \end{cases}$$

C'est un problème linéaire en les inconnues α_j. On peut l'écrire sous la forme matricielle suivante, en introduisant le vecteur de paramètre $\alpha_t = (\alpha_1, \ldots, \alpha_6)^\top$:

$$\min_{\alpha_t = (\alpha_1, \ldots, \alpha_6)^\top} \|L\alpha_t + d\|^2 \tag{5.14}$$

avec

$$L = \begin{bmatrix} u_1^1 & \cdots & u_6^1 \\ v_1^1 & \cdots & v_6^1 \\ \vdots & & \vdots \\ u_1^k & \cdots & u_6^k \\ v_1^k & \cdots & v_6^k \end{bmatrix} \in \mathcal{M}_{2k \times 6}(\mathbb{R}) \quad \text{et} \quad d = \begin{bmatrix} d_x^1 \\ d_y^1 \\ \vdots \\ d_x^k \\ d_y^k \end{bmatrix} \in \mathbb{R}^{2k}$$

Nous insistons sur le fait que nous ne résolvons pas le problème sous la même forme que 5.2 : nous recherchons la modification incrémentale α_t que doit subir la pose β_{t-1}. On suppose qu'un opérateur $\circ$ permet d'écrire : $\widehat{\beta_t} = \widehat{\beta_{t-1}} \circ \alpha_t$. Par ailleurs les entrées du problème sont également différentes : ce sont non plus les points de l'image précédente, mais les valeurs des vecteurs de mouvement $\{l_j^i\}_{j \in [\![1,6]\!]}$ en ces points. De même les sorties sont à présent les distances d_i induites par le mouvement α_t.

5.3.3 Symétrisation

Le modèle d'erreur postulé par l'équation 5.14 porte sur le vecteur d qui contient les distances entre points appariés. Les erreurs de localisation sur les points des deux ensembles $\mathcal{F}_{t-1}$ et $\mathcal{F}_t$ sont donc prises en compte. Cependant la matrice L est formée avec les vecteurs des bases de mouvement calculées en les points de $\mathcal{F}_{t-1}$. Une incertitude sur la position de ces points entraîne nécessairement une incertitude sur la matrice L. Pour prendre celle-ci

en compte, on peut introduire un correctif ΔL ce qui revient à formuler le problème en distance orthogonale, tout comme en section 5.2.2. On obtient le problème de moindres carrés totaux suivant :

$$\begin{cases} \min \|[\omega\Delta L \mid w\Delta d]\|^2 \\ (L+\Delta L)\alpha_t = d + \Delta d \\ \Delta L \in \mathcal{M}_{2k\times 6}(\mathbb{R}), \Delta d \in \mathbb{R}^{2k} \end{cases} \tag{5.15}$$

où w et ω sont deux facteurs de pondération.

Golub et Van Loan ont résolu ce problème ([GL91] théorème 12.3.1). La solution utilise la décomposition SVD de :

$$C = [\omega L \mid wd] = U\Sigma V^\top$$

avec

$$\Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_{p+1}) \qquad \text{et} \qquad V = \begin{bmatrix} V_{11} & v_{12} \\ v_{21}^\top & v_{22} \end{bmatrix}^{\begin{matrix} p \\ 1 \end{matrix}}_{\begin{matrix} p & 1 \end{matrix}}, \quad p = 2k.$$

La solution TLS est :

$$\widehat{\alpha_t} = -\frac{\omega}{w}\frac{v_{12}}{v_{22}}.$$

Elle est unique quand les valeurs singulières σ_p et σ_{p+1} sont distinctes.

5.3.4 Évaluation des vecteurs de mouvements

Nous présentons et comparons à présent plusieurs méthodes d'évaluation des vecteurs l_j^i en les points d'intérêt m_{t-1}^i. Nous proposons d'abord une méthode qui effectue une projection arrière explicite sur une surface définie par le nuage de points. C'est la plus précise et la plus coûteuse. Nous proposons ensuite une méthode d'interpolation du champ 2D évalué en les points du modèle uniquement. Enfin nous avons implanté une méthode qui étend les techniques de splatting présentées au chapitre 4. Elle génère directement une carte dense contenant les six champs de mouvements élémentaires.

Méthode par projection arrière sur le modèle 3D

La première approche que nous avons développée consiste à rechercher l'antécédent 3D à la surface du modèle d'un point d'intérêt m_i^{t-1}. Pour cela nous intersectons les rayons issus des points m_i^{t-1} avec la surface MLS définie par le nuage de points, comme détaillé au chapitre 4. On obtient un point $\mathbf{m}^i$. Le calcul de la base $\{l_j^i\}_{j\in[\![1,6]\!]}$ est alors immédiat, il suffit d'appliquer les relations 5.11.

Le coût de cette approche (plusieurs secondes pour la projection de quelques centaines de points) empêche son utilisation pour le suivi temps réel. En revanche, nous pourrons utiliser cette technique pour référencer sur le modèle les points d'intérêt détectés dans les images clés (puisque cette étape est faite hors ligne). La construction des images clés est présentée en section 5.4.

Méthode par *splatting*

Ici nous adaptons la méthode de rendu par *splats* présentée au chapitre 4. L'objectif ici n'est pas d'obtenir une image de synthèse de la surface définie par l'ensemble des *splats*, mais une carte où chaque pixel contient les douze composantes correspondants aux six vecteurs de la base. Il suffit ensuite de lire cette carte en les points m^i_{t-1}. Cette approche est évidemment très coûteuse et effectue plus de calculs que nécessaire. Cependant, de nombreuses mises en œuvre de cette technique sur les GPU récents ont été proposées et profitent efficacement de la puissance de ces processeurs graphiques. Nous avons donc adapté l'algorithme décrit dans [Gue05] à l'interpolation des vecteurs l_j. Les performances sont compatibles avec une exécution en temps réel, et l'utilisation du GPU permet également de libérer le CPU qui peut travailler sur d'autres tâches, comme le calcul de la pose. Les détails de mise en œuvre de cet algorithme et l'évaluation de ses performances sont présentés au chapitre 6.

La figure 5.4 montre une représentation des champs de mouvements, calculés sur toute l'image par le GPU. Pour des raisons de clarté, les vecteurs de déplacements ne sont représentés qu'en les points d'une grille régulière 30×30.

5.3.5 Robustesse aux mesures aberrantes

Les mesures $d_i = m^i_{t-1} - m^i_t$ sont issues de la mise en correspondance de points d'intérêt détectés dans les images. Ces mesures sont entachées d'erreurs :

- liées à la présence de bruit dans les images, qui rend la localisation des points d'intérêt imprécise. Ces erreurs sont d'amplitude faible (de l'ordre du pixel), elles se modélisent par une erreur numérique additive aléatoire, de loi normale sur les entrées et les sorties du modèle.
- liées à l'échec du prétraitement discret de mise en correspondance. Un point d'intérêt détecté dans une image peut ne pas être redétecté dans l'image suivante et apparié à un point localisé à l'emplacement d'un motif similaire.

Les mesures du premier type sont prises en compte par les critères aux moindres carrés que nous avons présentés jusqu'ici. En effet les estimateurs ODR des équations 5.3 et 5.15 atteignent le maximum de vraisemblance lorsque les erreurs sur les entrées et les sorties sont additives et gaussiennes.

Les échecs du dernier type introduisent des mesures dénuées de tout sens vis à vis du modèle de mouvement postulé. Les erreurs sur ces mesures sont d'amplitude arbitrairement grande (plus précisément uniquement limitée par l'algorithme de mise en correspondance lui-même), et peuvent arbitrairement modifier le vecteur de paramètre résultant. Il s'agit donc de détecter les mesures contaminées pour ne pas en tenir compte. Les critères aux moindres carrés ne sont pas robustes à ce type d'erreur. Plusieurs techniques permettent d'adapter ces critères pour les rendre robustes à taux de contamination des données satisfaisant.

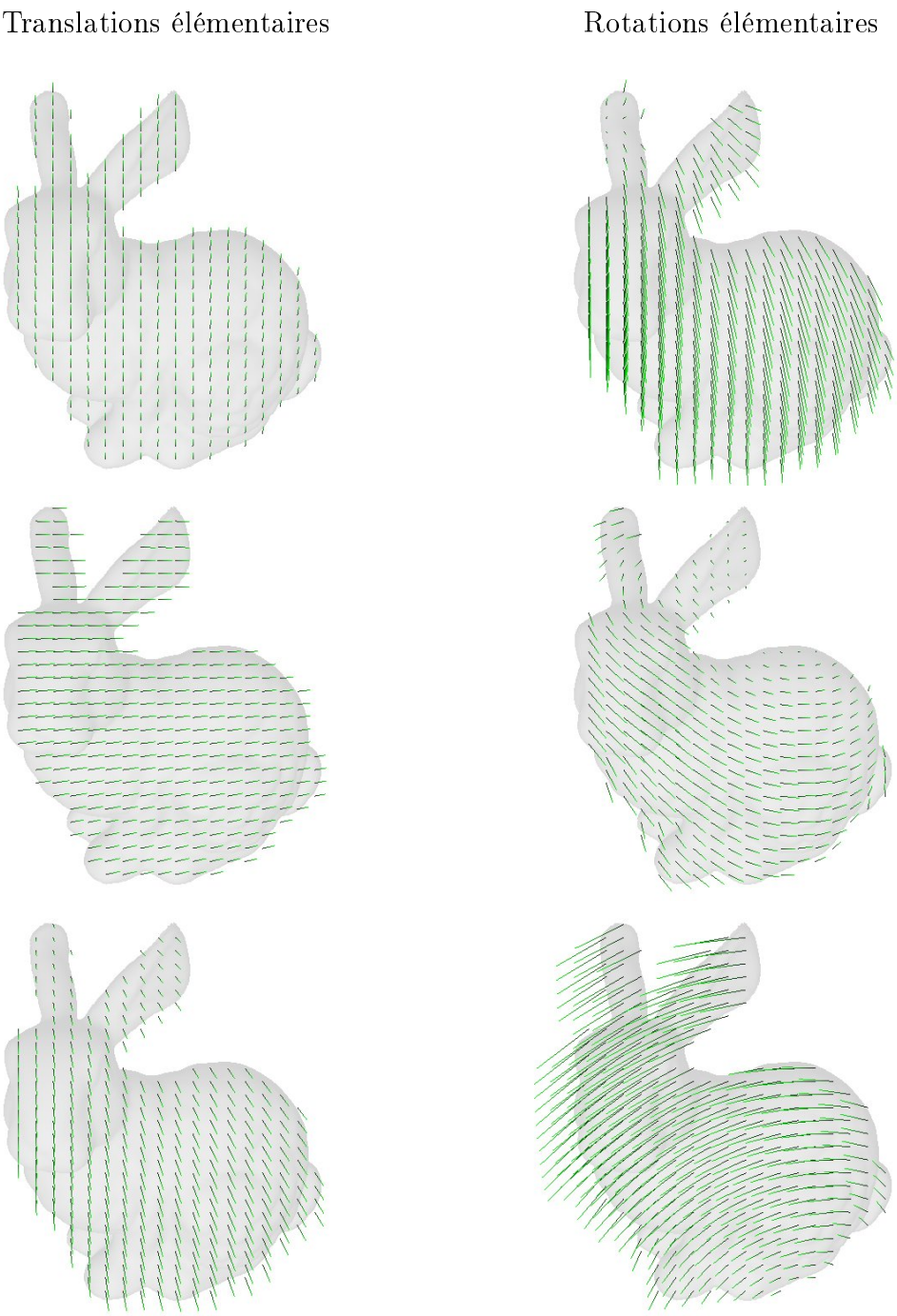


FIG. 5.4 – Champs de mouvements 2D. De haut en bas, par rapport l'axe des x, des y et des z.

Critères sélectifs Les critères sélectifs cherchent à ignorer explicitement les mesures aberrantes. Dans le cas des Moindres Carrés Tronqués, il s'agit d'appliquer le critère non robuste à un sous-ensemble de h expériences. Ces critères sont robustes si, pour un taux ϵ de mesures aberrantes, on prend $h < (1-\epsilon)n$.

Le critère des Moindres Carrés Médians minimise le maximum des résidus sur un ensemble de taille $h = \lfloor n/2 \rfloor$. Cela revient à minimiser la médiane de la norme de tous les résidus.

Critères modérateurs (M-estimateurs) Il s'agit d'une généralisation du critère OLS :

$$\widehat{\beta_{ME}} = \underset{\beta}{\operatorname{argmin}} \sum_i \rho(\|\widetilde{y_i} - f(\beta, \widetilde{x_i})\|)$$

où f est le modèle paramétré par β, $\{\widetilde{x_i}\}_i$ sont les entrées inexactes du problème et $\{\widetilde{y_i}\}_i$ les sorties inexactes.

La fonction $\rho : \mathbb{R}^+ \longrightarrow \mathbb{R}^+$ est une fonction de modération dérivable, croissante telle que $\rho(0) = 0$. On retrouve la formulation OLS lorsque $\rho(x) = x^2$. Si ρ est bornée, alors l'influence d'une mesure donnée sur le critère est limitée par cette borne et le critère est alors robuste.

Un exemple de fonction de modération est la fonction de Tukey [RL87] :

$$\rho(x) = \begin{cases} \frac{x^2}{2} - \frac{x^4}{2\sigma^2} + \frac{x^6}{6\sigma^4} & \text{si } x < \sigma \\ \frac{\sigma^2}{6} & \text{sinon} \end{cases} \tag{5.16}$$

σ où est un facteur d'échelle. Cette fonction est tracée en regard de $x \rightarrow x^2$ dans la figure 5.5.

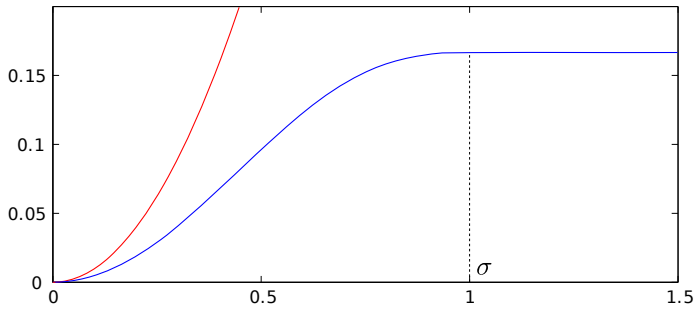


FIG. 5.5 – La fonction ρ de Tukey ($\sigma = 1$, en bleu) et la fonction «carré» (en rouge).

Vacchetti et al. utilisent ce type d'approche pour définir le critère robuste suivant :

$$\sum_{i=1}^{k} \rho(\|\Psi(\beta_{t-1}, \beta_t, m_{t-1}^i) - m_t^i\|^2 + \|\Psi(\beta_t, \beta_{t-1}, m_t^i) - m_{t-1}^i\|^2)$$

Dans notre cas, avec le critère non symétrique 5.14, l'algorithme itératif des moindres carrés pondérés itérés permet de faire la M-estimation.

5.3.6 Bilan

Nous avons présenté un algorithme itératif utilisant un modèle par points de l'objet à suivre. La suite des opérations de cet algorithme est résumée sur le schéma de la figure 5.6.

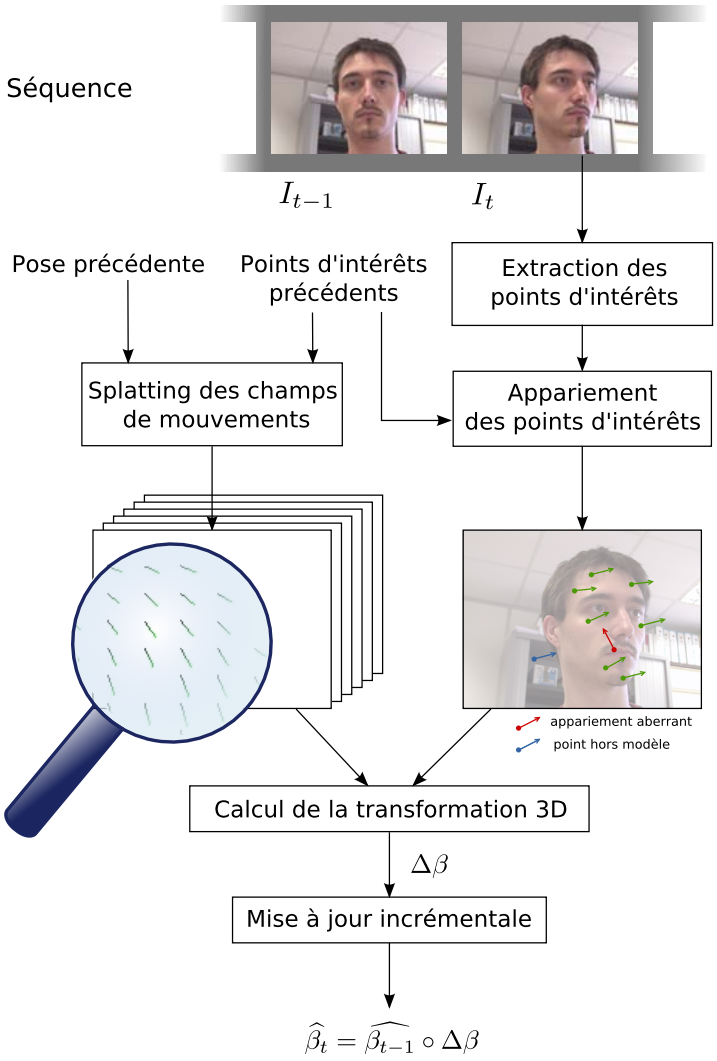


FIG. 5.6 – Schéma de principe du suivi itératif.

De part sa nature même, cet algorithme accumule les erreurs d'estimation : les points d'intérêt suivis n'étant pas fixes par rapport au modèle, le processus est en «boucle ouverte»

et la mesure du mouvement apparent est de moins en moins reliée au mouvement 3D au fur et à mesure que la pose estimée dérive par rapport à sa valeur exacte.

5.4 Intégration de connaissances a priori

Pour compenser les effets de dérive de l'algorithme précédent nous utilisons un jeu d'images clés, acquises et référencées hors-ligne. Le jeu d'images clés balaye idéalement les points de vue qui seront atteints approximativement au cours du suivi.

5.4.1 Images clés

Une image clé est composée d'une vue de l'objet dont la pose est référencée précisément hors ligne. Cette pose peut être déterminée à la main ou par toute autre méthode. Afin d'établir des correspondances 2D-3D, des points d'intérêt sont détectés dans l'image clé et reprojetés sur le modèle. Pour cette reprojection, nous considérons la surface MLS définie par le nuage de points et nous utilisons une technique de lancer de rayons décrite au chapitre 4. Cette méthode est lente mais fournit un résultat très précis. Enfin on assigne une couleur aux points visibles du modèle afin d'obtenir une texture partielle.

La figure 5.7 montre un jeu d'images clés typiques représentées dans le repère de l'objet.

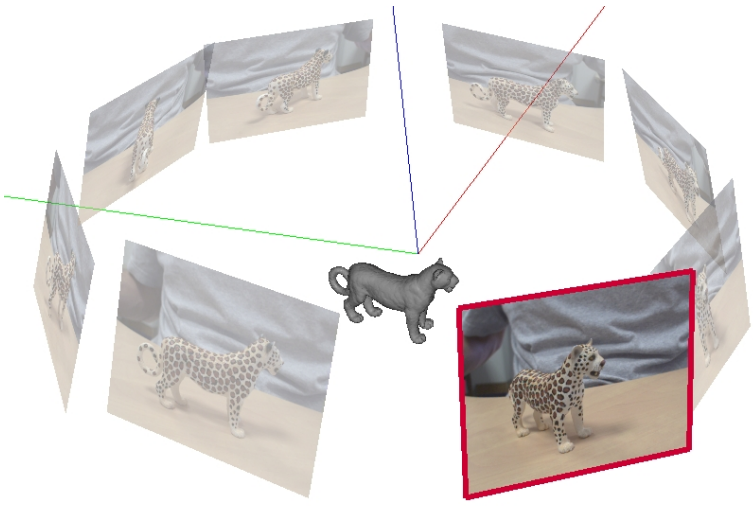


FIG. 5.7 – Un jeu d'images clés

À chaque image de la séquence, une image clé «proche» doit être choisie. Nous utilisons une distance de Mahalanobis favorisant les composantes de rotations entre les vecteurs de pose β_t et $\beta_{\text{clé}}$ et utilisons l'image clé associée à la plus petite distance parmi toutes celles de l'ensemble. La mise en correspondance de points d'intérêt entre l'image courante et

l'image clé sélectionnée fournit des correspondances 2D-3D, puisque les points de l'image clé sont également référencés en 3D sur le modèle. La figure 5.8 illustre ce principe.

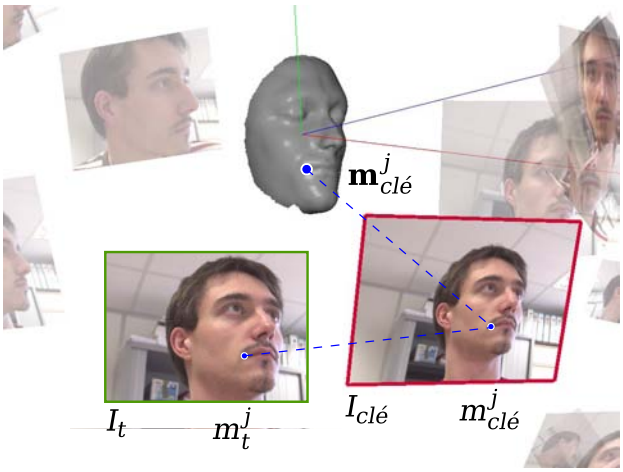


FIG. 5.8 – Obtention de correspondances 3D-2D par appariement de points dans l'image clé.

5.4.2 Appariements dans des images issues de points de vue distants

La mise en correspondance de points d'intérêt entre l'image clé et l'image courante est difficile car les deux peuvent être assez distantes. De nombreuses techniques existent pour définir des détecteurs de points et des descripteurs locaux invariants aux changements de points de vue importants. Le chapitre 7, consacré aux points d'intérêt, présente quelques unes de ces techniques. Affirmons d'ores et déjà qu'aucune n'est encore suffisamment rapide pour être utilisée en temps réel. Par conséquent nous abordons le problème différemment en synthétisant une vue intermédiaire rapprochée à partir de l'image clé. Ceci est fait de manière assez élégante en utilisant le même algorithme de *splatting* que pour le rendu des champs de mouvement. La figure 5.9 illustre comment le rendu d'une image clé permet d'obtenir une nouvelle vue. On peut ainsi, grâce à ces vues de synthèse, utiliser la même procédure d'appariement de points que dans le cas de deux images successives de la séquence.

5.4.3 Intégration à l'algorithme itératif et résultats

À partir des correspondances 2D-3D, la pose de l'image I_t est déterminée en optimisant le problème

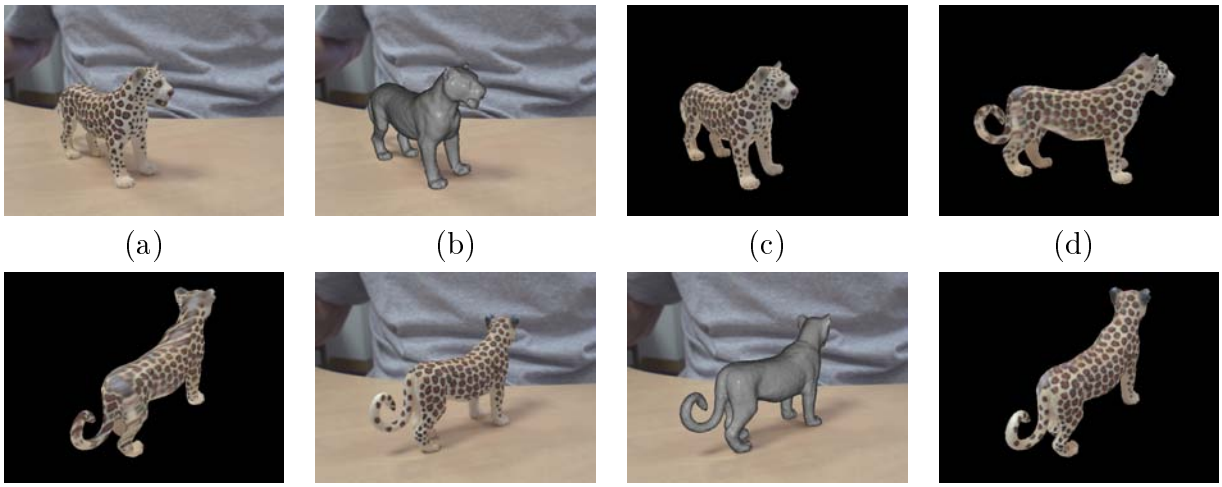


FIG. 5.9 – Rendu par *splatting* pour la génération de nouvelles vues à partir d'une image clé. Une image clé (a), dans laquelle le modèle de l'objet est référencé (b), permet le rendu dans des poses voisines (c) et (d). Pour les poses trop éloignées comme en (e), une autre image clé (f et g), plus proche de la vue demandée, permet d'obtenir un rendu adapté.

$$\widehat{\beta}_t = \underset{\beta}{\operatorname{argmin}} \sum_j \|\Phi(\beta, \mathbf{m}^j_{\text{clé}}) - m^j_t\|^2. \tag{5.17}$$

Ce problème est résolu en quelques itérations par l'algorithme de Levenberg-Marquardt. La pose initiale est choisie comme étant la pose $\widehat{\beta^1_t}$ obtenue à l'issue du suivi itératif.

Une autre approche consiste à reconstruire les prédicteurs de mouvement pour la pose $\widehat{\beta^1_t}$. Grâce aux correspondances 2D-2D calculées entre I_t et la vue de synthèse obtenue pour la pose $\widehat{\beta^1_t}$ à partir de l'image clé, nous avons les informations nécessaires pour formuler un problème similaire à celui du cas itératif. Ceci permet de raffiner la pose $\widehat{\beta^1_t}$ à partir des informations fournies par l'image clé. Cette étape peut éventuellement être itérée, mais généralement un seul pas suffit pour obtenir une précision correcte.

Évaluation quantitative Il est intéressant d'évaluer l'apport d'une approche hybride par rapport à un suivi uniquement itératif ou à un suivi s'appuyant uniquement sur les images clés. Nous avons mené une telle évaluation sur plusieurs séquences de synthèse, pour lesquelles la vérité terrain est connue. La figure 5.10 montre les résultats pour l'une d'entres elles. Dans cette séquence de 100 images tous les paramètres (translation et rotation) évoluent. Les courbes en trait plein correspondent à la vérité terrain. Les croix désignent le suivi utilisant uniquement les images clés, les cercles le suivi uniquement itératif. Les courbes en pointillés correspondent au suivi hybride.

Ces résultats illustrent bien les deux comportements distincts du suivi itératif seul et

du suivi à partir des images clés seul : le premier accumule des erreurs, la pose à la fin de la séquence étant relativement éloignée de la pose exacte. Le deuxième fournit des résultats peu précis, ce qui se traduit visuellement par un tremblement du suivi. Notre approche n'est pas affectée par le problème d'accumulation des erreurs et le tremblement est grandement réduit.

5.5 Performances et démonstration de quelques expériences de Réalité Augmentée

5.5.1 Mesures des performances

À chaque nouvelle image l'algorithme doit effectuer les tâches suivantes :

1. extraire les points d'intérêt dans l'image courante ;
2. apparier les points extraits avec ceux de l'image précédente ;
3. calculer les bases de prédicteurs de mouvements, selon la pose $\widehat{\beta_{t-1}}$;
4. résoudre le problème 5.14 pour obtenir une estimation $\widehat{\beta_t^1}$ de la pose courante ;
5. Transformer l'image clé la plus proche pour la rapprocher de la vue courante, c'est-à-dire :
   - calculer le rendu du modèle texturé de l'objet selon la pose $\widehat{\beta_t^1}$;
   - projeter les points 3D associés à l'image clé selon la pose $\widehat{\beta_t^1}$;
6. apparier les points de l'image courante et les points de l'image clé projetés selon $\widehat{\beta_t^1}$; ;
7. estimer à l'aide des correspondances 2D-3D obtenues ;

Le tableau et le graphe de la figure 5.11 résument les temps de calcul nécessaires pour chacune de ces opérations. Les opérations effectuées sur le CPU sont décalées et en teintes vertes sur le camembert. La machine de test est composée d'un Core2 Duo à 2.6 GHz et d'une carte graphique équipée d'une puce Nvidia G80. Le modèle de léopard utilisé contient approximativement 360 000 points.

Ce graphique montre une des particularités de notre algorithme : la plupart de ses opérations, en particulier les plus coûteuses sont effectuées par la carte graphique.

5.5.2 Démonstrations

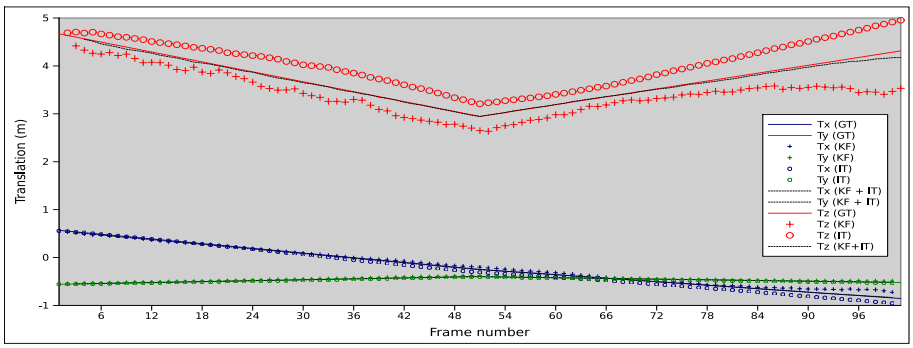
Nous illustrons le fonctionnement de l'algorithme par quelques images issues d'une application mettant en œuvre l'algorithme proposé. Le flux vidéo provient d'une caméra IEEE1394 connectée à la machine de test mentionnée plus haut.

La figure 5.12 montre des captures de sessions de suivi d'une boite de thé et d'un visage.

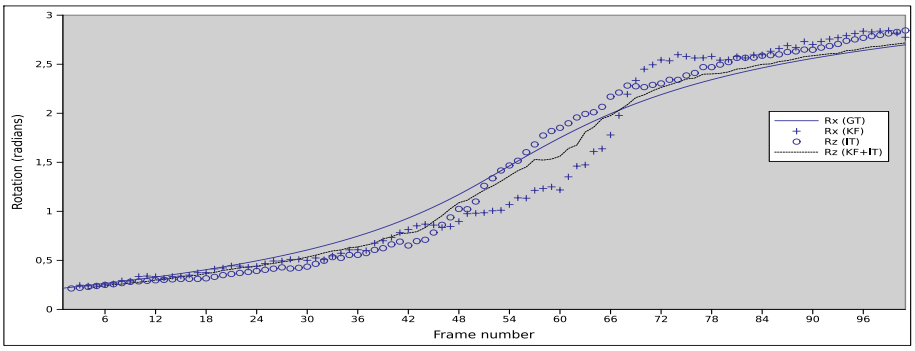
La figure 5.13 montre une démonstration simple de Réalité Augmentée. Des objets de synthèse sont superposés en temps-réel relativement au léopard. Les occultations du léopard sur les objets de synthèse sont prises en compte en désactivant les écritures dans le

(a)

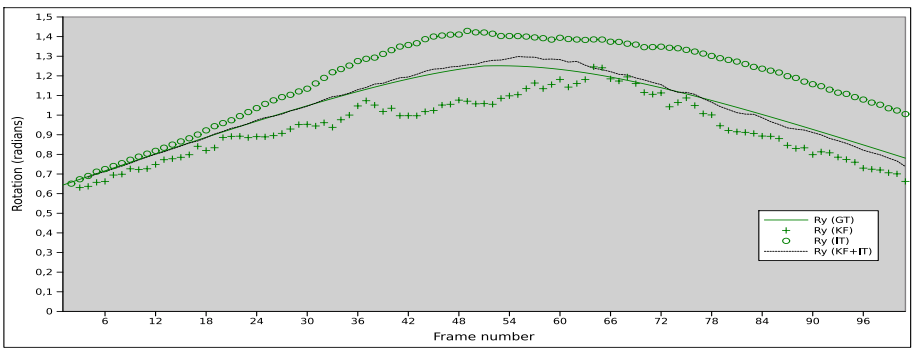
(b)



(c)



(d)



(e)

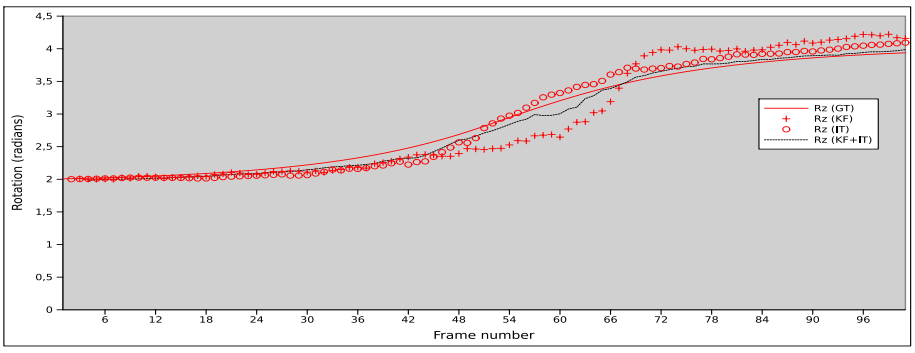


FIG. 5.10 – Estimation des paramètres de pose pour la séquence de synthèse (a). Graphe d'évolution des paramètres de translation (b). Graphes d'évolution des paramètres de rotation Rx (c), Ry (d) et Rz (e).

Opérations	temps (ms)	
Rendu texturé (GPU)	2	
Reconstruction des champs de mouvements (GPU)	17	×2
Extraction des points (GPU)	1.5	
Mise en correspondance des points (I_{t-1} et I_t et entre I_t et $I_{\text{clé}}$)	15	×2
Estimation itérative de la pose	7	×2
Estimation de la pose avec l'image clé	7	
total	81.5	(12 ips)

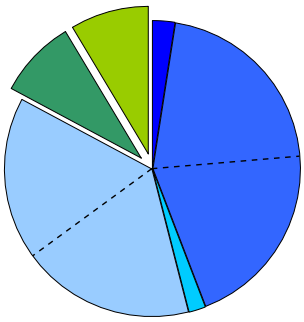


FIG. 5.11 – Temps de calculs des différentes partie de notre algorithme de suivi.

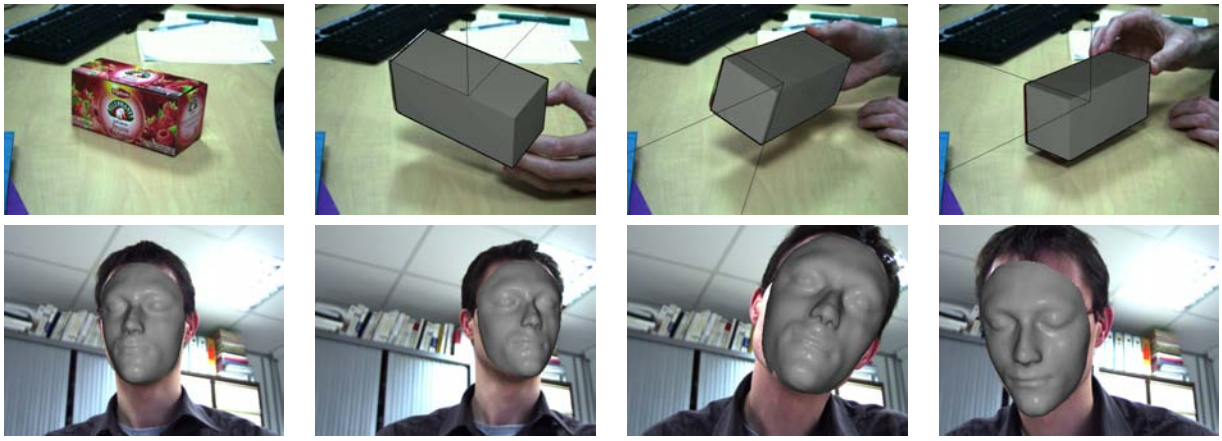


FIG. 5.12 – Captures d'écran du suivi d'une boite de thé et d'un visage. Dans le premier cas le modèle est construit de manière procédurale en échantillonnant un parallélépipède. Dans le second cas, le modèle du visage a été obtenu à l'aide d'un scanner laser.

tampon image lors du rendu du modèle du léopard. Ainsi seul le tampon de profondeur est modifié (il contient les valeurs de pronfondeur correspondant à l'objet réel) et les autres objets (rendus dans un deuxième temps) apparaissent correctement occultés.

5.6 Conclusion

Dans ce chapitre nous avons présenté une analyse détaillée du problème de suivi 3D à l'aide d'un modèle et à partir du suivi de points d'intérêt. Cette analyse nous a permis d'établir la principale contribution de notre travail, c'est-à-dire la proposition d'un algorithme de suivi visuel 3D temps réel utilisant un modèle par points. Nous présentons une solution qui permet, grâce à une mise en œuvre efficace sur GPU, détaillée dans le chapitre

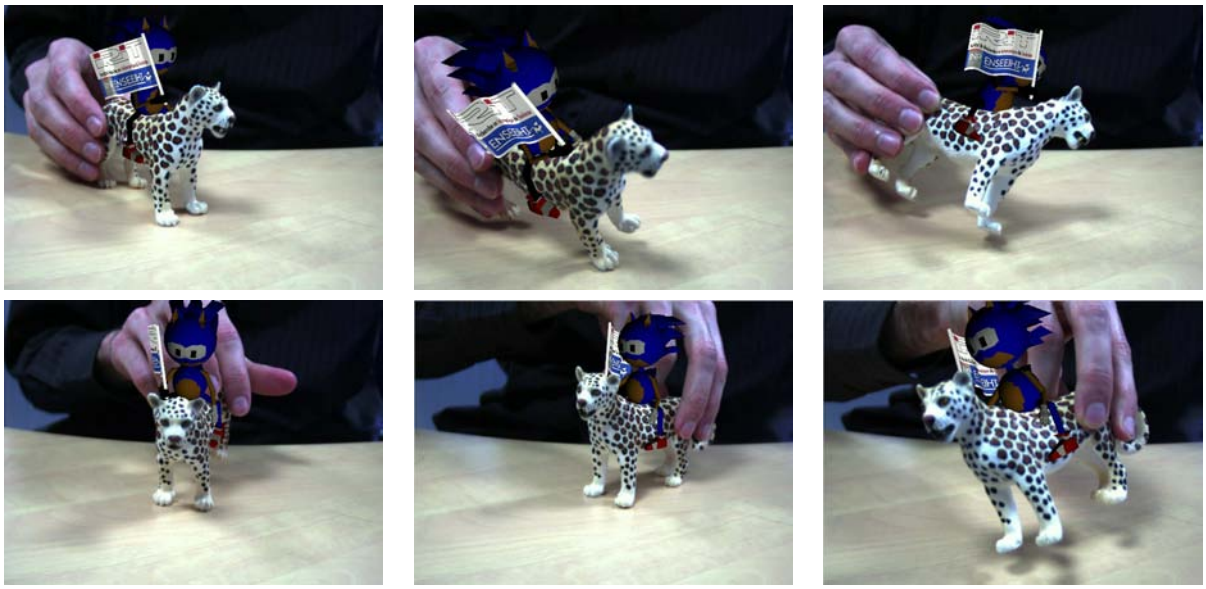


FIG. 5.13 – Une démonstration simple de Réalité Augmentée, avec prise en compte des occultations des objets virtuels par l'objet réel. Le modèle de la figurine de léopard a été obtenu à l'aide d'un scanner laser.

suivant, de calculer des champs de mouvement denses, conduisant à une résolution linéaire itérative du problème de suivi de la pose. Pour compenser les effets de dérive inhérents à ce type d'algorithme, nous utilisons un ensemble d'image clés. Le modèle par points de l'objet est ici aussi utilisé pour le référencement de points 3D et pour la synthèse de nouvelles vues.

Chapitre 6

Algorithmes de splatting sur GPU pour le rendu et l'interpolation de champs de mouvements

Un des intérêts de notre approche de suivi dans l'optique d'une exécution temps-réel est l'exploitation de l'accélération matérielle par la carte graphique pour les étapes les plus coûteuses. En particulier, les différents traitements réalisés sur le modèle 3D sont effectués par le processeur graphique. Cela présente deux avantages : d'une part les architectures graphiques sont adaptées à ce type de traitements géométriques et d'autre part il est possible de libérer le processeur central pour d'autres tâches.

Comme nous l'avons vu au chapitre 4, les techniques de reconstruction dense de modèle par points offrant actuellement le meilleur compromis entre précision et performances sont les approches par *splatting*. Un autre attrait de ces techniques est leur généralité : en effet, la reconstruction par splattting offre un cadre général pour l'interpolation dense d'attributs de la surface définie par le nuage de points, et c'est dans ce cadre que nous pouvons les utiliser pour la vision par ordinateur. Ce chapitre détaille notre mise en œuvre sur les processeurs graphiques récents des algorithmes de splatting nécessaires à l'algorithme de suivi présenté au chapitre 5. Pour rappel cet algorithme introduit deux «besoins» :

1. la capacité de déformer des images clés vers une vue proche de la vue en cours ; ceci est réalisé par un rendu «texturé» du nuage de points dans la pose cible (voir section 6.2) ;
2. la capacité d'interpoler un champ dense de vecteurs de mouvement calculés seulement sur un ensemble discret de points 3D ; nous utilisons pour cela un algorithme de *splatting* généralisé (voir section 6.3).

Les algorithmes présentés ici sont inspirés des techniques développées par Guennebaud [Gue05] et Botsch et al. [BHZK05]. Notre contribution repose sur leur utilisation originale en vision par ordinateur. Notons que Klein et Murray ont également proposé d'utiliser l'accélération matérielle de rendus 3D dans le cadre du suivi, avec une approche très différente de la notre s'appuyant sur un filtre à particules [KM06].

La suite du chapitre est organisée ainsi : la section 6.1 présente l'architecture des cartes graphiques modernes, dont la modularité permet la mise en œuvre des algorithmes de splatting présentés en sections 6.2 et 6.3. La section 6.4 évalue les performances de ces deux algorithmes et valide leur utilisation pour le suivi temps-réel.

6.1 Pipeline graphique GPU

Les cartes graphiques et les outils logiciels les exploitants pour la synthèse d'image suivent l'abstraction du *pipeline* graphique illustrée par la figure 6.1. Ce formalisme est bien adapté aux approches de synthèse d'images par projection et discrétisation de primitives 3D (par opposition aux approches par lancer de rayon, non adaptées au rendu temps réel). L'application fournit les paramètres (lumières, matrice des transformations géométriques, etc.) aux différents blocs fonctionnels (en bleu sur la figure) puis envoie dans le *pipeline* les données géométriques (sommets, normales, couleurs) qui sont successivement transformées pour finalement générer des pixels (appelés dans ce contexte *fragments*) dans les tampons images de sortie.

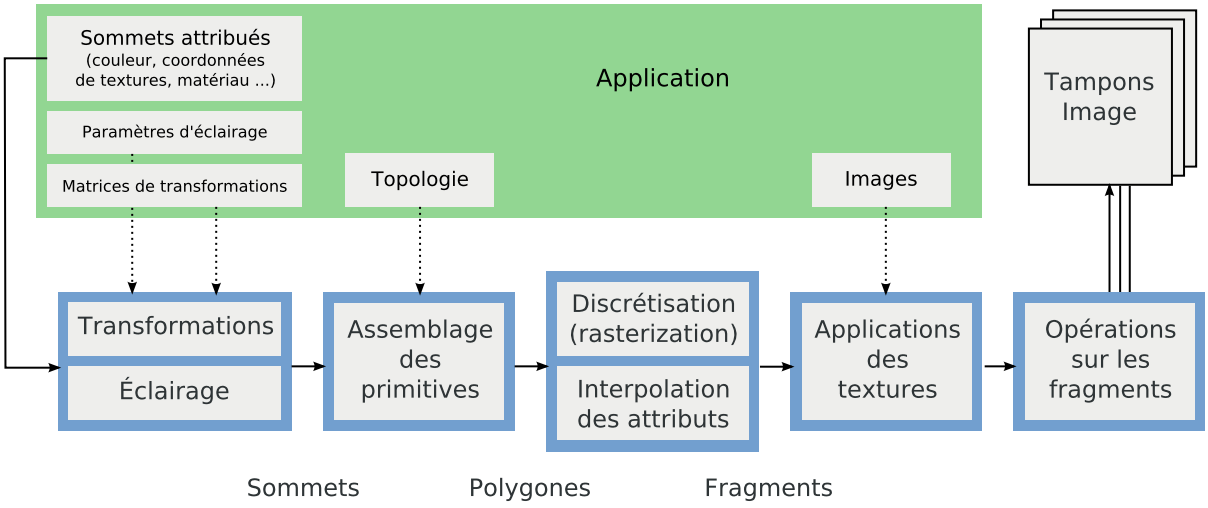


FIG. 6.1 – Le pipeline fixe de l'API OpenGL.

Les cartes graphiques modernes ont gagné des possibilités de programmation autorisant beaucoup plus de souplesse que le simple paramètrage de chaque étage du *pipeline*. Ainsi certaines des fonctionnalités fixes de la figure sont remplacées par des unités génériques programmables, indiquées en rouge sur la figure 6.2. Trois blocs sont actuellement entièrement programmables :

1. les processeurs de sommets remplacent les fonctionnalités fixes de transformations des sommets et de calcul de l'éclairage,

2. les processeurs de géométrie remplacent l'assemblage des sommets en primitives élémentaires (points, triangles et polygones),
3. les processeurs de fragments remplacent l'application de textures aux fragments générés par la discrétisation des primitives.

La programmation de chaque bloc correspond à l'écriture de petits programmes appelés *shaders* et exécutés par les processeurs de sommets, de géométrie ou de fragments. Bien entendu il est tout à fait possible de simuler tout ou partie des fonctionnalités fixes précédentes par l'intermédiaire des processeurs programmables.

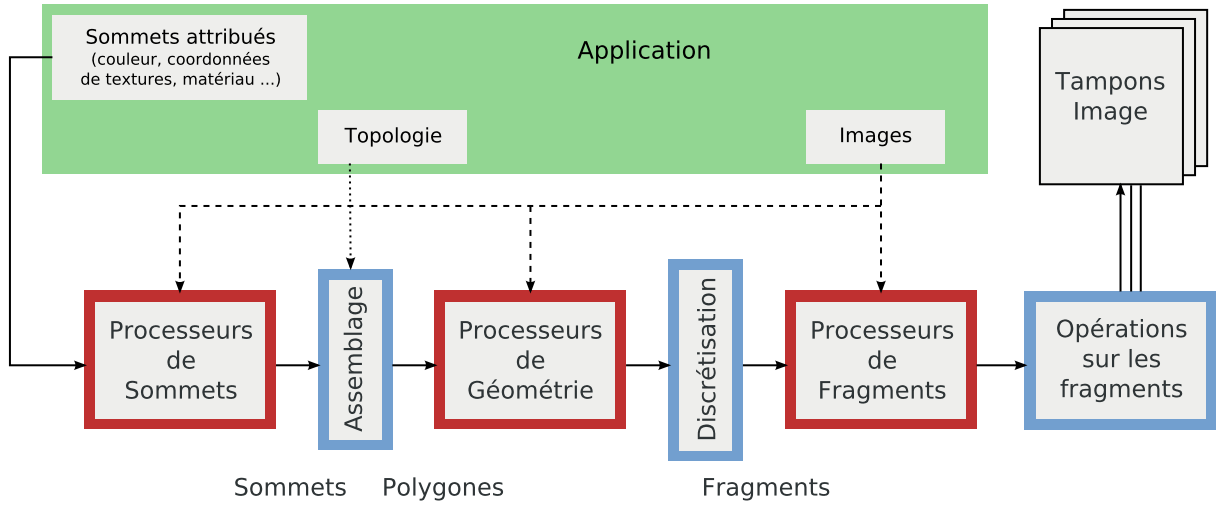


FIG. 6.2 – Le pipeline de l'API OpenGL actuelle. Les parties programmables sont représentées en rouge.

Une des particularités de cette architecture est que les données traversant les unités de traitements (sommets, géométrie, fragments) peuvent être vues comme des flux dont les données élémentaires (c'est-à-dire les sommets, les primitives ou les fragments) sont indépendantes les unes des autres. Il est ainsi possible d'exécuter parallèlement plusieurs instances d'un même *shader* (une par processeur de sommets/géométrie/fragments fourni par le matériel). C'est effectivement ce que font les GPU modernes, et c'est qui leur confère une puissance de calcul importante[1].

Dans ce chapitre, nous utilisons uniquement les unités de traitements des sommets et des fragments, par le biais de *vertex shaders* et de *fragment shaders*. Un *vertex shader* traite un seul sommet et a accès aux attributs associés (normales, couleurs, etc.). En sortie il doit fournir la position du sommet transformé. Un *fragment shader* traite un seul fragment à la fois ; il reçoit en entrée les différents attributs du fragment (position, profondeur,

[1]Le chapitre suivant propose une analyse plus poussée de ces architectures.

couleur, etc.) et doit fournir en sortie au moins la couleur du fragment transformé. La communication entre le *vertex shader* et le *fragment shader* se fait au moyen des attributs que le *vertex shader* associe au sommet transformé et qui sont automatiquement interpolés par l'étage de discrétisation. Par exemple si le *vertex shader* définit la couleur des sommets d'un triangle, les fragments générés pour ce triangle reçoivent une couleur interpolée en fonction des couleurs aux sommets.

Les *shaders* peuvent être écrits dans un langage d'assemblage spécifique ou dans des langages de haut niveau (Cg [nVi07b], GLSL (OpenGL Shading Language) [Khr07b]) qui facilitent l'interopérabilité et la maintenance du code. Pour notre part nous utilisons le langage GLSL, qui fait partie du standard OpenGL 2.0 [Khr07a].

6.2 Algorithmes de splatting sur le GPU

L'algorithme de splatting que nous utilisons est une mise en œuvre sur GPU de la technique développé par Zwicker et al. [ZPvBG01]. Nous avons présenté les fondements théoriques de cette technique au chapitre 4 (section 4.3.2). Rappelons qu'il s'agit d'interpoler différents attributs de la surface définie par le nuage de points (plus précisément de splats) par projection dans l'image de noyaux de reconstruction gaussiens définis localement en chaque splat. Un filtrage passe-bas est ensuite appliqué pour réduire les artefacts liés au rééchantillonnage de la surface.

Projeter les splats revient à calculer leur empreinte dans l'image, dont le contour est défini comme une iso-valeur des noyaux de rééchantillonnage ρ_i de l'équation 4.17 (page 66) , que nous rappelons ici :

$$\rho_i(\mathbf{x}) = |J_i| \; g_{V_{\rho_i}}(\mathbf{x} - \mathbf{x}_i).$$

Nous avons vu que lorsque les noyaux de reconstuction et le filtre passe-bas sont gaussiens et que la projection perspective est approximée localement par une transformation affine, les ρ_i sont également gaussiens et donc les iso-valeurs de ρ_i sont des ellipses.

En plus des différents attributs à interpoler (qui dépendent de l'application), les informations nécessaires en chaque point sont une normale et un rayon permettant de définir le splat.

6.2.1 Discrétisation des empreintes elliptiques

Nous détaillons ici les différentes étapes permettant d'implanter le rendu par splatting EWA sur les architectures de GPU actuelles.

Étape 1 La première étape consiste à déterminer les paramètres de l'empreinte elliptique d'un splat. Cette étape est donc réalisée en chaque point du modèle par un *vertex shader*. On considère d'abord la matrice de variance V_{ρ_i} du noyau de reconstruction ρ_i pour le splat $\mathbf{p}_i$ considéré (voir l'équation 4.18, page 66) :

$$V_{\rho_i} = J_i R_i J_i^\top + H = R'_i + H. \tag{6.1}$$

Nous suivons la méthode proposée par [Gue05] consistant à considérer la décomposition en valeurs propres et vecteurs propres de la matrice R'_i (c'est-à-dire la matrice de variance du noyau de reconstruction exprimé dans le repère du splat). On note

$$R'_i = V \Lambda V^{-1}, \tag{6.2}$$

où V est la matrice orthogonale des vecteurs propres et $\Lambda = diag(\lambda_0, \lambda_1)$ est la matrice diagonale des valeurs propres. Par ailleurs nous avons vu au chapitre 4 que le filtre passe-bas peut être défini par $H = I$, ce qui permet d'écrire :

$$\begin{aligned} V_{\rho_i} &= R'_i + I & (6.3)\\ &= V \Lambda V^{-1} + V V^{-1} & (6.4)\\ &= V(\Lambda + I) V^{-1}. & (6.5) \end{aligned}$$

En développant l'equation 4.17, on obtient l'expression suivante pour le filtre de rééchantillonnage :

$$\begin{aligned} \rho_i(\mathbf{x}) &= \frac{|J_i|}{2\pi\sqrt{|V_{\rho_i}|}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^\top V_{\rho_i}^{-1}(\mathbf{x}-\mathbf{x}_i)} & (6.6)\\ &= \Delta_i e^{-\frac{1}{2} d_{V_{\rho_i}}(\mathbf{x}-\mathbf{x}_i)}. & (6.7) \end{aligned}$$

Le filtre de rééchantillonnage s'exprime donc à l'aide d'un facteur de normalisation $\Delta_i = \frac{1}{2\pi}|J_i||V_{\rho_i}|^{-\frac{1}{2}}$ et de la distance de Mahalanobis $d_{V_{\rho_i}}(\mathbf{x}) = \mathbf{x}^\top V_{\rho_i}^{-1}\mathbf{x}$ associée à la matrice de variance V_{ρ_i}. Seule l'inverse de la matrice V_{ρ_i} est nécessaire dans ces deux termes et, en poursuivant 6.5, on a :

$$\begin{aligned} V_{\rho_i}^{-1} &= V(\Lambda + I)^{-1} V^\top & (6.8)\\ &= V \begin{bmatrix} \frac{1}{\lambda_0+1} & 0 \\ 0 & \frac{1}{\lambda_1+1} \end{bmatrix} V^\top & (6.9) \end{aligned}$$

Déterminer la matrice V équivaut à déterminer les axes de l'ellipse du splat, que l'on peut obtenir par le raisonnement géométrique suivant. Le petit axe est $\mathbf{a}_1$ (associé à la plus petite valeur propre) est colinéaire à la projection de la normale dans l'image. Ainsi si on note $\mathbf{n_i}$ la normale exprimée dans le repère de la caméra, on a (voir figure 6.3) :

$$\mathbf{a}_1 = -\frac{(\mathbf{n}_{i,x}, \mathbf{n}_{i,y})^\top}{\|(\mathbf{n}_{i,x}, \mathbf{n}_{i,y})^\top\|}. \tag{6.10}$$

Le grand axe est simplement déduit par rotation de $-\frac{\pi}{2}$:

$$\mathbf{a}_0 = (\mathbf{a_1}_{i,y}, -\mathbf{a}_{1,x})^\top. \tag{6.11}$$

Le schéma de gauche de la figure 6.3 illustre le calcul de ces vecteurs.

La plus grande valeur propre λ_0 est reliée au rayon r_i du splat : on choisit arbitrairement λ_0 comme étant le carré du rayon de la projection dans l'image de la sphère englobant le splat. La projection perspective effectue une mise à l'échelle de la forme $\eta_i = \frac{\eta}{\mathbf{p}_{i,z}}$ où η est une constante dépendant de la taille en pixels de l'image et de l'angle d'ouverture de la caméra et $\mathbf{p}_{i,z}$ est la profondeur du splat. Dans ces conditions on a :

$$\lambda_0 = (\eta_i r_i)^2. \tag{6.12}$$

Comme les splats sont des disques, le rapport des rayons de l'ellipse est donné par le cosinus de l'angle α formé par la normale du splat et la direction de visée, on a donc :

$$\frac{\lambda_1}{\lambda_0} = \cos(\alpha)^2 = (-\mathbf{v}_i^\top \mathbf{n}_i)^2. \tag{6.13}$$

Ceci est illustré sur le schéma de droite de la figure 6.3.

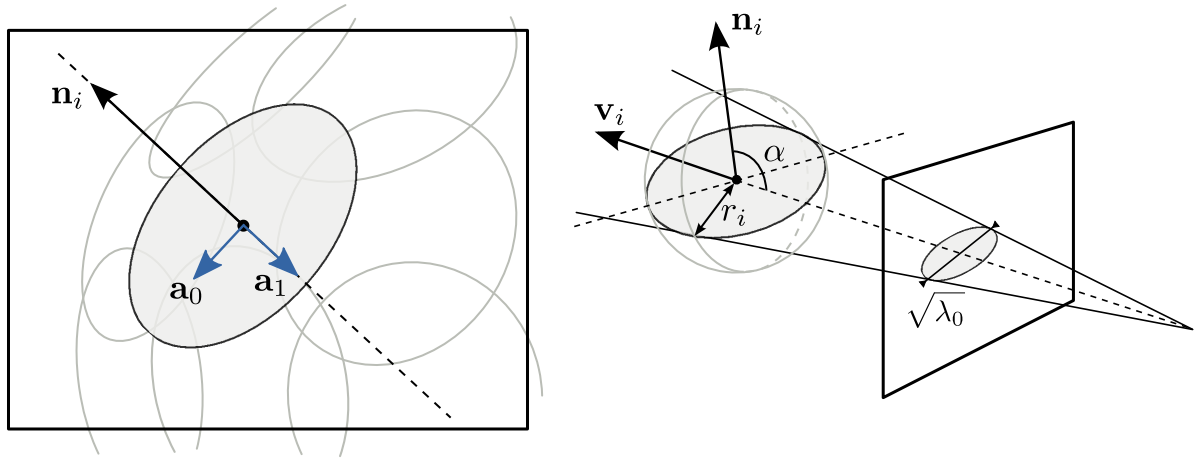


FIG. 6.3 – Calcul des paramètres de l'empreinte elliptique d'un splat : à gauche sont représentés les grand axe et petit axe de l'ellipse, à droite la valeur propre λ_0 obtenue par projection de la sphère englobante de rayon r_i. L'angle α permet de déduire la plus petite valeur propre λ_1.

Le facteur de normalisation Δ_i de l'équation 6.7 nécessite le calcul du déterminant de la matrice jacobienne J_i de la projection $\mathcal{M}_i$, que l'on obtient en considérant la mise à l'échelle des axes du repère lié au splat. On obtient :

$$\begin{aligned} |J_i| &= \left| \begin{bmatrix} \sqrt{\lambda_0}/r_i & 0 \\ 0 & \sqrt{\lambda_1}/r_i \end{bmatrix} \right| \\ &= \frac{1}{r_i^2} \sqrt{\lambda_0 \lambda_1}. \end{aligned} \tag{6.14}$$

Puis

$$\begin{aligned}\Delta_i &= \frac{1}{2\pi}|J_i|\left|\begin{bmatrix}\frac{1}{\lambda_0+1} & 0\\ 0 & \frac{1}{\lambda_1+1}\end{bmatrix}\right|^{\frac{1}{2}}\\ &= \frac{1}{2\pi\, r_i^2}\sqrt{\frac{\lambda_0\lambda_1}{(\lambda_0+1)(\lambda_1+1)}}.\end{aligned} \tag{6.15}$$

La fonction $t \rightarrow \frac{1}{2\pi}e^{-\frac{1}{2}t}$ peut être tabulée par l'intermédiaire d'une texture 1D, conduisant à une mise en œuvre très efficace. Pour résumer le *vertex shader* calcule les données suivantes :

$$\texttt{pointsize} = 2\sqrt{\lambda_0+1} \tag{6.16}$$

$$\Delta_i' = \frac{1}{r_i^2}\sqrt{\frac{\lambda_0}{\lambda_0+1}\frac{\lambda_1}{\lambda_1+1}} \tag{6.17}$$

$$Q = 4V^\top\begin{bmatrix}1 & 0\\ 0 & \frac{\lambda_0+1}{\lambda_1+1}\end{bmatrix}V. \tag{6.18}$$

Étape 2 La deuxième étape consiste en la discrétisation dans l'espace image (ou *rasterization*) de l'ellipse de matrice de variance V_{ρ_i}. Cette étape est réalisée par une combinaison des fonctionnalités fixes du pipeline graphique et un *shader*. La primitive ellipse est en effet simulée grâce à une primitive graphique particulière appelée *point sprite* : il s'agit simplement d'un carré aligné sur les axes de l'image et dont le côté est spécifié en pixels par la variable spéciale `pointsize` positionnée par le *vertex shader*. Ceci nous permet d'adapter la taille du *point sprite* à la taille calculé du splat dans l'image. L'ensemble des fragments couvrant le carré sont générés et envoyés au *fragment shader*. Ce dernier sélectionne ensuite les fragments situés à l'intérieur de l'ellipse par comparaison de $\rho_i(\mathbf{x})$ à un seuil s pour un chaque fragment $\mathbf{x}$.

Le carré couvert par le *point sprite* est paramétré par les coordonnées de texture $\mathbf{y} = (s,t)$; on a $(s,t) = (0,0)$ dans le coin inférieur gauche du carré et $(s,t) = (1,1)$ dans le coin supérieur droit (voir la figure 6.4). Pour le fragment de coordonnées image $\mathbf{x}$, on a

$$\begin{aligned}\mathbf{x} &= \mathbf{x}_i + \texttt{pointsize} \times (\mathbf{y} - (\frac{1}{2},\frac{1}{2})^\top)\\ &= \mathbf{x}_i + \texttt{pointsize} \times \mathbf{y}'.\end{aligned}$$

Comme illustré sur la figure on choisit $\texttt{pointsize} = 2\sqrt{\lambda_0+1}$, ce qui conduit à générer légèrement plus de fragment que nécessaire, mais qui est plus simple que calcul la taille exacte du carré englobant le splat.

A l'aide de la matrice $V_{\rho_i}^{-1}$ et du facteur de normalisation Δ_i calculé par le *vertex shader*, le *fragment shader* évalue le poids $\rho_i(\mathbf{x})$ du fragment $\mathbf{x}$. En notant

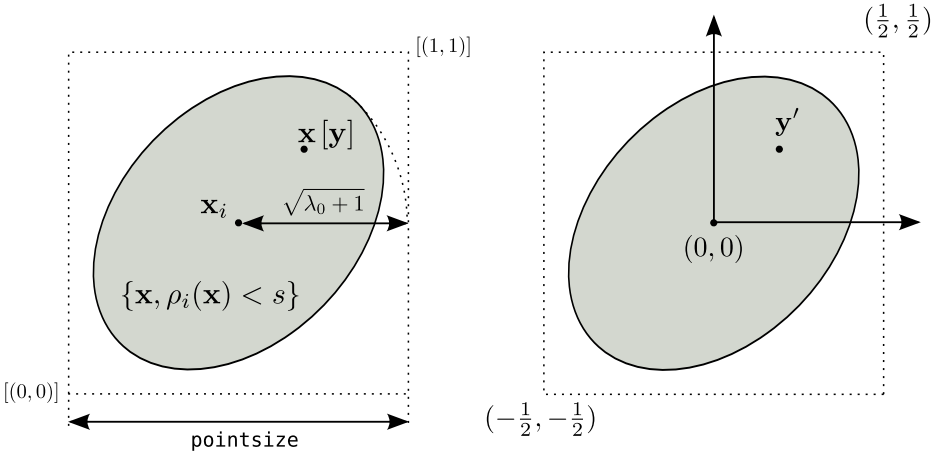


FIG. 6.4 – Rendu de l'empreinte elliptique d'un splat à l'aide de la primitive *point sprite*. Le schéma de gauche montre la primitive *point sprite* (carré en trait pointillé) et sa paramétrisation par les coordonnées de texture (notée entre crochets). Le schéma de droite montre la paramétrisation en $\mathbf{y}'$ déduite des coordonnées de texture $\mathbf{y}$ du fragment $\mathbf{x}$.

$$\begin{aligned} \Delta_i' &= \frac{1}{2\pi}\Delta_i && (6.19) \\ Q &= \texttt{pointsize}^2\, V_{\rho_i}^{-1} \\ &= 4(\lambda_0+1)V^\top \begin{bmatrix} \frac{1}{\lambda_0+1} & 0 \\ 0 & \frac{1}{\lambda_1+1} \end{bmatrix} V && (6.20) \end{aligned}$$

on obtient, dans la paramétrisation en $\mathbf{y}'$:

$$\begin{aligned} \rho_i(\mathbf{x}) &= \Delta_i e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^\top V_{\rho_i}^{-1}(\mathbf{x}-\mathbf{x}_i)} \\ &= \Delta_i e^{-\frac{\texttt{pointsize}^2}{2}(\mathbf{y}'^\top V_{\rho_i}^{-1}\mathbf{y}')} \\ &= \Delta_i' \frac{1}{2\pi} e^{-\frac{1}{2}(\mathbf{y}'^\top Q\,\mathbf{y}')}. && (6.21) \end{aligned}$$

Le *fragment shader* calcule la paramétrisation en $\mathbf{y}'$ du fragment $\mathbf{x}$, calcule la distance $\mathbf{y}'^\top Q\,\mathbf{y}'$ qui est utilisée pour accéder à la texture stockant les valeurs de l'exponentielle. Le poids du fragment est obtenu en multipliant la valeur de l'exponentielle par Δ_i'. Le fragment de sortie RVBA renvoyé est de la forme $(A_1^i, A_2^i, A_3^i, \rho_i(\mathbf{x}))$, où A_j^i désigne un des attributs que l'on souhaite interpoler.

Étape 3 Cette étape décrit la façon d'accumuler les différentes contributions des splats. Nous avons vu au chapitre 4 que les fragments contribuant à la reconstruction en un pixel donné sont sélectionnés par un algorithme de tampon de profondeur «flou», permettant

de mélanger les splats sur une petite épaisseur de surface. Malheureusement, il n'est pas possible de programmer le test de z-buffer des cartes graphiques actuelles (ce test est une des opérations du bloc appelé «opérations sur les fragments» dans la figure 6.2). Nous simulons donc ce test «flou» par un rendu en deux passes :

1. une passe de visibilité effectue un rendu simplifié consistant à n'écrire que dans le tampon de profondeur. Les fragments générés sont décalés de ε au loin dans la direction de visée. Les poids des attributs ne sont pas calculés et aucune valeur n'est accumulée ;
2. une passe de reconstruction. Cette passe utilise sans le modifier le tampon de profondeur calculé précédemment.

L'accumulation des attributs est réalisée en activant les unités de mélange de la carte graphique (opération de *blending*) et en spécifiant les équations de mélange suivantes (destination désigne le contenu courant du tampon image et fragment correspond à la valeur calculée par le *fragment shader*) :

$$\begin{aligned} RGB_{destination} &\longleftarrow RGB_{destination} + RGB_{fragment} \times A_{fragment} \\ A_{destination} &\longleftarrow A_{destination} + A_{fragment} \end{aligned}$$

Lorsque tous les splats ont été projetés, les canaux R, V et B contiennent chacun la somme pondérée de leur attribut respectif (c'est-à-dire le numérateur de l'équation 4.17) et le canal A contient la somme des poids (c'est-à-dire le dénominateur de l'équation 4.17). Une dernière passe de normalisation, consistant simplement à diviser chaque canal d'attribut par le canal A termine le rendu. Cette passe est réalisée par un *fragment shader*.

6.2.2 Résultats

Nous avons mis en oeuvre deux modes de rendu : un rendu «texturé» où la couleur en chaque splat est fournie avec le nuage de points et un rendu avec un matériau de couleur uniforme mais avec un modèle d'éclairage calculé en chaque splat par le *vertex shader*. Les figures 6.5 montrent deux images illustrant ces deux types de rendu.

Le rendu obtenu est de bonne qualité. Dans notre algorithme de suivi, nous utilisons le rendu «texturé», qui permet de déformer la texture des images clés de manière très précise et permet une recherche fiable de points homologues dans la vue déformée.

Botsch et al. [BHZK05] proposent d'interpoler la couleur et la normale de la surface et de calculer l'équation d'éclairage par pixel, pour un rendu de meilleure qualité encore et plutôt efficace, puisque l'équation d'éclairage n'est calculée qu'en les points visibles (principe de *deferred shading*).

6.3 Interpolation dense de champs de mouvement

Dans cette section, nous adaptons l'algorithme de rendu par splatting avec filtrage EWA précédent pour interpoler non plus la couleur, mais des vecteurs de mouvements définis en chaque splat.

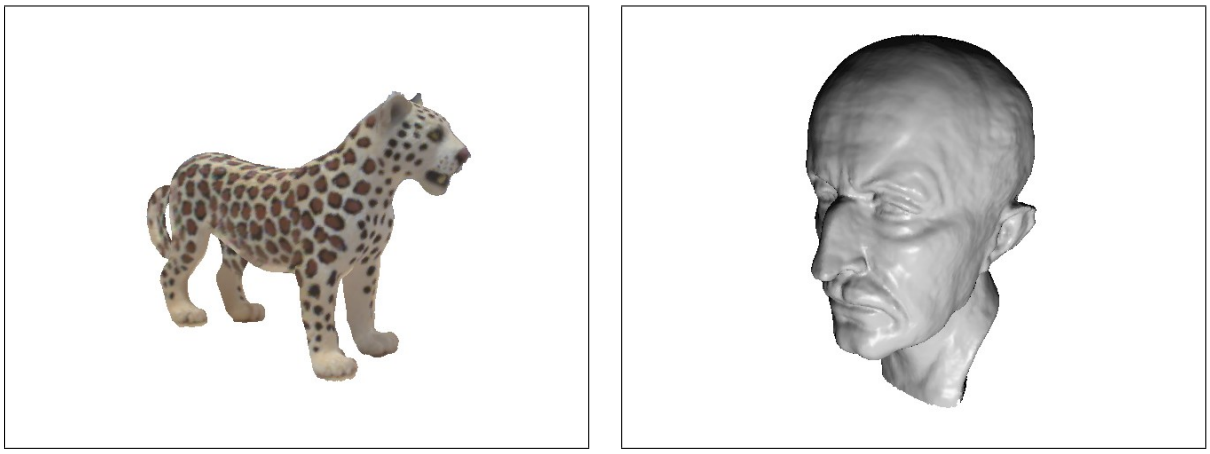


FIG. 6.5 – Deux exemples de rendu par splatting : à gauche, rendu «texturé» et éclairage uniforme ; à droite, matériau uniforme et modèle d'éclairage par splat.

6.3.1 Principe

A partir des paramètres de la caméra, il est possible de définir en chaque point du modèle une base $\{l_i, i = 1, \ldots, 6\}$ de vecteurs de mouvements 2D décrivant le déplacement de l'image du point dans la direction des transformations élémentaires 3D (rotations et translations). Voir le chapitre 5 pour plus de détails.

En chaque splat les 6 vecteurs de mouvement sont calculés par le *vertex shader* en plus des autres données nécessaires à la discrétisation de l'ellipse. Il y a donc 12 attributs envoyés au *fragment shader* pour interpolation. Puisqu'une composante supplémentaire est nécessaire pour stocker la somme des poids, $\lfloor (12+1)/4 \rfloor = 4$ tampons de sorties sont nécessaires. La somme des poids n'est cependant pas utile après la passe de normalisation, nous profitons donc de cette passe pour réorganiser les 12 attributs en seulement 3 ($\lfloor 12/4 \rfloor$) tampons de sortie. Ceci est important car le nombre de tampons est directement lié à la bande passante nécessaire pour transférer les données finales vers la mémoire centrale et donc au temps de transfert.

La figure 6.6 montre l'organisation des trois tampons de sortie finaux et une visualisation possible des vecteurs de mouvements interpolés. La lecture des composantes adéquates dans les tampons permet de représenter les valeurs d'un des six champs de mouvement en des points échantillonnés dans l'image de manière arbitraire (ici, sur une grille régulière). Dans la procédure de suivi itératif décrite au chapitre 5, les tampons sont lus aux positions correspondant aux points d'intérêt extraits dans l'image précédente.

6.4 Performances et conclusion

Dans cette section nous présentons les performances des deux algorithmes présentés précédemment. Nous les avons mesurées sur deux GPU de la marque Nvidia, le G71 et le

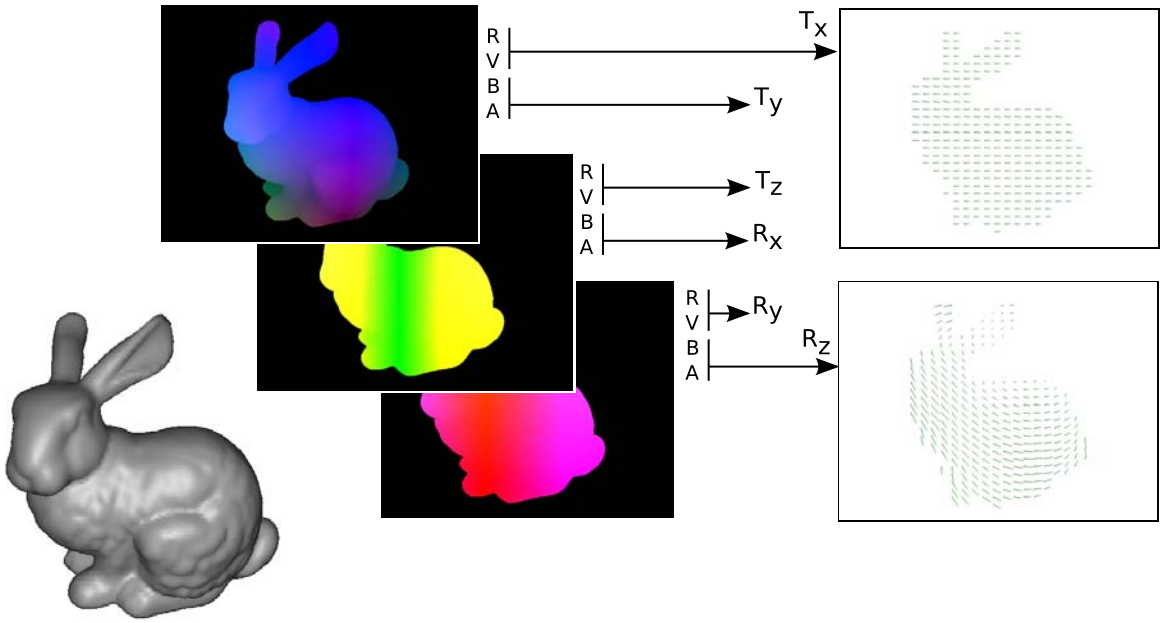


FIG. 6.6 – Visualisation des vecteurs de mouvements interpolés, à partir des trois tampons de sortie de l'algorithme sur une grille régulière 30×30.

G80.

6.4.1 Temps de rendu pour l'interpolation de la couleur

Les expériences menées portent sur trois modèles de complexité croissante et rendu autour de trois vues différentes : éloignée (1), intermédiaire (2) et proche (1). La figure 6.7 résume les résultats en terme de temps de rendu moyen par image. Nous avons détaillé les temps correspondants à chaque partie de l'algorithme : la passe remplissant le tampon de profondeur, la passe réalisant le rendu EWA et la passe de normalisation. Dans le cas où seule la couleur est interpolée, les deux premières passes ont une complexité comparable. La passe de normalisation a un coût négligeable ici.

Les vues proches requièrent le rendu de primitives plus larges, et donc le calcul de plus de fragments. D'une manière générale les performances chutent lorsque la vue est plus proche, sauf sur l'architecture G80 et pour le modèle plus complexe. Nous n'avons pas d'explication définitive pour ce phénomène, mais il est possible que la capacité de ce GPU à arrêter le rendu des fragments hors du cône de vision au plus tôt (sans exécuter le *fragment shader* notamment) soit à son origine.

6.4.2 Temps de reconstruction dense des vecteurs de mouvements

Nous avons mené la même évaluation pour l'algorithme d'interpolation dense des vecteurs de mouvements. Les résultats sont rassemblés dans la figure 6.8. Les temps de calculs ne concernent pas la phase de visualisation des vecteurs (même si les images représentent le résultat de cette visualisation) mais, contrairement à la section précédente, nous mesurons le temps de téléchargement en mémoire centrale des tampons contenant les valeurs des vecteurs de mouvements. Ce temps est comptabilisé conjointement à la normalisation des 12 composantes interpolées. Les mêmes conclusions que précédemment peuvent être tirées. Notons que le temps de téléchargement n'est pas négligeable et est même prépondérant dans le temps total pour le modèle de plus faible complexité.

6.4.3 Adaptation au suivi temps réel

Les mesures de performances montrent donc que les deux algorithmes présentés dans ce chapitre peuvent être utilisés dans le cadre du suivi temps réel. Nous avons encadré en trait gras les conditions typiques rencontrées dans le cadre du suivi, c'est-à-dire des objets moyennement complexes et filmés en vue intermédiaire. Sur l'architecture G80 les temps de calculs sont de 6.5ms et 24.5ms par image, soit 31ms pour les deux algorithmes combinés. Ceci autorise une marge suffisante pour les autres parties de l'algorithme (suivi des points d'intérêt et mise à jour de la pose).

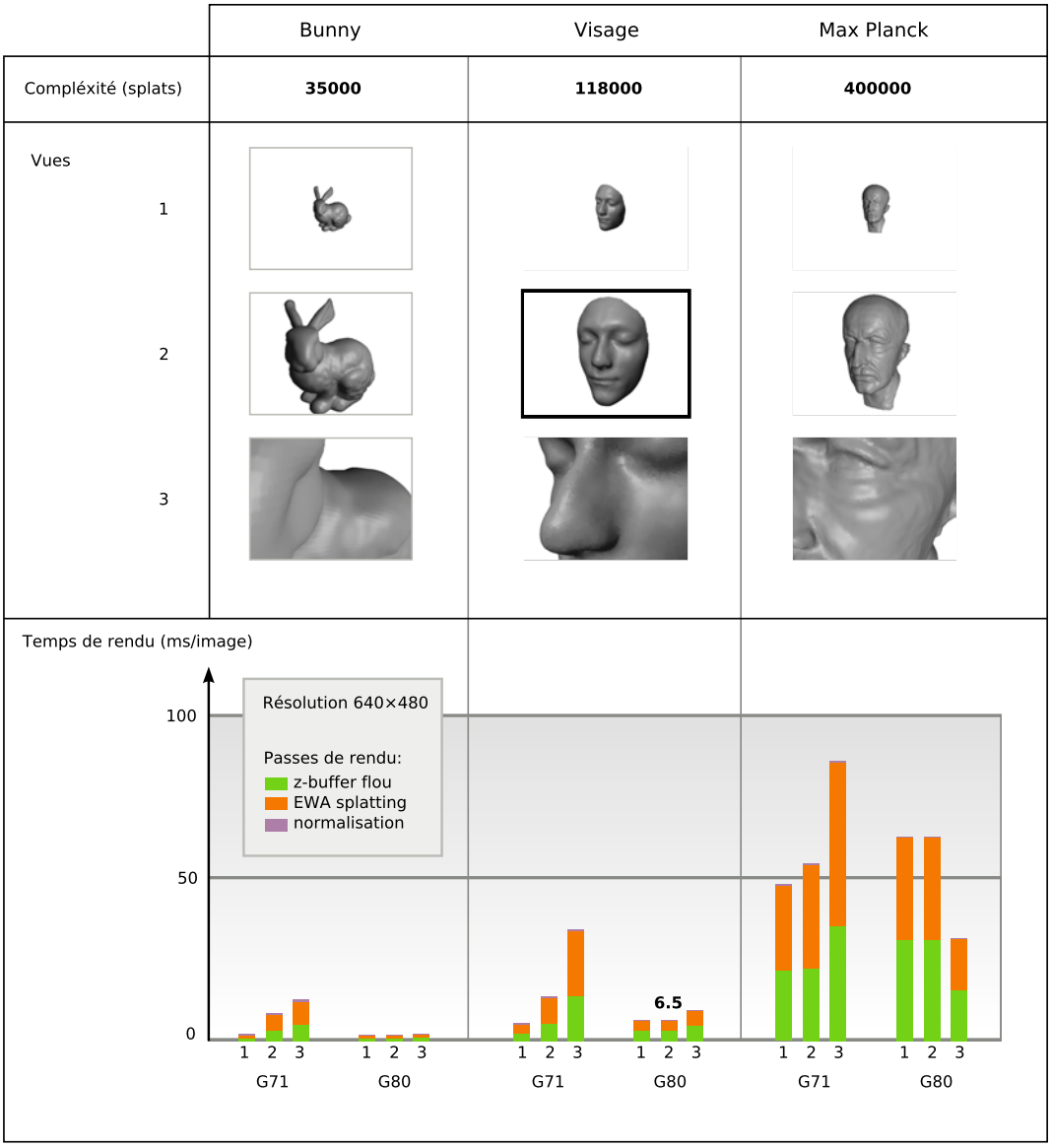


FIG. 6.7 – Temps de rendu pour l'interpolation de la couleur.

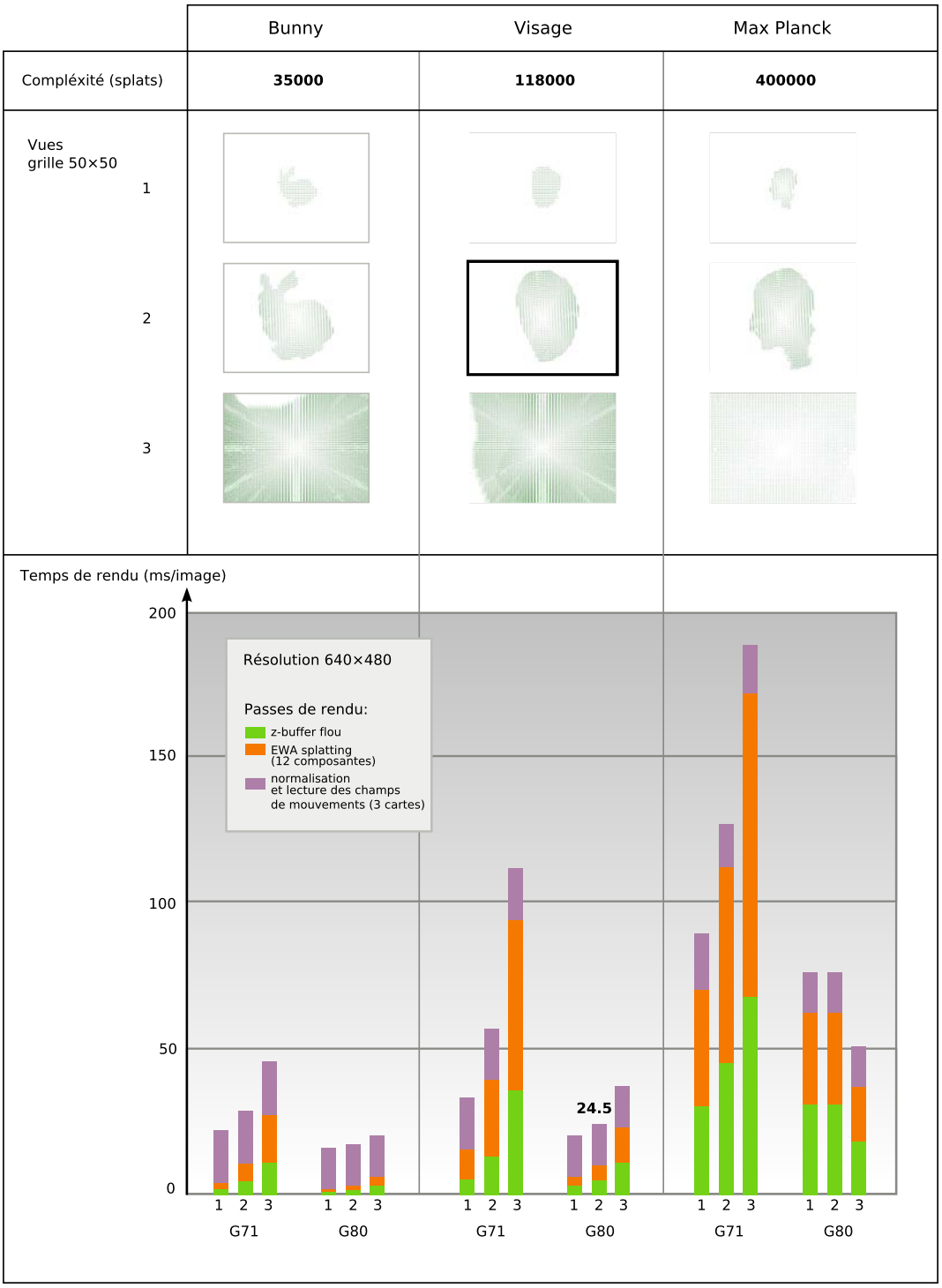


FIG. 6.8 – Temps de reconstruction dense des vecteurs de mouvements.

Chapitre 7

Extraction et appariement de points d'intérêt : étude de différentes mises en œuvre

Contexte

Au cours de la mise en œuvre de l'algorithme de suivi présenté au chapitre 5, nous avons constaté que les opérations bas niveau, liées au traitement des images de la vidéo, étaient parmi les plus coûteuses. Nous nous fixons dans ce chapitre pour objectif de mettre en œuvre ces tâches de la manière la plus efficace possible, en tirant partie des caractéristiques des CPU et des GPU modernes.

Schématiquement, on peut distinguer trois tâches dans notre application de suivi, effectuées pour chaque nouvelle image de la séquence :

1. l'extraction des points d'intérêt dans la séquence d'image et leur mise en correspondance avec les points précédemment extraits ;
2. les différents rendus du modèle c'est-à-dire d'une part le rendu texturé permettant d'obtenir une image clé déformée et d'autre part le calcul par splatting des vecteurs de mouvements élémentaires ;
3. l'estimation de la transformation 3D inter-image, puis la mise à jour et le raffinage de la pose courante. Ceci correspond aux procédures de suivi itératif et de suivi à partir d'une image clé décrites au chapitre 5.

Notre approche exploite la carte graphique pour effectuer les tâches de rendu par splatting (voir chapitre 6). Les calculs relatifs à l'estimation des paramètres de pose de l'objet utilisent les routines d'algèbre linéaire et d'optimisation aux moindres carrés fournies par la bibliothèque GSL [Fre08]. Les calculs sont donc effectués par le processeur central et nous n'avons pas exploré les possibilités d'optimisations plus loin qu'une utilisation vigilante des procédures adaptées à nos besoins. Dans ce chapitre nous nous intéressons à la mise en oeuvre de l'extraction de points d'intérêt et de la recherche des points homologues

entre deux images[1]. Nous proposons une revue rapide des techniques de la littérature et nous verrons que, s'il s'agit de techniques aujourd'hui bien maîtrisées de traitement d'images, leur coût calculatoire reste néanmoins important. A titre indicatif, Vacchetti et al. rapportent des temps correspondant à 58 % du temps de calcul total de leur application de suivi [VLF04a]. Ce chapitre est une étude des mises en œuvre sur CPU et sur GPU des algorithmes utilisés pour ces tâches.

D'une manière générale, les données vidéo ont pour caractéristiques :

- d'être de grande taille : un flux vidéo non compressé de taille 640×480 à 25 images par secondes représente près de 22 Mo par seconde ;
- d'être composées de blocs de données pouvant être traités indépendemment : c'est le cas des images individuelles au sein d'une vidéo, mais aussi celui des pixels au sein d'une même image.

Ces caractéristiques permettent soit de scinder les traitements en plusieurs parties exécutées en parallèle soit d'exécuter parallèlement plusieurs instances d'un même traitement, chacune agissant sur une portion différente des données (par exemple une image).

La suite du chapitre est organisée comme suit : la première section (7.1) est une revue des techniques proposées pour accélérer les traitements vidéos, en exploitant différents types de matériels. Les sections suivantes décrivent, après un rapide état de l'art, les mises en œuvre des algorithmes d'extraction de points (section 7.2) et de recherche des points homologues (section 7.3) que nous proposons. Nous présentons les résultats des expériences menées pour les évaluer et les comparer. Nous verrons qu'il est très intéressant sur les processeurs centraux modernes d'exploiter plusieurs cœurs. Avec un programme adapté, les performances dans le cas d'un processeur à deux cœurs sur l'algorithme d'extraction sont presque doublées (voir section 7.2.2). Les processeurs des cartes graphiques modernes (GPU) peuvent offrir des performances encore meilleures, tout en déchargeant le processeur central pour d'autres tâches. Sur le même algorithme les performances sont multipliées par trois par rapport à la version CPU à 2 coeurs (voir section 7.2.4).

7.1 Accélération matérielle des algorithmes de vision par ordinateur

La littérature propose de nombreuses approches pour accélérer les algorithmes de traitement d'images et de vision par ordinateurs à l'aide du matériel. Ces approches consistent soit à développer des unités de traitements dédiées, soit à exploiter l'architecture des composants présents dans les calculateurs généraux mais non nécessairement conçus pour ces tâches.

[1] plus précisément entre images consécutives ou entre une image de la séquence et une image clé.

7.1.1 Architectures dédiées et architectures reconfigurables

De nombreux travaux portent sur le développement d'architectures dédiées (processeurs de signal numérique, *DSP*) ou reconfigurables (circuit logique programmable, *FPGA*), dont l'architecture est bien adaptée aux traitements parallèles de grande quantité de données. Citons par exemple des circuits FPGA utilisés pour le calcul d'une carte de disparité stéréo dense [JZLA04], pour la compression vidéo [LMLN01] ou encore pour la conversion entre espaces de couleurs [BA04]. La spécialisation et la faible consommation de ces architectures les rend par ailleurs très attrayantes pour les applications embarquées (par exemple dans le domaine de l'automobile ou de la robotique).

7.1.2 Évolution des processeurs généralistes

Bien qu'extrêmement flexibles les processeurs centraux offrent des performances médiocres en comparaison des architectures dédiées pour les algorithmes utilisés en traitement d'images, qui sont souvent hautement parallélisables. Un certain degré de parallélisme peut être obtenu en utilisant les jeux d'instructions vectorielles comme MMX et SSE sur l'architecture x86. Ces instructions de type SIMD (*Single Instruction Multiple Data*) permettent de traiter jusqu'à 4 données scalaires simultanément. L'inconvénient majeur de ces approches est un temps de développement plus long et une maintenance des programmes plus difficile. Il est parfois nécessaire de réorganiser les données et de spécialiser les algorithmes pour tirer parti au mieux de ces instructions. Certains compilateurs permettent de générer directement du code exploitant ces jeux d'instructions, mais les gains de performances ne sont alors pas optimaux.

L'exploitation efficace des processeurs centraux réside donc dans l'écriture de logiciels optimisés. De nombreuses bibliothèques de fonctions existent à cette fin, citons par exemple [Int08], dont les fonctions exploitent les instructions vectorielles des processeurs x86.

Depuis quelques années un autre type de parallélisme a été introduit dans les processeurs généralistes grâce à l'utilisation de plusieurs cœurs d'exécution par processeur. On peut exploiter ces unités supplémentaires de la même manière que sur un système à plusieurs processeurs, c'est-à-dire en organisant les programmes en plusieurs processus ou processus légers (*threads*). Une partie de la gestion du parallélisme est alors déchargée au système d'exploitation. Une analyse d'une telle approche et des résultats d'expérimentations pour l'algorithme d'extraction des points d'intérêt sont présentés en section 7.2.2.

7.1.3 Utilisation des cartes graphiques

Les processeurs graphiques équipent de plus en plus de systèmes et couvrent pratiquement tout le spectre des plateformes cibles des applications de Réalité Augmentée (PDA, ordinateurs personnels et stations de travail). Ils sont donc de ce point de vue plus intéressants que les architectures dédiées, et peuvent être la base d'une plateforme générique pour la vision par ordinateur.

Architecture des processeurs graphiques

L'architecture des GPU actuels est conçue pour le traitement parallèle de blocs de données indépendantes, via un ensemble d'unités de calcul dédiées. Classiquement ces données sont des tableaux de sommets (un point en coordonnées homogènes $(x, y, z, t)^\top$) ou des tampons de pixels (définis par 4 composantes RVBA). A titre indicatif, le G71 de Nvidia dispose de 8 unités pour le traitement des sommets et de 24 unités pour le traitement des pixels. Le GPU G80 en comporte 128, utilisables aussi bien pour les sommets que pour les pixels, mais opérant sur des données scalaires (au lieu de vecteurs de taille 4). Cette architecture peut-être vue de manière abstraite comme celle d'un processeur parallèle agissant sur un flux de vecteurs (*stream processor*). Par le biais de cette abstraction, on peut exprimer des calculs généraux, c'est-à-dire sans sémantique «graphique», technique connue sous la dénomination GPGPU *General Purpose computations on GPU*. Ce domaine d'application visant à développer des techniques spécifiques pour la programmation sur GPU et à exploiter les capacités de calculs de ces coprocesseurs a produit ces dernières années de nombreuses applications démontrant l'intérêt de l'approche. Citons par exemple des utilisations en simulation en mécanique moléculaire [SPF+07] ou en algèbre linéaire pour la résolution de systèmes [GGHM05].

Programmation

Dans le modèle de programmation GPGPU, les données sont organisées en flux (*streams*) et les traitements sont définis par des noyaux (*kernels*). Les flux sont représentés par les tampons images, les textures et les tableaux de sommets ; les noyaux sont des petits programmes exécutés par les processeurs de sommets, de géométrie ou de fragments (blocs rouges sur la figure 6.2, chapitre précédent). Puisque plusieurs instances d'un même noyaux peuvent être exécutées en parallèle, le modèle d'accès mémoire impose certaines contraintes interdisant les accès concurrents à une même adresse. Ainsi un noyau de calcul peut lire aléatoirement dans les tampons d'entrée mais ne peut écrire qu'un seul sommet ou pixel. Cela implique en particulier que les tampons de sortie doivent être différents des tampons d'entrée, ce qui conduit à concevoir certains calculs comme une suite d'exécution de noyaux, entre lesquelles les tampons d'entrée et de sortie sont échangés (technique de *ping-pong*).

La technique de programmation de noyau la plus classique consiste à écrire des *shaders* que l'on compile et installe dans le pipeline de la figure 6.2. Ceci a pour effet de remplacer les fonctionnalités fixes (calcul des transformations, de l'éclairage, etc.) des APIs graphiques standards par des calculs quelconques. Cette nécessité de «détourner» un environnement de développement initialement conçu pour les applications graphiques, notamment du point des interfaces de programmation comme OpenGL, est un inconvénient indéniable de la programmation des GPU. Pour pallier à ce problème la firme NVidia développe la technologie CUDA [nVi07a], une extension du langage C dédié à la programmation générique du GPU. La marque ATI propose aussi une solution logicielle (CTM [AMD07]). Des cartes équipées de GPU mais dépourvues de sortie vidéo sont commercialisées par ces deux fabricants. Ces dernières technologies (aussi bien au niveau matériel que logiciel) sont encore très récentes,

nous n'avons pas eu l'opportunité de les évaluer. Nos mises en œuvre sur GPU présentées dans ce chapitre (sections 7.2.4 et 7.3.2) utilisent le langage GLSL dont la spécification fait partie de l'API OpenGL 2.0.

Performance

Les performances brutes des processeurs graphiques modernes sont d'un ordre de grandeur supérieures à celles des processeurs centraux (le G80 a des performances en pointe de 346 GFlops, contre 19 GFlops pour un Core2 Duo [2]). Ces chiffres n'ont qu'une portée pratique limitée, mais ils donnent une idée des capacités relatives de chaque matériel. D'autres aspects interviennent dans les performances d'une mise en œuvre particulière, telles que les bandes passantes des mémoires utilisées (mémoire centrale, mémoire embarquée sur la carte graphique) et les temps de transferts de données sur les bus de données connectant CPU et GPU.

7.1.4 Solutions implantées

L'utilisation (et encore moins le développement) d'un processeur dédié est hors du cadre de notre travail. Nous voulions conserver un certain degré de généralité pour l'approche de suivi proposée, ce qui nous incite plutôt à étudier les possibilités d'optimisations sur les CPUs et les GPU modernes dont la disponibilité et la diffusion est largement supérieure aux FPGA. Par aileurs les progrès réalisés ces derniers années par les fabricants de carte graphique sont tels que les architectures dédiées perdent de leur intérêt (du moins pour ce qui est des performances, d'autres avantages comme l'encombrement et la consommation énergétique pouvant être encore en leur faveur). Dans [CCLW05], Cope compare les performances de FPGA et de GPU pour des algorithmes de conversion colorimétrique et de convolution et montre que les GPU sont plus performants dans le premier cas, alors que les FPGA sont meilleurs dans le second. L'auteur incrimine la lenteur des accès mémoire aléatoires, et nos propres expériences (voir section 7.3.2) tendent à confirmer ce constat. Nous verrons qu'il est important d'organiser les données de manière adéquate pour limiter ce type d'accès. En particulier, les travaux de Fialka [FC06] montrent que les algorithmes de convolution peuvent être implantés sur les GPU de manière très efficace, y compris pour des filtres de grande taille, en utilisant la transformée de Fourier de l'image puis en formulant les filtres dans le domaine fréquentiel.

Sur les systèmes ne disposant pas de GPU ou lorsque les algorithmes ne sont pas adaptés à l'architecture spécifique de ce matériel, une mise en œuvre sur processeur central est nécessaire. Pour tirer partie des architectures à plusieurs cœurs qui semblent se généraliser chez tous les constructeurs, il est nécessaire de paralléliser les codes de calculs, en décomposant le code en plusieurs *threads*. Nous avons aussi analysé et évalué cette approche (voir

[2] Calcul : le G80 comporte 128 unités de calcul en virgule flottante simple précision et capables de traiter 2 instructions simultanément (MULT et ADD). Sa fréquence de référence est de 1.35GHz, ce qui donne $128 * 2 * 1.35 = 345.6$ GFlops. Le Core2 à 2.4GHz est capable de calculer 2 instructions SSE 128 bits par cycle, soit 8 instructions flottantes double précision ce qui donne $8 * 2.4 = 19.2$.

section 7.2.2).

7.2 Extraction de points d'intérêt

Les performances des algorithmes de suivi *haut-niveau* tel que celui que avons détaillé dans les chapitres précédents sont sensiblement liées aux performances des opérations *bas-niveau* (c'est-à-dire opérant directement sur les images) qui produisent les entrées de ces algorithmes. Dans notre cas le suivi des points d'intérêt fournit un échantillonnage du champ de mouvement 2D entre deux images, qui est la donnée d'entrée de l'estimation du mouvement inter-image.

Si la rapidité d'exécution est importante pour les applications de suivi temps-réel, la qualité du détecteur ne doit pas être négligée ; en effet les étapes suivantes seront d'autant plus complexes (et coûteuses) que les données en entrée sont imprécises et entachées d'erreurs. Il existe plusieurs critères qualitatifs permettant d'évaluer un certain détecteur de points d'intérêt. Schmid [SMB00] a proposé les critères de distinctivité et de répétabilité. La distinctivité d'un certain descripteur local mesure la quantité d'information véhiculée par l'ensemble des points détectés. Un détecteur produit des points distinctifs si les valeurs prises par le descripteur local sur l'ensemble des points sont «éparpillées». La distinctivité peut être définie comme l'entropie de l'ensemble des valeurs du descripteur.

La répétabilité caractérise la stabilité du détecteur vis à vis des transformations que peut subir une image. Ces transformations peuvent être de nature géométrique (changement d'échelle, rotation, transformation affine ou perspective) ou photométrique (variation de l'intensité lumineuse). Si on note H la transformation géométrique subie par une image I (et produisant I'), un point d'intérêt $\mathbf{p_1}$ détecté sur I est répété s'il existe un point $\mathbf{p'_1}$ détecté sur I' tel que $\mathit{distance}(\mathbf{p'_1}, H(\mathbf{p_1})) < \varepsilon$ (ε est un seuil de détection prenant en compte les erreurs de localisation). On caractérise la répétabilité comme la proportion de points répétés parmi tous les points détectés.

De nombreuses méthodes ont été proposées pour extraire des points d'intérêt dans une image. Toutes suivent le principe général suivant : on définit une mesure de saillance s'appuyant sur les intensités des pixels de l'image et on retient les points présentant les valeurs localement maximales de cette mesure. Le terme saillance vient de l'intuition que les points géométriquement intéressants sont ceux situés sur des coins de l'image (par exemple à l'intersection de deux contours), mais d'une manière générale les points extraits sont ceux pour lesquels la variation locale d'intensité lumineuse est grande.

L'opérateur défini par Moravec [Mor80] calcule la variation d'intensité (au sens de la somme des différences au carré, SDC) entre une fenêtre centrée sur le pixel candidat et des fenêtres légèrement décalées dans différentes directions. La mesure de saillance est définie comme la variation minimum mesurée parmi tous ces décalages ; ainsi les points d'intérêt sont localisés aux endroits de fort changement d'intensité.

Harris [HS88] étend cette idée en utilisant une approximation de la dérivée seconde de la mesure SDC par rapport au décalage de la fenêtre. Cette approximation peut s'écrire sous la forme d'une matrice M, appelée matrice d'auto-covariance ou matrice des moments

d'ordre 2, qui décrit la distribution du gradient de l'image dans le voisinage local du point $\mathbf{x} = (x, y)^\top$:

$$M = \begin{bmatrix} \overline{I_x^2(\mathbf{x})} & \overline{I_x I_y(\mathbf{x})} \\ \overline{I_x I_y(\mathbf{x})} & \overline{I_y^2(\mathbf{x})} \end{bmatrix}.$$

où I_x et I_y sont les dérivées de l'image et $\overline{\cdot}$ dénote un lissage local améliorant la stabilité (généralement réalisé par un filtre gaussien). Les points d'intérêt sont ceux en lesquels la variation du signal image est forte dans deux directions orthogonales, ce qui est caractérisé par deux valeurs propres grandes pour M. Plusieurs caractérisations de cette propriété ont été proposées [HS88, ST94].

Les développements récents dans le domaine de la détection d'objets ont contribué à l'apparition de détecteurs robustes aux déformations géométriques de grandes amplitudes, notamment les changements d'échelles. Le plus connu est le détecteur utilisé pour l'opérateur SIFT [Low04], qui s'appuie sur l'espace d'échelles obtenus par convolutions et rééchantillonnages successifs de l'image avec un filtre de différence de gaussiennes (DoG). Les extrema dans cet espace d'échelles sont retenus comme coins. Mikolajczyk et al. ont également proposés différentes adaptations du détecteur de Harris pour le rendre invariant aux changements d'échelles [MS01] et aux transformations affines [MS02]. Ces approches offrent des résultats très intéressants et sont bien adaptées entre autres aux applications de reconnaissance d'images ou de construction de panorama à partir d'images. Cependant leur coût est encore trop important pour les appliquer au suivi temps réel, surtout dans le cas d'une approche itérative pour lesquelles l'amplitude des transformations est limitée.

Une autre catégorie de méthodes consistent à analyser explicitement le voisinage d'un candidat pour voir s'il «ressemble» à un coin. Le détecteur SUSAN (Smallest Univalue Segment Assimilating Nucleus, [SB97]) analyse les niveaux de gris des pixels contenus dans un motif circulaire de taille fixe centré autour du point candidat. L'aire couverte par les pixels dont l'intensité lumineuse est proche de celle du centre (cette surface est appelée USAN) est calculée et les points d'intérêt sont définis comme ceux correspondant aux aires les plus petites. Ce détecteur est caractérisé par une bonne résistance au bruit (il ne fait pas appel aux dérivées des images) et une bonne rapidité.

Le détecteur FAST (Features from Accelerated Segment Test, [RD06]) analyse un cercle de 16 pixels autour du point candidat. S'il existe sur ce cercle un arc de n pixels plus clairs ou plus sombres que le pixel central, alors celui-ci est classé comme coin. Les auteurs proposent une mise en œuvre extrêmement efficace s'appuyant sur un test de rejet rapide ne considérant que 4 pixels lorsque $n = 12$. Pour d'autres valeurs de n, un classifieur entraîné sur une série d'images remplace ce test. Bien qu'essentiellement conçu pour être rapide, les tests montrent que ce détecteur offre une répétabilité supérieure aux détecteurs SUSAN, Harris et DoG. Cependant, le critère se montre très sensible au bruit.

Notons enfin que Trujillo et al. ont également proposé une approche par apprentissage utilisant la programmation génétique pour faire émerger un opérateur optimal au sens d'un certain critère (incluant la répétabilité) [TO06].

Nous avons choisi d'implanter le détecteur de Harris qui est encore à ce jour un des meilleurs compromis entre rapidité et répétabilité. Son algorithme conduit à des mises en œuvre simples, y compris sur GPU, contrairement aux approches reposant sur un classifieur (FAST) ou nécessitant des algorithmes itératifs (Harris affine). En effet dans ces approches, le code comporte de nombreux branchements et boucles qui empêchent d'exploiter de manière optimale l'architecture des GPU.

Les sections suivantes présentent l'algorithme d'extraction de points en détail et discutent de sa mise en œuvre sur CPU et sur GPU. Dans le cas du traitement de séquences vidéos, les images successives sont indépendantes, on peut donc en traiter plusieurs parallèlement. Cependant l'algorithme lui-même se décompose en plusieurs sous-tâches dont certaines ont pour entrée un même tampon intermédiaire, ce qui permet de les exécuter indépendamment.

7.2.1 Détails de l'algorithme

Nous travaillons sur des images en niveaux de gris. Une étape préalable convertit l'image couleur vers un espace d'intensité lumineuse.

Comme nous l'avons vu plus haut, la détection des points d'intérêt s'appuie sur l'analyse de la matrice d'auto-covariance. Nous utilisons la formulation généralisée de [MS02] :

$$M = \mu(\mathbf{x}, \sigma_I, \sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\mathbf{x}, \sigma_D) & I_x I_y(\mathbf{x}, \sigma_D) \\ I_x I_y(\mathbf{x}, \sigma_D) & I_y^2(\mathbf{x}, \sigma_D) \end{bmatrix}.$$

Les valeurs des dérivées de l'image I sont lissées par convolution avec un filtre gaussien de variance σ_D (échelle de dérivation). En pratique, on exploite la commutativité des opérateurs de dérivation et de convolution, pour écrire :

$$\begin{aligned} I_x(\mathbf{x}, \sigma_D) &= g(\mathbf{x}, \sigma) * \frac{\partial I}{\partial x}(\mathbf{x}) \\ &= \frac{\partial g}{\partial x}(\mathbf{x}, \sigma) * I(\mathbf{x}) \\ &\underset{not.}{=} g_x(\mathbf{x}, \sigma) * I(\mathbf{x}). \end{aligned}$$

Ainsi la dérivation se réduit à une unique opération de convolution avec le filtre $g_x(\mathbf{x}, \sigma)$ (dont les valeurs sont pré-calculées).

Les coefficients de M sont ensuite obtenus à partir des coefficients de corrélation par lissage avec un filtre gaussien de variance σ_I (échelle d'intégration). Il s'agit également d'une opération de convolution.

Les points d'intérêt sont ceux en lesquels la variation du signal image est forte dans deux directions orthogonales, ce qui est caractérisé par deux valeurs propres grandes pour M. Au lieu de calculer explicitement les valeurs propres de M, nous utilisons le critère proposé par Harris et al. [HS88] :

$$S(\mathbf{x}) = det(M) - \alpha\ trace(M)^2, \text{ où } \alpha = 0.06. \tag{7.1}$$

Les points d'intérêt sont ceux pour lequel $S(\mathbf{x})$ est grand. La *carte de saillance* S ainsi obtenue est ensuite filtrée pour éliminer les points qui ne sont pas des maxima locaux sur un voisinage 3×3.

Enfin on génère la liste des points d'intérêt soit par seuillage de S avec un seuil ϵ, soit en ne retenant que les k plus grandes valeurs. Dans le premier cas, on ne contrôle pas le nombre de points obtenus, dans l'autre on assure qu'un nombre constant de points sont extraits de l'image. L'avantage de cette dernière méthode est qu'elle ne nécessite pas de déterminer le seuil ϵ, qui dépend de l'image considérée.

La figure 7.1 résume les différentes étapes du traitement.

7.2.2 Mise en oeuvre sur le CPU

Nous avons d'abord implanté de manière classique l'algorithme décrit précédemment à des fins de référence. Les processeurs utilisés sont un Pentium4 cadencé à 2.6GHz et un Core2 Duo à 2.4GHz. Ce dernier possède 2 coeurs d'exécution.

L'image initiale est constituée d'un tableau d'octets codant le niveau de gris du pixel. Pour les opérations de convolutions (dérivation et lissage), on exploite la séparabilité des filtres gaussiens. La taille des supports est de 5×5 dans les deux cas. Nous utilisons des flottants simple précision pour tous les tampons d'images intermédiaires[3]. Les temps rapportés ne concernent que le calcul de la carte C (en particulier ils excluent la construction de la liste de points).

Pour comprendre les gains que peuvent apporter les architectures multi-cœurs, nous proposons une analyse très simple du parallélisme de l'algorithme.

La figure 7.2 montre une représentation chronologique des étapes de l'algorithme, où les relations de causalité entre les différentes étapes de la figure 7.1 sont signalées par des traits épais gris. L'algorithme est ainsi «compartimenté» en traitements similaires :

1. Calcul des dérivées de l'image (convolution avec les filtres g_x et g_y). Chaque convolution nécessite un temps T_0.
2. Calcul des carrés des dérivées. Il s'agit de trois multiplications termes à termes d'images. Chaque produit nécessite un temps T_1.
3. Lissage des termes précédents. Il s'agit d'une convolution avec le filtre g, qui nécessite un temps T_2.
4. Calcul de la carte S, c'est-à-dire évaluation du critère de Harris (équation 7.1) et filtrage des non-maxima. Cette étape prend un temps T_3.

On fait les hypothèses réalistes suivantes : $T_1 < T_0$, $T_1 < T_2$ et $T_3 < T_0$.

Dans le cas d'un traitement séquentiel des opérations par une seule unité d'exécution, le temps total pour traiter une image est $T_{\text{ref}} = 2T_0+3T_1+3T_2+T3$. Lorsque plusieurs unités

[3]Il est possible en mettant les filtres à l'échelle de réaliser tous les calculs en nombre entiers. Nous avons expérimenté une telle solution, mais elle ne conduit pas à des gains significatifs sur les plateformes utilisées. Par ailleurs, travailler avec les nombres entiers introduits les problèmes classiques d'aliasage et de pertes de précision.

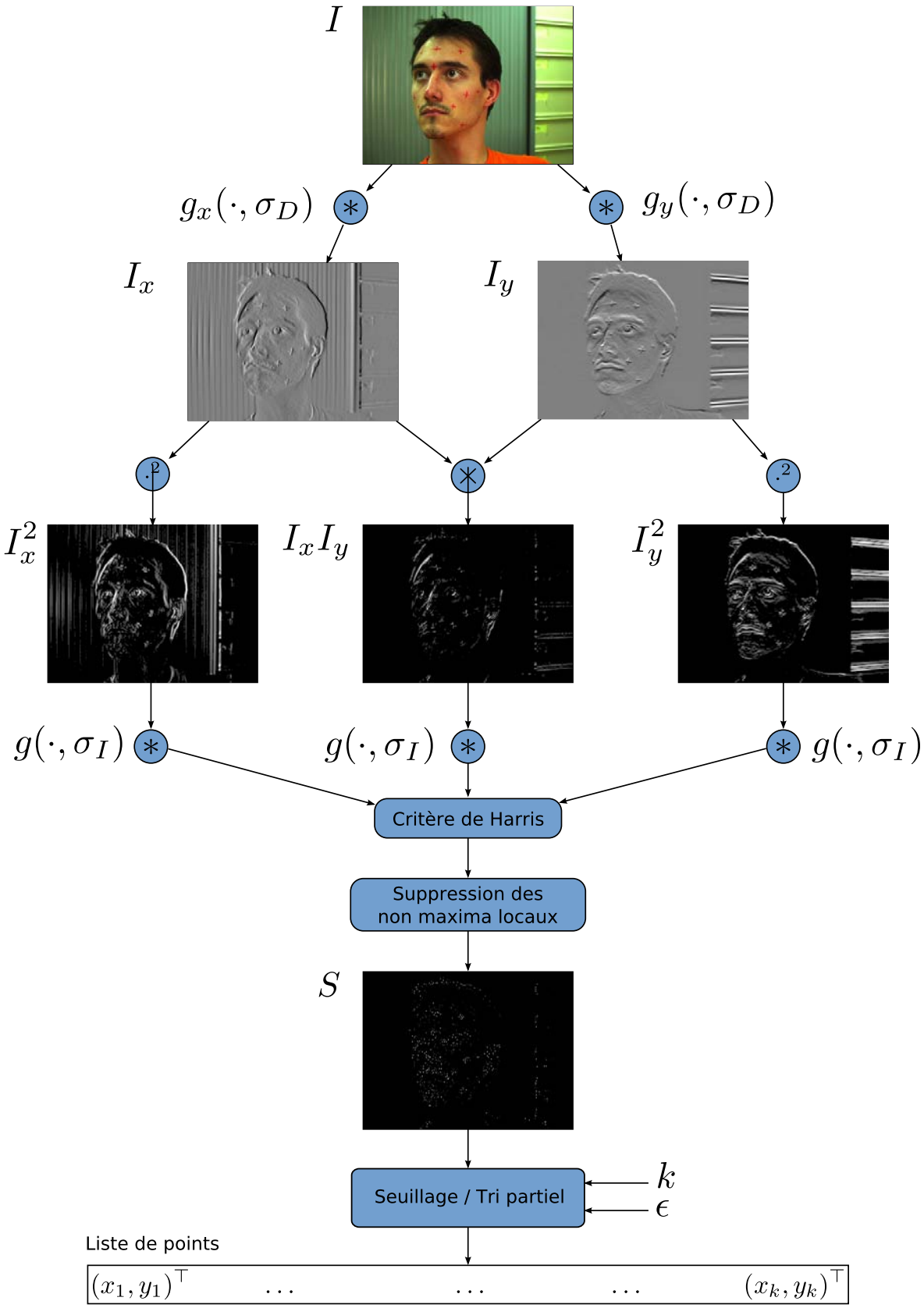


FIG. 7.1 – Schéma de principe des traitements successifs pour l'extraction de points d'intérêt.

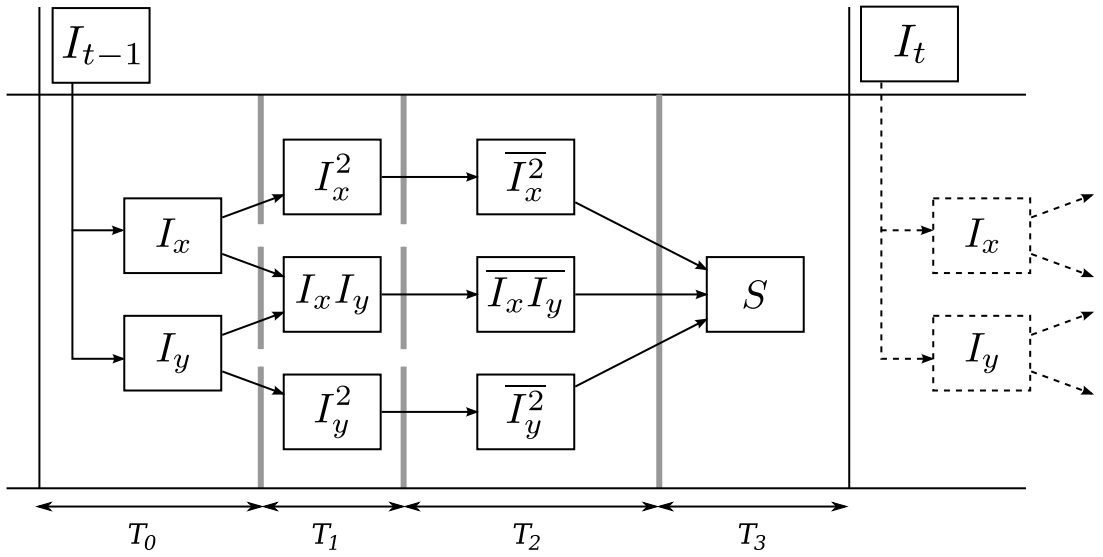


FIG. 7.2 – Représentation temporelle des tâches de la figure 7.1. Les traits gris épais correspondent aux relations de causalité entre tâches.

d'exécution sont disponibles, les traitements au sein d'un même compartiment peuvent être réalisés en parallèle. La figure 7.3 détaille le cas de deux processeurs. La deuxième unité d'exécution permet de «masquer» totalement une des convolutions de la première étape. L'ordonnancement des étapes suivantes est plus délicat et on abouti à un temps total de traitement de $T_0 + T_1 + 2T_2 + T_3$, avec un reliquat de temps équivalent à $T_2 + T_3 - T_1$. Comme signalé en pointillé sur la figure, ce reliquat peut servir à masquer une partie du calcul d'une convolution de l 'étape 1 pour l'image suivante. Avec deux unités de calculs le temps moyen optimal (en «régime permanent») vaut $\frac{1}{2}T_{\text{ref}} = T_0 + \frac{3}{2}T_1 + \frac{3}{2}T_2 + \frac{1}{2}T_3$. Pour l'atteindre, il faut prendre en comptes les contraintes de causalité, afin d'éviter la famine de l'un des processeurs. Ainsi, on donnera toujours la priorité aux calculs des dérivées I_x et I_y sur d'autres tâches pouvant être effectuées. En effet le calcul du produit $I_x I_y$ dépend de ces deux résultats. De même le calcul de I_x^2, I_y^2 et $I_x I_y$ doit être privilégié par rapport aux calcul des moyennes pondérées.

La figure 7.4 présente une analyse similaire avec trois CPU. On a également un reliquat, de longueur T_3 pour deux unités de calculs qui peuvent être utilisées pour démarrer les calculs indépendants I_x et I_y pour l'image suivante. On masque ainsi le temps T_3 pour obtenir, sans contrainte particulière d'ordonnancement, un temps total équivalent à $T_1 + T_2 + T_0$. Le temps optimal vaut ici $\frac{1}{3}T_{\text{ref}} = \frac{2}{3}T0 + T_1 + T_2 + \frac{1}{3}T_3$, et il difficile d'établir les règles d'ordonnancement permettant d'atteindre ce temps d'exécution.

Cette analyse montre qu'il est possible de profiter du parallélisme intrinsèque de l'algorithme d'extraction de points d'intérêt. Une mise en œuvre simple consiste à utiliser une

file d'attente par compartiment et un ensemble de *threads*. Un *thread* exécute la boucle consistant à sélectionner un traitement dans l'une des files, en appliquant les règles de priorité, à dépiler la donnée correspondante, à la traiter et à placer le résultat dans la file du traitement suivant. L'objectif est d'éviter que toutes les files soient vides pendant une période donnée, car alors un *thread* est en attente.

Il faut enfin noter qu'en pratique, d'autres aspects sont à prendre en compte, comme la cohérence des caches de données, les latences d'entrées/sorties (acquisition depuis une caméra, communication avec le système d'affichage) et le sur-coût induit par l'ordonnancement des tâches sur les CPU par le système.

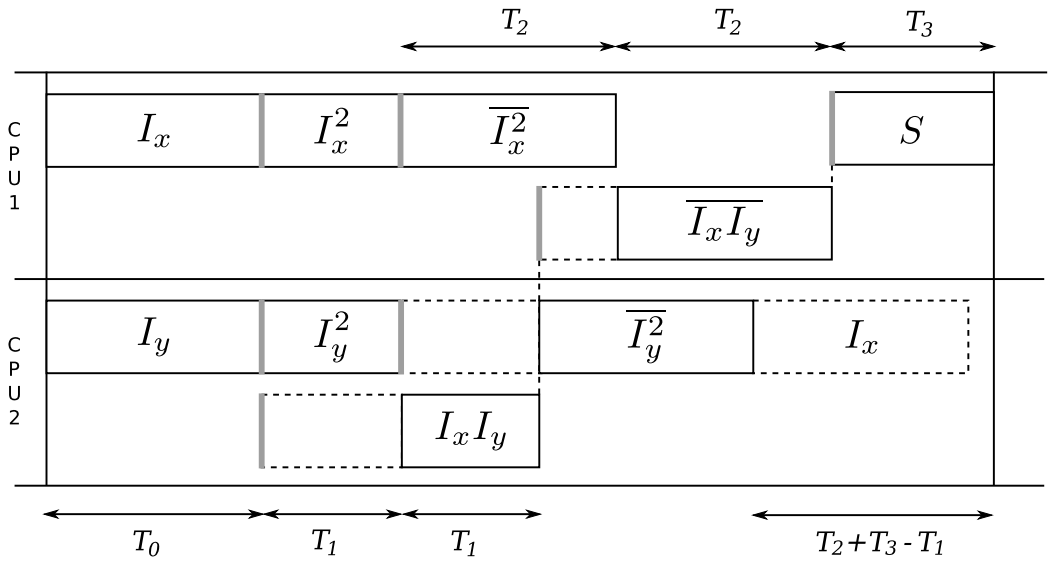


FIG. 7.3 – Parallélisation des traitements avec 2 unités d'exécution. Les traits gris épais correspondent aux instants de début au plus tôt d'une tâche donnée, contraints par les relations de causalité.

7.2.3 Mesure des performances

Pour évaluer les gains possibles en utilisant plusieurs *threads*, nous avons implanté un mécanisme plus simple consistant à assigner à chaque *thread* une image de la séquence, prise en séquence dans une file d'entrée. Notons qu'il est nécessaire, pour les traitements suivant l'extraction, de mettre en place un système de ré-ordonnancement, par exemple en utilisant une file de sortie maintenue en permanence classée par numéro d'ordre des images. Cette approche est plus simple à mettre en oeuvre et se met a priori bien à l'échelle lorsque le nombre d'unités de calcul augmente (il suffit d'augmenter le nombre de *threads*).

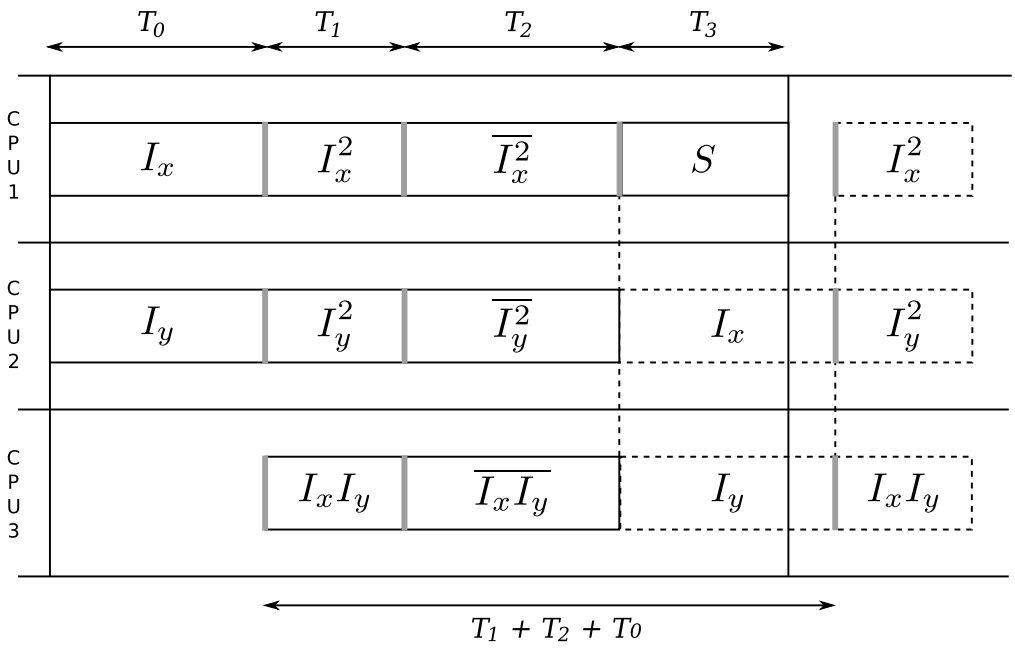


FIG. 7.4 – Parallélisation des traitements avec 3 unités d'exécution.

Les séquences de test

Pour comparer les performances des différentes mises en œuvre, nous utilisons tout au long de ce chapitre trois séquences vidéos de taille différentes, illustrées sur la figure 7.5. La première a une résolution de type visio-conférence (320×240), la seconde est de qualité similaire au DVD (640×480) et la dernière est une séquence haute définition (1280×720).

Les trois séquences comportent chacune 500 images. Nous avons mesuré le temps nécessaire à l'extraction des points d'intérêt sur la totalité de la séquence et nous déduisons le temps moyen par image. Les séquences sont lues depuis le disque dur et se présentent sous la forme de fichiers image non compressés.

Gains apportés par la version *multi-threads*

Le graphe de la figure 7.6 montre les temps de calcul sur nos deux machines de test des versions mono et multi *threads* de l'algorithme d'extraction des points d'intérêt. La version multi-threads utilise 2 *threads* : en effet, sur les deux machines, nous avons constaté qu'utiliser 3 *threads* n'apportait aucun gain de performances et qu'à partir de 4 *threads*, les performances se dégradent, à cause du surcoût système lié à la gestion des *threads*.

On constate que même sur le Pentium 4 qui ne possède qu'une unité de calcul, l'utilisation de plusieurs *threads* apporte des gains notables, (jusqu'à $1,34$ fois plus rapide), ceci étant vraisemblablement dû au masquage des temps d'accès mémoire. Sur le Core 2

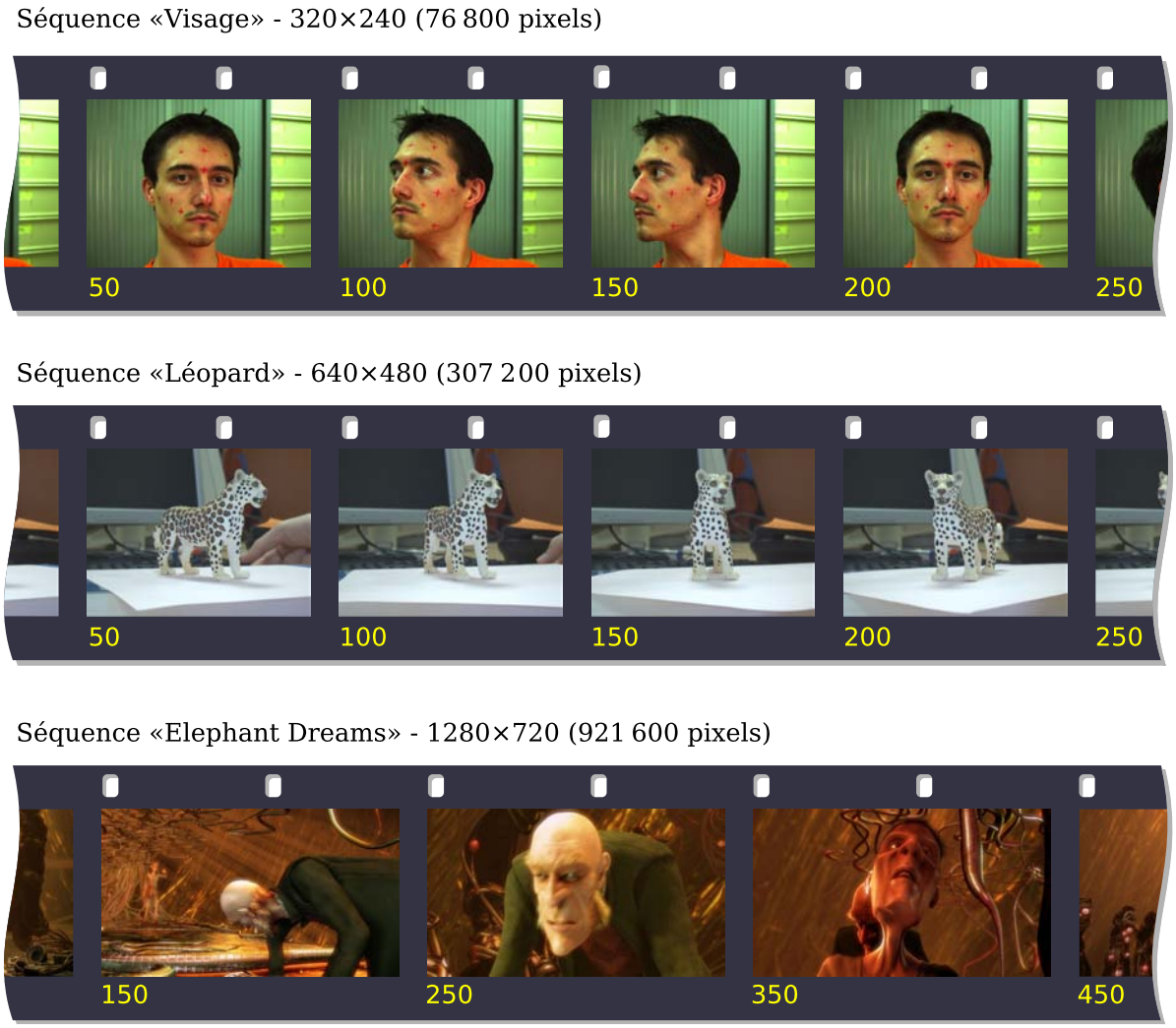


FIG. 7.5 – Les trois séquences de tests utilisées.

qui possède deux unités d'exécutions, les gains sont sans surprise encore supérieurs (en moyenne $1,8$ fois plus rapide).

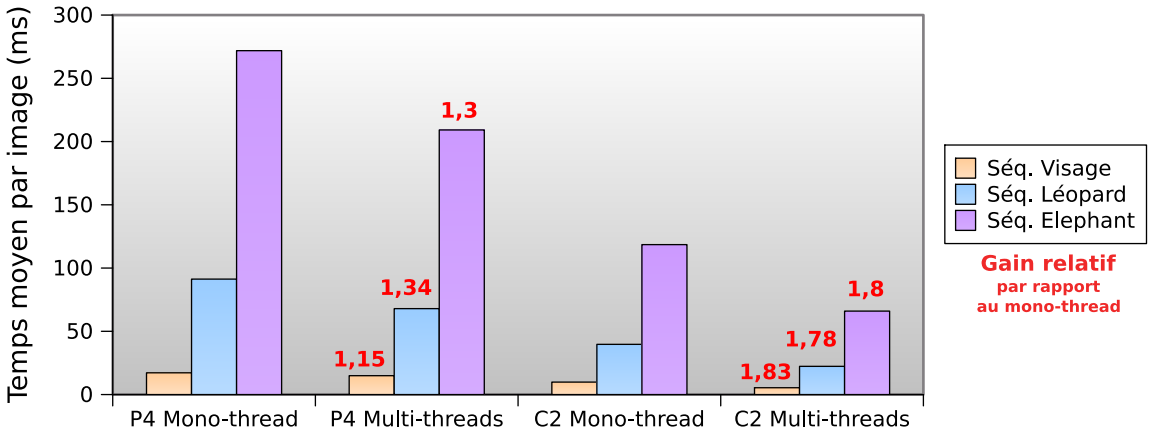


FIG. 7.6 – Performances des versions mono et multi *threads* de l'algorithme d'extraction des points d'intérêt. La machine équipée d'un processeur Pentium 4 à 2.6Ghz est notée P4, celle équipée d'un processeur Core 2 Duo à 2.6Ghz est notée C2.

7.2.4 Implantation sur GPU

La mise en œuvre sur GPU de l'algorithme présenté en section 7.2.1 repose sur un certain nombre de noyaux de calcul que nous détaillons ici. En pratique ceux-ci sont écrits avec le langage GLSL, et les différents tampons image intermédiaires sont représentés par des textures. Notons que du fait de l'architecture particulière des GPU, il n'y a pas de correspondance un pour un entre les étapes de la figure 7.1 et ces noyaux. Nous détaillons la conversion de l'image en niveaux de gris, qui ne fait pas explicitement partie de l'algorithme, mais qui est aussi effectuée par le GPU et nous permet d'introduire le format de texture compact utilisé dans la suite.

Conversion d'une image en couleurs (espace RVB) vers une image en niveaux de gris La luminance L d'un pixel est calculé par la combinaison linéaire $L = 0.2125\,R + 0.7154\,V + 0.0721\,B$ qui correspond à un produit scalaire. La texture en sortie est une texture RVBA dans laquelle chaque pixel stocke 4 valeurs de luminance. La texture des niveaux de gris est donc quatre fois moins large que l'image d'origine, comme illustrée sur la figure 7.7.

Calcul des dérivées de l'image Il s'agit de réaliser 2 convolutions par les filtres dérivés $g_x(\cdot, \sigma_D)$ et $g_y(\cdot, \sigma_D)$. Ces filtres sont séparables, une convolution est donc réalisée en deux

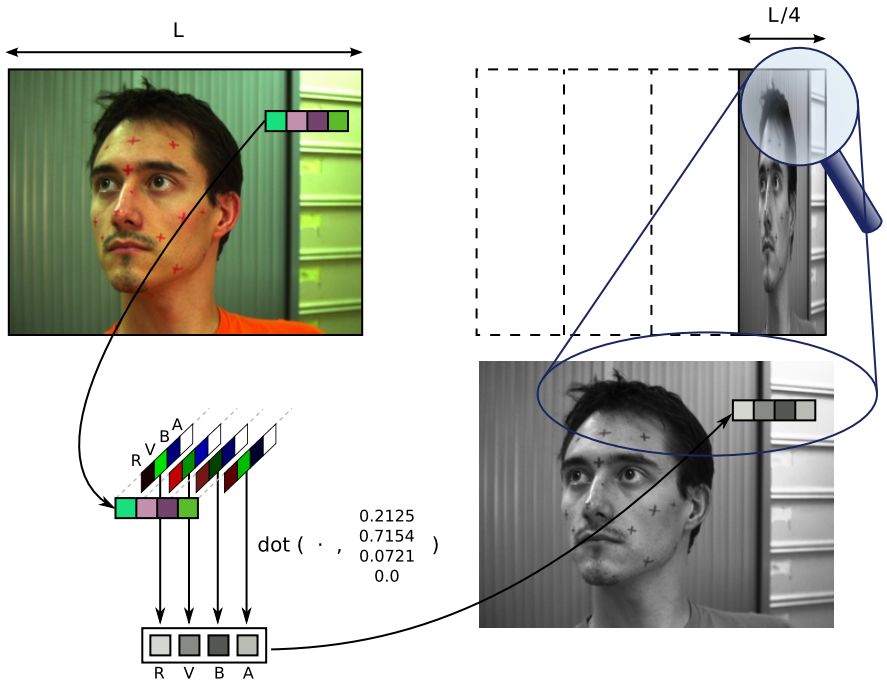


FIG. 7.7 – Conversion d'une image en couleurs (espace RVB) vers une image en niveaux de gris

passes avec des noyaux linéaires N_x et N_y, tronqués au delà de leur dernière valeur significative :

$$\text{En x :}\quad I'(x,y) = \sum_{i\in[\![-k;k]\!]} I_{\text{entrée}}(x+i,y)N_x(k+i)$$

$$\text{En y :}\quad I_{\text{sortie}}(x,y) = \sum_{i\in[\![-k;k]\!]} I'(x,y+i)N_y(k+i)$$

La passe en x exploite le format «compact» de l'image de luminance. Par exemple pour $\sigma_D = 0.7$, g_x est tronqué à un support de taille 5, ce qui permet de calculer quatre pixels avec seulement 3 accès à la texture source, 4×3 produits scalaires et 1 somme vectorielle. La figure 7.8 résume ce schéma.

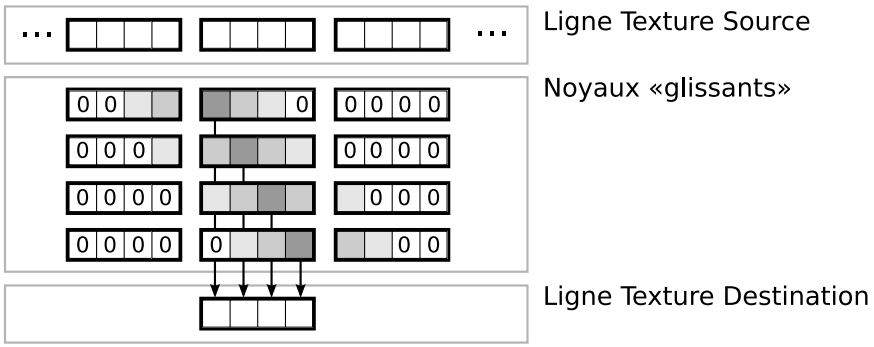


FIG. 7.8 – Convolution rapide avec un noyau linéaire horizontal de taille 5.

La passe en y effectue $2k+1$ accès à la texture source autour du pixel destination. Les traitements vectoriels du GPU et l'organisation compacte de la texture source, permettent de calculer 4 pixels en parallèle, comme illustré sur la figure 7.9.

Calcul des éléments de la matrice d'auto-corrélation Ce calcul comporte deux étapes :

1. Calcul des produits. Le *shader* correspondant à cette passe utilise en entrée les deux textures contenant les dérivées I_x et I_y de l'image en niveau de gris et calcule en sortie les produits pixels à pixels I_x^2, I_y^2 et I_xI_y, chacun étant dirigé vers une texture différente. Ici encore la nature compacte des textures permet de calculer 4 pixels en une opération.
2. Lissage. Les 3 textures précédemment calculées sont lissées par le filtre $g(\cdot,\sigma_I)$, en exploitant les mêmes optimisations que celles présentées plus haut pour le calcul des images dérivées.

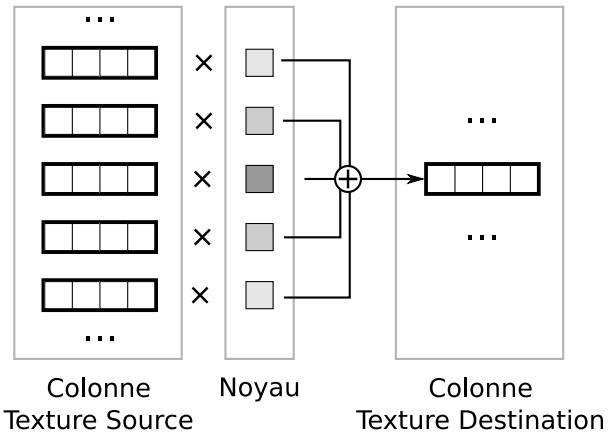


FIG. 7.9 – Convolution rapide avec un noyau linéaire vertical de taille 5.

Calcul de la carte de saillance Le critère de Harris est calculé selon la formule 7.1. Au total 4 multiplications et 3 additions sont nécessaires pour calculer 4 valeurs du critère (4 pixels consécutifs verticalement). On applique enfin une passe de suppression des points qui ne sont pas des maxima locaux, afin d'éviter les amas de points lors de l'extraction. Nous prenons comme voisinage local les 8 pixels autour du point courant, cette passe nécessite donc au total 9 accès à la texture en entrée.

Extraction des points Cette étape est effectuée sur le CPU. Les deux stratégies présentées en section 7.2.1 sont possibles, c'est-à-dire soit un seuillage soit un tri partiel de la carte de saillance. Récemment, Ziegler et al. ont proposé un algorithme de génération de liste de points sur le GPU [ZTTS06] et Wu a adapté son algorithme GPUSIFT [SFPG06] pour utiliser cet algorithme [Wu06]. L'avantage principal de cette approche est de réduire la taille des données à télécharger depuis la carte graphique, puisque la liste des points est beaucoup plus légère que la carte de saillance.

7.2.5 Résultats

Dans cette section nous présentons les temps de calculs mesurés sur trois cartes graphiques équipées de processeurs différents (du plus ancien au plus récent, et du moins puissant au plus puissant) : un NV40 sur bus AGP , un G71 et un G80 sur bus PCI-X. La figure 7.10 montre que les gains sont très significatifs grâce aux dernières générations de processeurs graphiques par rapport à la génération actuelle de processeurs généralistes. Les temps sur le G80 correspondent à des cadences d'image de respectivement 53, 173 et 674 images par seconde.

Le graphique de la figure 7.11 détaille le temps passé à télécharger la carte de saillance calculé sur le GPU vers le CPU, mesuré sur le G80. On constate que cette opération peut

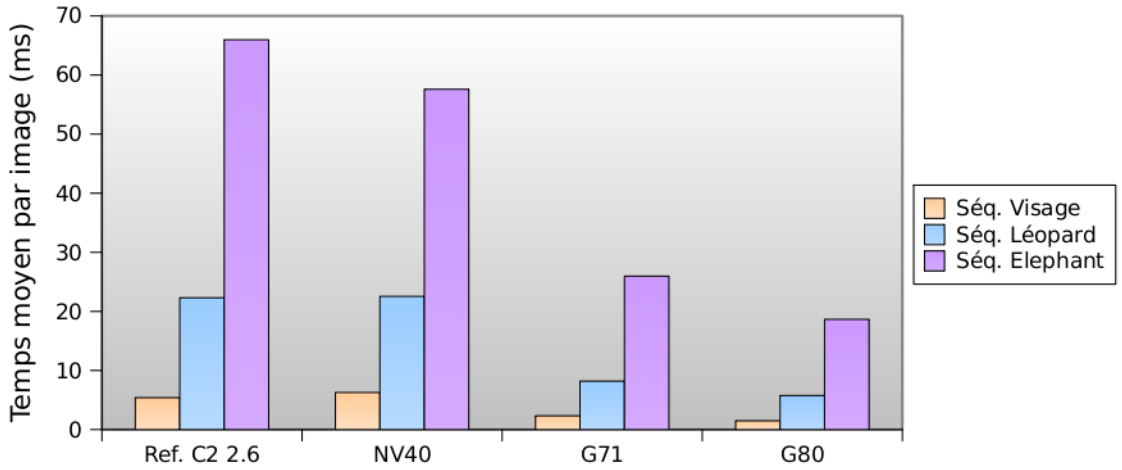


FIG. 7.10 – Temps de calcul de la carte de saillance sur les différents GPU, comparés à la valeur de référence sur CPU.

consommer une partie non négligeable du temps total, notamment lorsque la taille des images devient importante : pour la séquence «Elephant Dream» le temps de téléchargement représente 42% du temps de calcul total. Ceci montre tout l'intérêt de la technique de génération des points sur le GPU évoquée plus haut. En effet cette liste est de taille beaucoup plus petite que la carte de saillance, le temps de transfert gagné compense alors le coût supplémentaire lié à la création de la liste (par ailleurs déchargée du CPU).

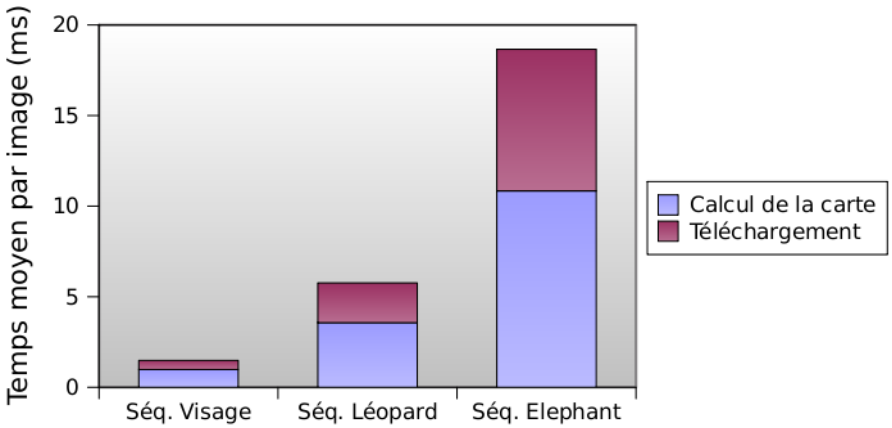


FIG. 7.11 – Détail de la part du temps de téléchargement dans le calcul total de la carte de saillance (exécution sur le G80).

On peut limiter l'impact de ce temps de téléchargement grâce à l'utilisation d'un mé-

canisme de transfert asynchrone, c'est-à-dire n'impliquant pas le CPU[4]. Dans ce mécanisme appelé *DMA (Direct Memory Access)*, un contrôleur externe (ici la carte graphique) prend en charge l'écriture des données en mémoire centrale.

La figure 7.12 illustre la mise en œuvre de ce principe, qui repose sur la décomposition du transfert en deux commandes : l'une, non-bloquante, qui initie le téléchargement (directive OpenGL `glReadPixels()`, l'autre, bloquante, qui synchronise le tampon en mémoire centrale avec sa version en mémoire de la carte graphique, autrement dit qui attend que le transfert soit terminé (directive OpenGL `glMapBuffer()`). Entre ces deux instructions, le CPU est libéré pour d'autres tâches ; on masque ainsi le temps de téléchargement par du calcul. Un inconvénient de cette synchronisation explicite est la difficulté de déterminer le nombre de tâches à placer entre ces deux appels. Les applications *multi-threads* offre une solution naturelle, puisqu'alors l'appel bloquant `glMapBuffer()` passe le *thread* appelant en arrière plan et un autre *thread* acquiert les ressources du CPU.

Un mécanisme similaire est théoriquement possible pour le chargement des textures vers le GPU. Cependant, dans notre cas une conversion de données est nécessaire (*RVG* vers *RVBA*) et cette conversion est nécessairement faite par le CPU.

Bilan Au vu des résultats présentés ci-dessus, on constate que le détecteur de Harris est extrêmement rapide sur le GPU. Même pour des séquences haute définition, le coût de calcul représente moins de 50 % du temps disponible par image (sur une base de 25 images par seconde). Nous avons donc retenu cette mise en œuvre pour notre algorithme de suivi, ce qui permet par ailleurs de décharger le CPU pour les calculs nécessaires à l'estimation de la pose de l'objet.

7.3 Appariement des points extraits dans deux images

L'algorithme de suivi présenté au chapitre 5 utilise abondamment l'estimation des déplacements inter-images : dans sa phase «itérative», les déplacements mesurés sont comparés aux déplacements théoriques interpolés grâce au modèle de l'objet ; dans sa phase utilisant une image clé, les déplacements fournissent les correspondances 2D-3D permettant l'estimation de la pose de l'objet. Ces mesures sont obtenues en recherchant les points homologues entre deux images successives ou entre une image et une image clé.

Le principe général pour établir des appariements est de définir un descripteur local du signal image dans une petite région autour d'un point. Un critère de similarité C permet ensuite de mesurer la ressemblance de deux points donnés. Les points homologues correspondent idéalement aux images d'un même point 3D de l'objet observé. La transformation géométrique subit par la scène entre les deux vues induit une déformation locale des voisinages des points. Un bon descripteur doit par conséquent être le plus stable possible vis à vis de ces déformations, autrement dit il doit fournir une réponse similaire pour les deux

[4]Pour cela il faut que le format des données (type et organisation) sur le CPU et sur le GPU soit le même, pour que le CPU n'ait pas a effectuer de conversion.

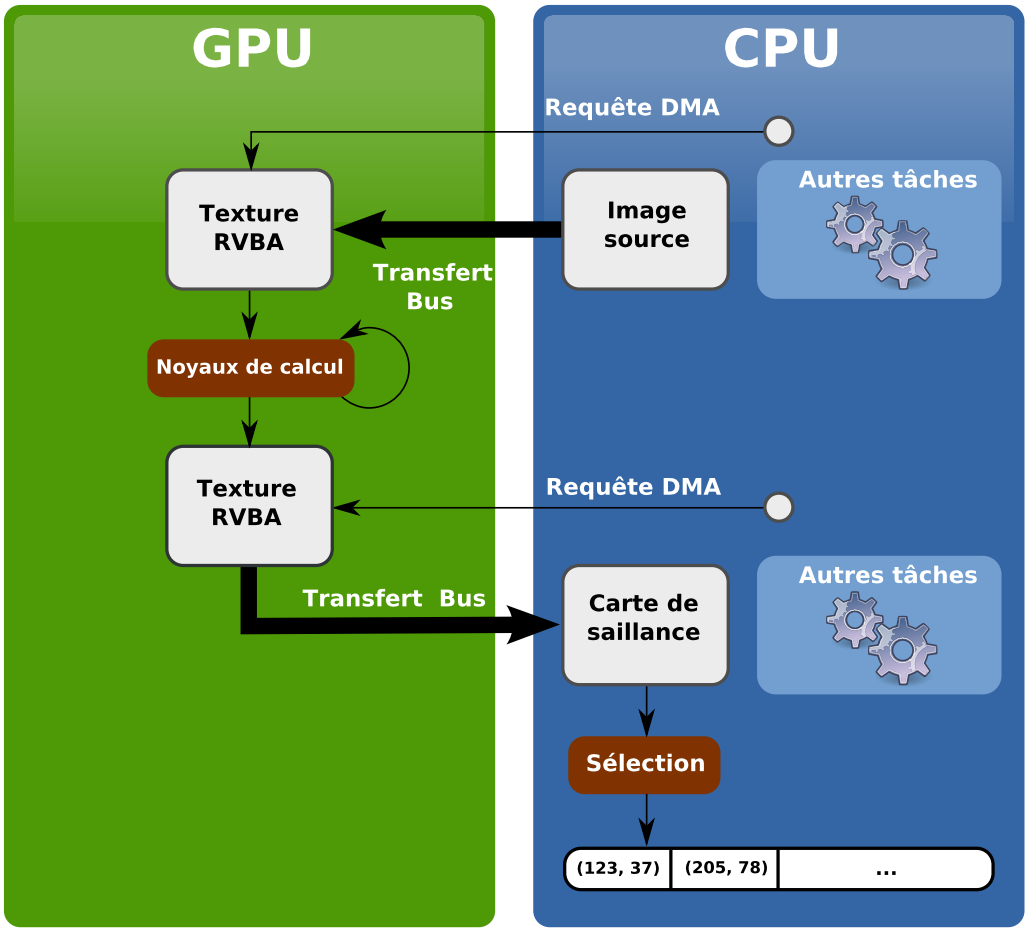


FIG. 7.12 – Transferts mémoire entre le CPU et le GPU. Lorsque le mode DMA est supporté, les transferts n'impliquent pas le CPU qui peut réaliser d'autres tâches.

points images. Il est également souhaitable que la réponse soit la plus indépendante possible des variations d'illumination, ce qu'on obtient généralement en normalisant le signal image autour des points considérés.

Pour un point 3D situé sur une surface localement plane et vue par une caméra orthographique, la déformation géométrique locale dans l'image (entre les voisinages de 2 points images) est une transformation affine [HZ03, chapitre 13]. Cette situation idéale est une bonne approximation pour de nombreux cas réels, ce qui motive la recherche de descripteurs invariants à ce type de déformations. Dans l'article déjà cité [MS01], Mikolajczyk et Schmid utilisent les dérivées jusqu'à l'ordre 4 du signal image, calculées avec un filtre gaussien adaptés à l'échelle estimée du point d'intérêt. L'invariance à la rotation est obtenue en «pivotant» le repère local afin de l'aligner avec la direction du gradient. L'invariance aux transformations affines est obtenue en divisant les valeurs des dérivées par celle de la dérivée première. Chaque point est ainsi décrit par un vecteur de dimension 12 et une distance de Mahalanobis est utilisée pour les comparer.

Pour la méthode SIFT, Lowe [Low04] propose de découper une petite fenêtre autour du point en sous-blocs. Les histogrammes des directions du gradient de l'image calculés sur chacun de ces blocs sont utilisés pour définir le descripteur.

Citons enfin l'approche par apprentissage proposée par Lepetit [LLF05], qui considère la recherche des points homologues comme un problème de classification. Une classe, associée à un point d'intérêt particulier, correspond aux vues possibles de petites fenêtres autour d'un point. La phase d'apprentissage collecte un grand nombre de ces vues en générant, à partir d'une ou plusieurs images réelles d'un objet, un ensemble d'images synthétiques (créées à l'aide d'un modèle texturé lorsque celui-ci est disponible ou en déformant de manière affine les fenêtres autour des points). Le nombre de paramètres décrivant chaque classe est ensuite réduit par une Analyse en Composantes Principales de l'ensemble des fenêtres de chaque classe. Pour trouver l'homologue d'un point donné dans une nouvelle vue de l'objet, un classifieur effectue une recherche du plus proche voisin dans l'ensemble des classes. La mise en œuvre proposée par les auteurs est suffisamment rapide pour être utilisée par exemple comme procédure de récupération dans les cas de divergence d'un algorithme de suivi.

Si l'invariance aux transformations affines, éventuellement de grande amplitude, est particulièrement intéressante pour les applications telles que la détection d'objets ou la modélisation 3D à partir d'images, leur intérêt est moins évident dans le cas du suivi itératif, où les images successives ne présente que de petites déformations. Par ailleurs aucune des méthodes présentées plus haut n'est encore suffisamment rapide pour le suivi temps réel. Nous avons donc choisi d'utiliser une approche plus simple, utilisant une mesure de corrélation des voisinages des points. La section suivante en détaille le principe.

7.3.1 Principe

Nous appelons image gauche (I_g) et image droite (I_d) les deux images en niveaux de gris dans lesquelles nous recherchons des points homologues. Un ensemble de points d'intérêt est extrait de chaque image avec le détecteur présenté en section 7.2.

Une mesure de similarité possible est la somme des différences au carré (SDC) des pixels d'une fenêtre carrée de largeur $w = 2m + 1$. Soit $f_d^i \in \mathbb{R}^{n^2}$ le vecteur contenant les valeurs de niveaux de gris de l'image I_d dans la fenêtre autour d'un point $\mathbf{p}_i$. On définit de même f_g^j autour du point $\mathbf{p}_j$ dans I_g. Le critère SDC s'écrit alors :

$$C_{SDC}(\mathbf{p}_i, \mathbf{p}_j) = f_d^i - {f_g^j}^\top f_d^i - f_g^j$$

Ou, de manière détaillée :

$$C_{SDC}(\mathbf{p}_i, \mathbf{p}_j) = \sum_{(k,l)\in[\![-m,m]\!]^2} (I_d(\mathbf{p}_i + (k,l)^\top) - I_g(\mathbf{p}_j + (k,l)^\top))^2$$

Cette mesure est très efficace mais elle est sensible aux changements d'intensité (biais ou gain). Une autre mesure, appelée corrélation croisée centrée et normalisée (CCCN) pallie à ces défauts, en opérant un centrage et une normalisation locale du signal image contenu dans f_g^j et f_d^i. Le critère CCCN s'écrit :

$$C_{CCCN}(\mathbf{p}_i, \mathbf{p}_j) = \frac{(f_d^i - \overline{f_d^i})^\top (f_g^j - \overline{f_g^j})}{\|f_d^i - \overline{f_d^i}\| \|f_g^j - \overline{f_g^j}\|}$$

Ou encore :

$$C_{CCCN}(\mathbf{p}_i, \mathbf{p}_j) = \sum_{(k,l)\in[\![-m,m]\!]^2} \frac{(I_d(\mathbf{p}_i + (k,l)^\top) - \overline{I_d})(I_g(\mathbf{p}_j + (k,l)^\top) - \overline{I_g})}{m^4\ \sigma_d^2\ \sigma_g^2}$$

où $\overline{I_g}$ et σ_g^2 (resp. $\overline{I_d}$ et σ_d^2) sont la moyenne et l'écart-type des niveaux de gris de la fenêtre gauche (resp. droite).

Le critère CCCN a des valeurs entre 0 et 1. Un point est d'autant plus similaire à un autre que le critère SDC est petit ou que le critère CCCN est grand.

Pour chaque point d'intérêt de l'image gauche, on recherche l'homologue parmi tous les points de l'image droite. Si le score de similarité du meilleur point est supérieur à un seuil, alors les deux points sont considérés comme homologues.

Nous utilisons deux améliorations classiques de ce schéma :

1. en faisant l'hypothèse que le mouvement entre les images est petit, on peut limiter l'ensemble des candidats testés à *une zone de recherche*, par exemple à ceux contenus dans un disque autour du point à apparier. On réduit ainsi considérablement le nombre d'évaluations du critère de similarité ;
2. on peut également inverser le rôle des images gauche et droite, répéter la procédure de recherche des points homologues et ne conserver que les appariements cohérents, c'est-à-dire ceux pour lequel l'homologue d'un point $\mathbf{p}$ a pour homologue $\mathbf{p}$ lui-même. Cette étape est appelée *validation croisée*.

Ces deux améliorations permettent de réduire le nombre d'appariements erronés, notamment ceux dû à la présence de motifs similaires dans l'image. Il faut cependant noter que limiter la zone de recherche signifie également renoncer à détecter des déplacements d'amplitude supérieure au rayon de cette zone. Nous verrons que sur le GPU (voir section suivante) cette restriction n'est pas nécessaire pour améliorer les performances.

7.3.2 Implantation sur le GPU

Pour présenter les adaptations de l'algorithme d'appariement nécessaires à sa mise en œuvre sur le GPU, nous distinguons deux grandes étapes : la première, illustrée par la figure 7.13 consiste à charger les points d'intérêt obtenus en section 7.2.4. La deuxième étape consiste à calculer une carte de similarité, à partir de laquelle les appariements de l'image gauche vers l'image droite (et inversement) sont obtenus. Cette étape, illustrée en figure 7.15, est terminée par la procédure de validation croisée.

Chargement des points d'intérêt

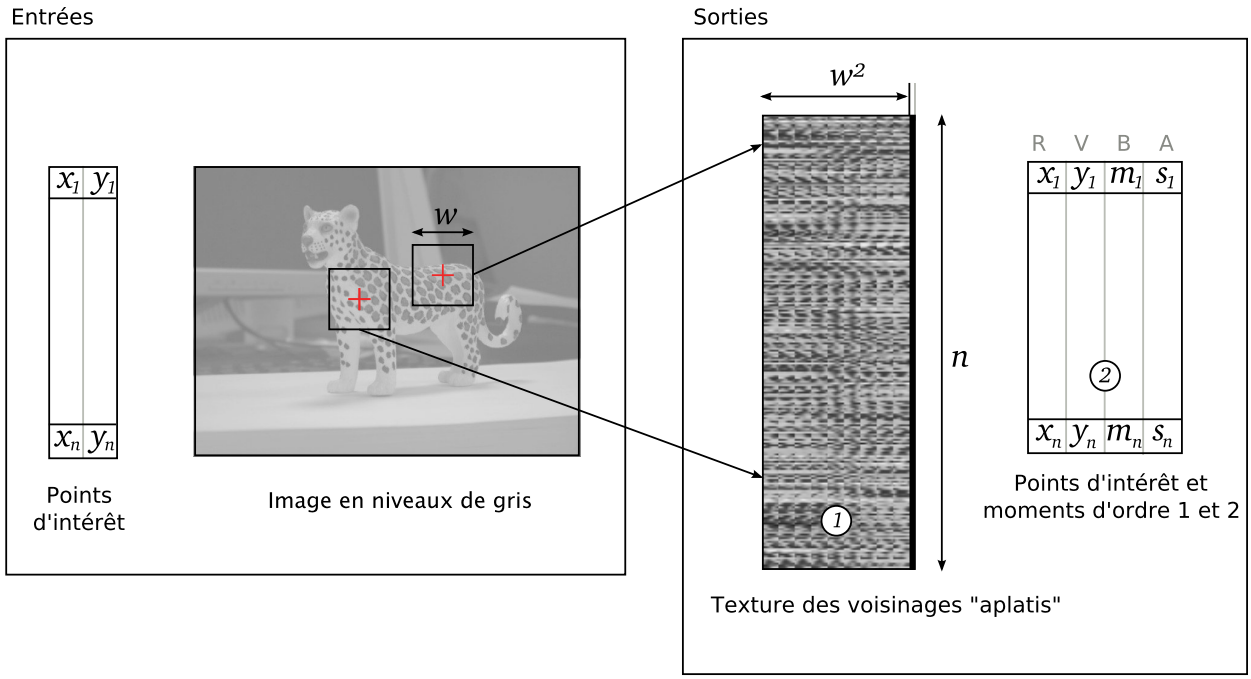


FIG. 7.13 – Transferts des points d'intérêt vers le GPU : construction d'une texture des voisinages "aplatis" en ligne (1) et calculs des moments d'ordre 1 et 2 (2).

Dans une approche que nous appelons «naïve», seul le tableau contenant les points d'intérêt et les moments d'ordre 1 et 2 (moyenne et écart-type) des voisinages est calculé, sous la forme d'une texture ligne (indiquée par (2) sur la figure 7.13). Ceci n'est nécessaire que dans le cas où le critère de similarité est CCCN.

Dans une approche «optimisée», une texture intermédiaire, que nous appelons *texture des voisinages «aplatis»* (indiquée par (1) sur la figure 7.13) est également générée. Cette texture est construite à partir de l'image en niveaux de gris et d'une texture contenant les positions des points d'intérêt. Autour de chaque point, une fenêtre de taille w^2 est

réorganisée en une ligne de w^2 pixels[5].

L'étape de calcul des critères de similarité (voir plus loin) n'est formellement pas dépendante de l'organisation des fenêtres de voisinages, seule l'accès doit est adapté. Nous avons comparé la procédure d'appariement dans sa version naïve, où les valeurs des voisinages sont lues directement dans l'image en niveau de gris et la version optimisée où ces valeurs sont lues en ligne dans la texture des voisinages aplatis. Les performances de chacune sont reportées sur la figure 7.14. Comme le montre le graphique les gains apportés par la version optimisée par rapport à la version naïve vont de 50 à 70 %.

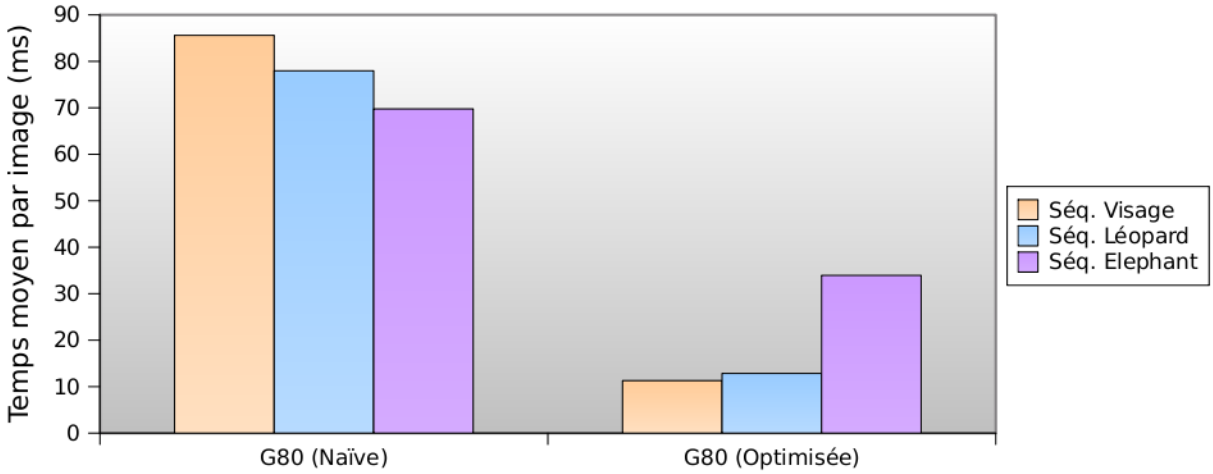


FIG. 7.14 – Comparaison des performances des approches naïve (N) et optimisée (O) de l'algorithme de mise en correspondance de 500 points d'intérêt.

Nous pensons que les mauvaises performances de la version naïve sont liées à un schéma d'accès à la mémoire sous-optimal, puisque les voisinages sont «éparpillés» dans les images. Au contraire, utiliser une représentation compacte des voisinages permet des accès plus locaux et donc mieux adaptés aux optimisations de l'architecture des GPU.

Calcul de la carte de similarité et validation croisée

La deuxième étape évalue, à partir des textures calculées précédemment pour deux images gauche et droite, une carte de similarité encodant la similarité entre chaque paire de points d'intérêt (voi la figure 7.15). Les optimums sur chaque ligne et chaque colonne permettent d'obtenir les points homologues.

Un premier noyau de calcul évalue le critère de similarité pour tous les couples de points. Contrairement à la version CPU, nos tests ont montré que le temps de calculs ne dépend

[5]En pratique la texture obtenue a une forme compacte telle que celle de l'image en niveaux de gris construite en section 7.2.4. Sa largeur est donc un multiple de 4. Par ailleurs w est impair : $w = 2p + 1$, donc $w^2 = 4k + 1 = 4(k + 1) - 3$; les 3 pixels inutilisés sont représentés par la bordure noire à droite dans la figure 7.13.

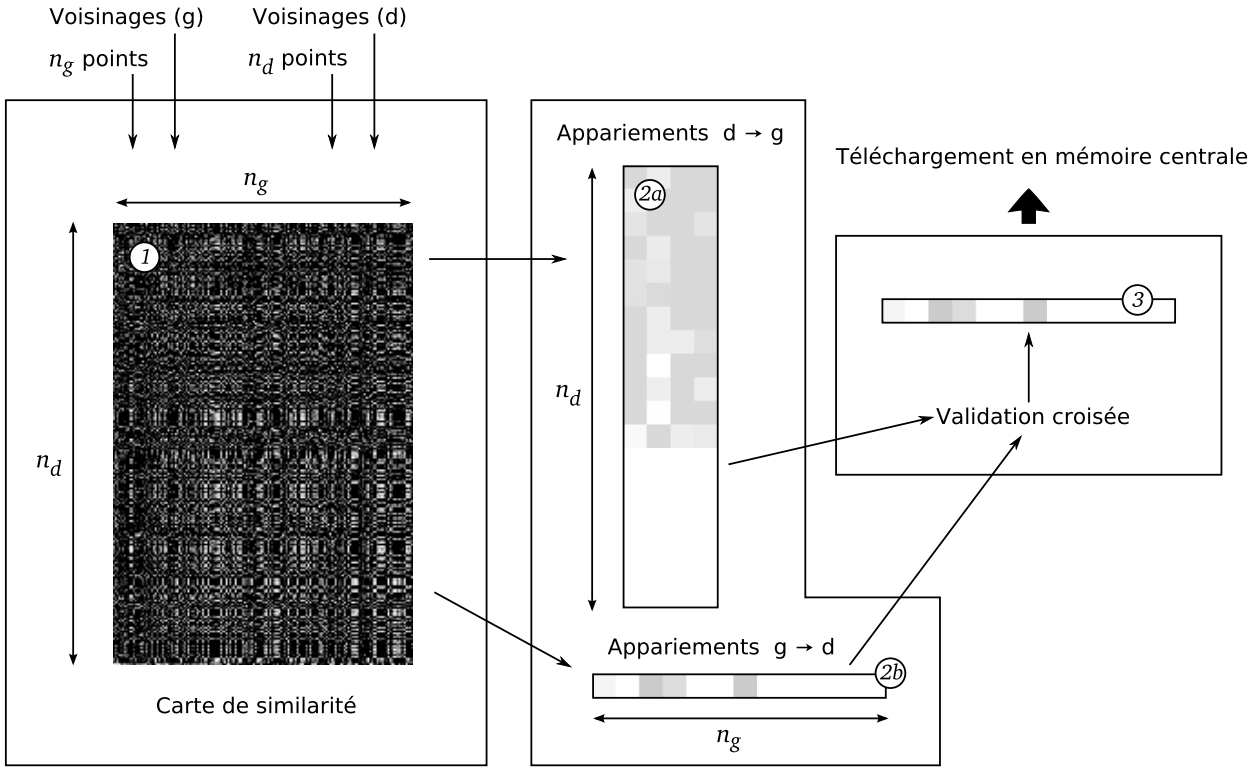


FIG. 7.15 – Calcul de la carte de similarité (1), calcul des meilleurs appariements de l'image précédente (resp. suivante) vers l'image précédente (resp .suivante) (2a) (resp. 2b), validation croisée (3).

pas du rayon de la zone de recherche. Par conséquent nous supprimons complètement cette étape pour éviter un branchement conditionnel sans intérêt. La texture de la carte de corrélation est compacte : chaque pixel stocke quatre évaluations du critère. Ainsi le traitement du fragment de coordonnées de textures (i, j) évalue les critères de corrélations $C(\mathbf{p}_i, \mathbf{p}_{4j}), C(\mathbf{p}_i, \mathbf{p}_{4j+1}), C(\mathbf{p}_i, \mathbf{p}_{4j+2})$ et $C(\mathbf{p}_i, \mathbf{p}_{4j+3})$.

Une fois la carte de similarité calculée, il faut déterminer les homologues et faire la validation croisée. Calculer le meilleur candidat dans l'image gauche d'un point $\mathbf{p}_i$ de l'image droite correspond à calculer l'optimum du critère le long de la ligne numéro i de la carte de similarité. De même, calculer le meilleur candidat d'un point $\mathbf{p}_j$ de l'image droite correspond à calculer l'optimum du critère le long de la colonne numéro j. Ces opérations, notées (2a) et (2b) sur la figure 7.15) mettent en oeuvre une opération classique en programmation *GPGPU*, appelée *réduction* et illustrée en détail sur la figure 7.16 pour le cas de la réduction verticale. Une opération de réduction est une opération pour laquelle la taille du tampon de sortie est successivement réduite d'un facteur fixe par rapport à celle du tampon d'entrée. Elle permet d'implanter des boucles de grandes tailles efficacement. Un exemple classique est la création des différentes échelles de textures utilisant la technique de *mipmapping*. Le noyau de calcul d'une telle opération rassemble plusieurs valeurs en entrée pour produire une valeur en sortie. On répète l'opération en échangeant les tampons d'entrée et de sortie et en réduisant d'un facteur la taille de la zone écrite, jusqu'à obtenir un tampon de sortie de taille 1 pixel dans une ou dans toutes ses dimensions.

La texture de la carte de similarité est réduite verticalement par quatre à chaque étape, jusqu'à n'obtenir qu'une seule colonne de 4 valeurs de similarité (une par composante RVBA). La plus grande valeur (ou la plus petite selon le critère) des quatre valeurs sur la ligne i correspond au meilleur candidat pour le point $\mathbf{p}_i$[6]. Dans le cas ou le critère est CCCN, l'optimum correspond au maximum du critère, on réalise ainsi le calcul suivant pour chaque ligne :

$$C^{\star}_{CCCN}(\mathbf{p}_i) = \max_{j=1,\ldots,n_{t-1}} C_{CCCN}(\mathbf{p}_i, \mathbf{p}_j)$$

L'opération (2b) est similaire, il s'agit d'une réduction horizontale permettant de calculer, pour chaque colonne j :

$$C^{\star}_{CCCN}(\mathbf{p}_j) = \max_{i=1,\ldots,n_t} C_{CCCN}(\mathbf{p}_i, \mathbf{p}_j)$$

Plus que la valeur du critère de similarité du meilleur correspondant, c'est son indice dans la liste des points qui est nécessaire. Parmi les solutions possibles, on pourrait définir un autre arrangement des composantes des pixels de la carte de similarité (par exemple intercaler valeur du critère et indice dans un fragment RVBA selon le schéma $(C1, i1, C2, i2)$) ou encore utiliser un second tampon de sortie. Ces solutions ont comme inconvénient de nécessiter plus de place mémoire et d'interdire l'utilisation directe de l'instruction `max` et `min` des GPU. En particulier elles nécessitent de coûteux branchements conditionnels. Nous

[6]Pour des raisons pratiques, le calcul de l'optimum des 4 composantes est incorporé dans l'étape de validation croisée.

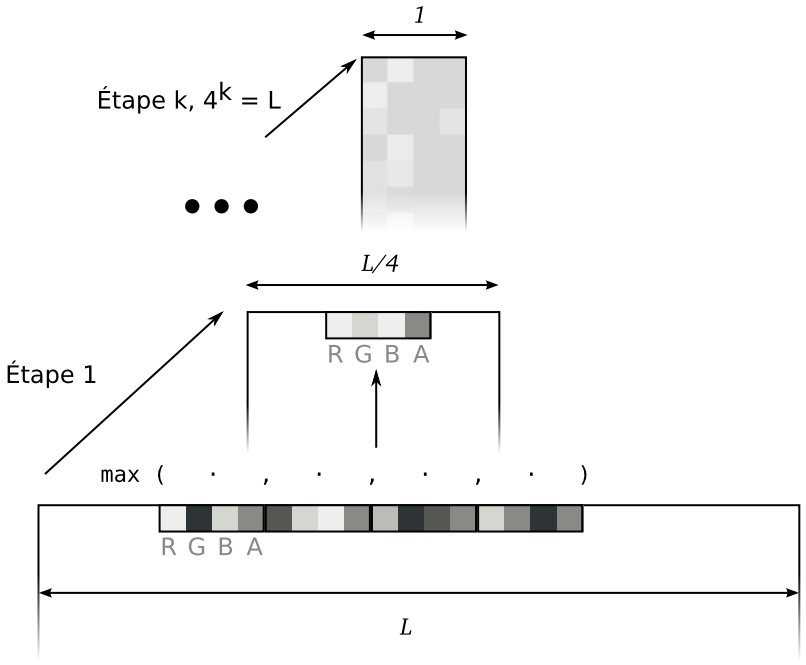


FIG. 7.16 – Réduction verticale de la carte de similarité pour obtenir le critère maximum par ligne (cas du critère CCCN).

proposons donc d'encoder l'indice dans la valeur du critère afin de le transporter naturellement tout au long des calculs, sans rien modifier à ce qui a été présenté précédemment. Pour permettre l'utilisation de l'instruction `max`, l'indice est encodé dans la partie après la virgule du nombre flottant représentant le critère. Pour encoder un indice couvrant l'intervalle $[0, 2^k]$, il faut k bits. Les nombres réels sur les GPU actuels sont codés selon le standard IEEE574, c'est-à-dire utilisent 23 bits pour la mantisse, 8 bits pour l'exposant et 1 bit pour le signe. Comme le codage doit conserver l'ordonnancement, nous ne pouvons pas utiliser le bit de signe. Par ailleurs pour garder un encodage peu coûteux à calculer, nous n'utilisons pas l'exposant (le nombre ainsi codé est donc à virgule fixe). Nous avons donc au total 23 bits à notre disposition dont $23 - k$ sont disponibles pour le codage de la valeur du critère de similarité. Ainsi k définit le nombre maximum de points appariables. Voici en pseudo code les opérations d'encodage d'un indice `i` dans la valeur `c` du critère CCCN (pour lequel $c \in [0, 1]$) :

```
scale_down = 1.0 / pow (2, k);
scale_up = pow (2, 23-k);

ci_encode = floor (c * scale_up) + i * scale_down;
```

Le terme `floor (c * scale_up)` correspond à la partie avant la virgule du résultat et le terme `i * scale_down` correspond à la partie après la virgule.

Le décodage est également très simple :

```
scale_up = pow (2, 23-k);

i_decode = floor (fract (ci_encode) * scale_up);
c_decode = floor (ci_encode) / scale_up;
```

Le choix de ne pas utiliser les bits de l'exposant permet d'écrire le codage et le décodage très simplement avec les opérations natives `floor (x)` (partie entière de `x`) et `fract (x)` (partie fractionnaire de `x`), au prix d'une perte de 8 bits de précision. Nous utilisons $k = 11$, ce qui permet d'apparier des listes d'au plus 2048 points (ce qui ne pose pas de problème dans notre contexte). La précision restante de 12 bits pour la valeur du critère lui-même offre une capacité de séparation suffisante en pratique.

Grâce au codage précédent, la dernière étape de validation croisée est très simple : le noyau de calcul s'applique à la texture ligne (2b)[7] et produit en sortie un tampon de même taille. L'indice $i^\star$ du meilleur candidat est extrait de la valeur $c_j^\star$ en colonne j. Le fragment optimal de la ligne $i^\star$ de la texture (2b) est alors décodé et donne un meilleur candidat d'indice $j^\star$. L'appariement est validé lorsque $j^\star = j$ (voir la figure 7.17).

[7] Ce choix (au dépend du parcours de la texture colonne (2a)) est arbitraire.

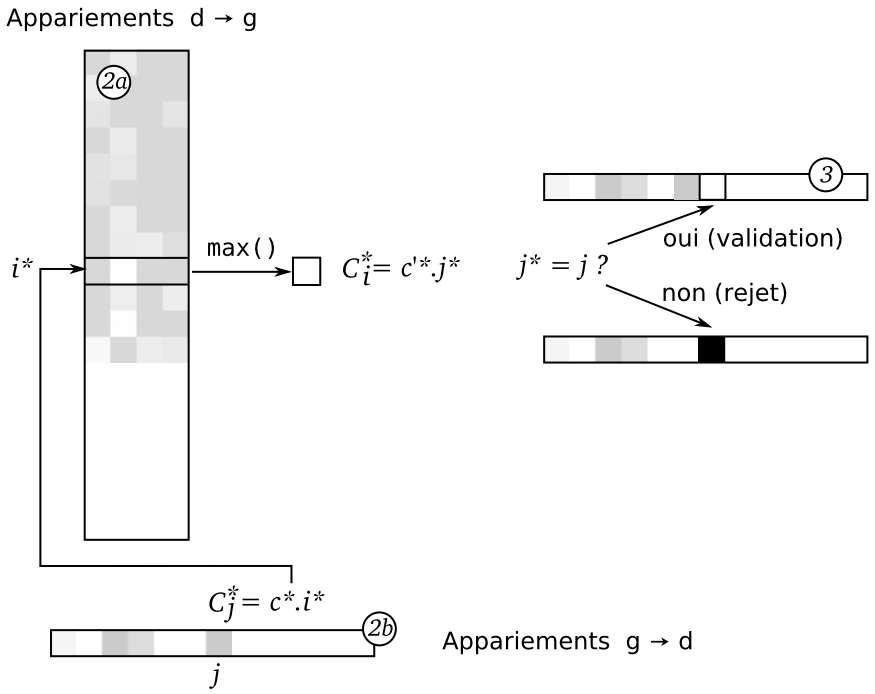


FIG. 7.17 – Détails de l'étape de validation croisée.

7.3.3 Résultats

La figure 7.18 compare les temps de calcul moyens par image des mises en œuvre sur CPU et sur GPU de l'algorithme de mise en correspondance des points. Le nombre de points est fixé à 900 pour toutes les images, afin d'avoir une complexité pour la recherche des appariements constante tout au long de la séquence. Le rayon de la zone de recherche est de 30 pixels pour l'exécution sur le CPU. La mise en œuvre sur le GPU présente la caractéristique intéressante de ne pas dépendre de la taille de la zone de recherche. Par conséquent nous avons complètement supprimé le test de proximité, ce qui revient à considérer tous les points comme candidats potentiels. Ceci est un avantage incontestable lorsque le déplacement inter-image peut être arbitrairement grand. Dans le cas de notre algorithme de suivi, cela peut arriver en particulier lors de l'appariement avec une image clé. On peut ainsi fournir une initialisation moins précise et récupérer une pose plus facilement lorsque le suivi itératif échoue.

Les gains de la mise en œuvre sur le GPU par rapport à celle sur CPU sont globalement moins importants que pour l'algorithme de détection. Le GPU est même moins bon qe le CPU pour la séquence en haute définition. Le caractère intrinsèquement itératif de l'algorithme et les accès mémoires non séquentiels (bien que réduits dans notre version optimisée) sont à l'origine de ces résultats.

Nous utilisons la mise en œuvre sur GPU dans notre algorithme de suivi.

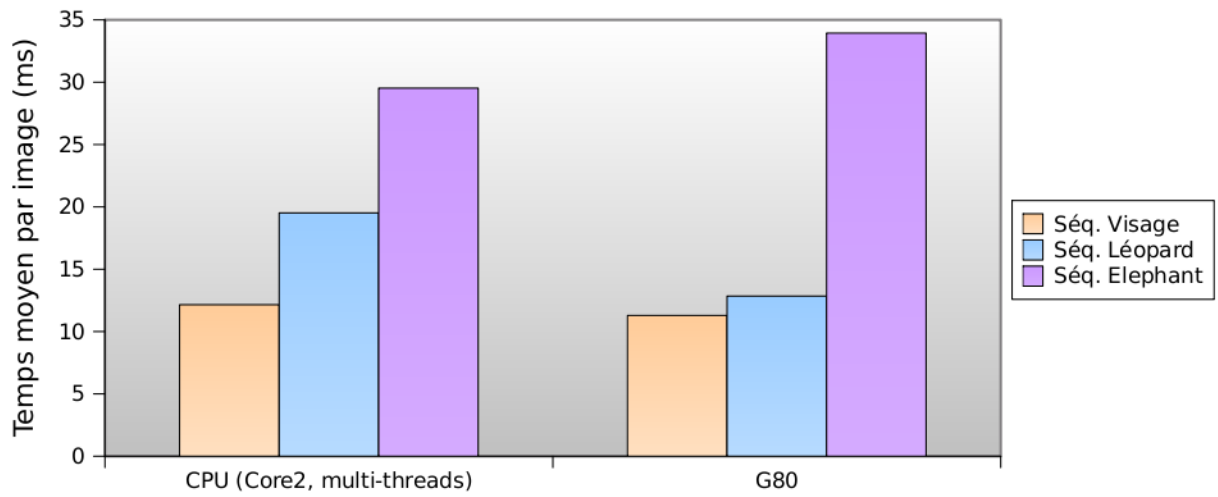


FIG. 7.18 – Performances comparées des mises en œuvre CPU et GPU de l'algorithme de mise en correspondance.

7.4 Bilan et contribution

Ce chapitre avait pour but d'étudier les gains de performances apportés par deux approches utilisables sur les calculateurs modernes : d'une part l'utilisation parallèle des multiples unités de calculs des processeurs centraux et d'autre part l'exploitation de l'importante puissante de calculs offerte par les processeurs graphiques.

Nous avons montré que des gains conséquents peuvent être obtenus dans les deux approches, avec cependant un ordre de grandeur supérieur lorsque le GPU est bien exploité. La figure 7.19 présente les temps par image pour les deux algorithmes cumulés (détection et appariement), dans leur mise en œuvre sur CPU et sur GPU. Les lignes rouges symbolisent le temps disponible par image pour des objectifs de suivi en temps interactif et en temps réel. La mise en œuvre sur GPU permet approximativement de doubler les performances. Dans le cas de la séquence «Léopard» et sur CPU, la totalité du temps imparti par image pour une cadence de 25 images par seconde est consommée par la détection et l'appariement des points. Sur GPU, seulement 45% du temps est nécessaire.

Il convient pour exploiter la puissance combinée des processeurs centraux et des processeurs graphiques de répartir les tâches de sorte que les deux types d'unités travaillent en parallèle. Ceci complique beaucoup la conception des programmes, et nous pensons que seule l'arrivée de nouvelles bibliothèques voire de nouveaux paradigmes de programmation permettra l'exploitation systématique de toute la puissance de calcul disponible sur les calculateurs actuels et à venir.

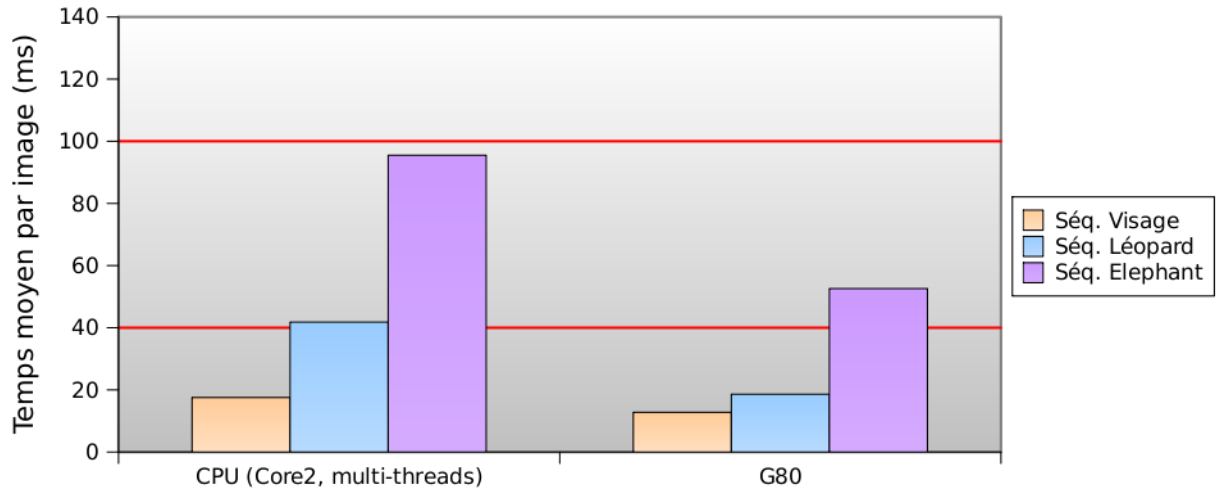


FIG. 7.19 – Performances de la procédure globale d'extraction et de mise en correspondance des points d'intérêt. Les deux lignes rouges indiquent les temps correspondants à l'objectif de temps interactif et à une cadence de 25 images par seconde.

Chapitre 8

Conclusion

Dans cette thèse, nous avons abordé quelques unes des problématiques de vision par ordinateur soulevées par les applications de Réalité Augmentée.

Les contributions que nous relevons de nos travaux sont :

- la proposition de deux applications de Réalité Augmentée simples,
- le développement d'un algorithme de suivi visuel 3D à partir d'un modèle à base de points,
- une mise en œuvre temps-réel de cet algorithme exploitant le GPU.

Applications de Réalité Augmentée 2D Les deux applications présentées au chapitre 2 mettent en lumière la diversité des problématiques de Réalité Augmentée. Celle (à laquelle nous avons contribué) mettant en œuvre une technique de suivi de fond dans le cadre restreint d'une caméra à centre optique fixe propose une solution intéressante à la question de la création des scènes augmentées, grâce à une représentation unifiée du décor de la scène et des plans de composition. Pour la seconde (que nous avons entièrement développée), nous nous sommes concentrés sur le recalage de données de natures différentes (photographies et relevés à main levée) et sur le mécanisme d'interaction avec l'utilisateur.

Algorithme de suivi 3D utilisant un nuage de points La contribution majeure de ce travail est le développement d'un algorithme de suivi visuel 3D utilisant comme modèle de l'objet à suivre un nuage de points. Au chapitre 5 nous avons présenté les difficultés théoriques liées à l'utilisation d'un tel modèle et nous proposons une solution permettant un suivi hybride robuste, précis et non affecté par le phénomène de dérive.

Nous nous sommes placés dans le cadre général de l'application des modèles, des méthodes et des outils développés par la communauté de l'informatique graphique pour le traitement des nuages de points. Notre contribution à la transposition de ces outils au domaine de la vision par ordinateur repose sur trois points :

- le calcul de prédicteurs ponctuels de mouvement et la reconstruction dense dans l'espace image d'un champs de mouvement apparent ;
- une formulation linéaire itérative de la solution du problème d'estimation de la pose sur la base de ces prédicteurs ;

– l'utilisation d'un algorithme de rendu texturé d'un modèle par points pour générer de nouvelles vues à partir d'une image clé.

En introduction, nous avons affirmer que cette thèse défendrait le fait que les modèles utilisés pour le suivi 3D ne requièrent pas nécessairement d'informations topologiques comme celles apportées par un maillage. Nous pensons avoir apporté, au fil de ce manuscrit, des arguments théoriques, pratiques et expérimentaux appuyant cette «thèse» !

Mise en œuvre pour le temps réel L'intérêt majeur de nos travaux en suivi visuel 3D concerne la mise en œuvre efficace des algorithmes proposés pour satisfaire la contrainte du temps réel. Grâce à la mise en œuvre sur le GPU, il est possible d'obtenir des informations sur le mouvement 2D dense, à la résolution des images, en quelques dizaines de millisecondes.

Publications Les travaux présentés dans ce manuscrit sont l'objet de trois publications dans des conférences internationales. L'application exploitant le suivi de fond a été présentée à ICPR'04 [DDCM04]. Le dispositif intéractif utilisant un tableau magnétique a fait l'objet de la publication [DCC07] à EUROMEDIA'07. Enfin nos travaux sur le suivi visuel 3D temps réel utilisant les modèles par points ont été publiés en 2006 à VMV'06 [DCM06]. Par ailleurs nos dernières avancées dans ce domaine font l'objet d'une communication proposée à la conférence ECCV'08 et actuellement en cours de revue.

Parallèlement à mes travaux, la vie du laboratoire m'a également conduit à contribuer aux recherches entreprises par Bertrand Vandeportaele [VDCM06].

Perspectives de nos travaux en suivi visuel 3D

Ayant validé l'utilisation des modèles par points à toutes les étapes d'un algorithme de suivi visuel, plusieurs perspectives sont envisageables :

Suivi d'objets déformables L'une des difficultés que rencontrent les approches de suivi d'objets déformables par maillage est la gestion de la topologie et de sa cohérence, notamment lors des auto-occultations. Dans ce contexte les nuages de points peuvent être bénéfiques pour la conception d'algorithmes plus simples et plus flexibles.

Par ailleurs nous pensons que la formulation linéaire que nous proposons pour le suivi rigide peut s'étendre au suivi déformable de manière assez naturelle si les déformations sont exprimées comme une combinaison linéaire de modes de déformations. En effet, à chaque mode correspond alors un champ de mouvement 2D expliquant les mesures de déplacements observées dans les images.

Intégration d'un suivi de contours Pour les objets faiblement texturés, il faut s'appuyer sur les indices visuels offerts par exemple par les contours et la silouhette. Il serait intéressant d'étudier comment extraire d'un modèle par points une représentation continue

ou un échantillonnage du contour, par exemple en «labélisant» les points du nuage formant un contour selon la vue courante.

Bibliographie

[AA03a] A. Adamson and M. Alexa. Ray tracing point set surfaces. In *Shape Modeling International*, pages 272–279, 2003.

[AA03b] Anders Adamson and Marc Alexa. Approximating and intersecting surfaces from points. In *Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 245–254, 2003.

[ABB+01] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *Computer Graphics and Applications, IEEE*, 21(6), pages 34–47, Nov/Dec 2001.

[ABK98] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *SIGGRAPH '98 : Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421, New York, NY, USA, ACM Press, 1998.

[AKP+05] Bart Adams, Richard Keiser, Mark Pauly, Leonidas J. Guibas, Markus Gross, and Philip Dutré. Efficient raytracing of deforming point-sampled surfaces. In *Proceedings of Eurographics 2005*, Dublin, Ireland, 2005.

[AMD07] AMD, Inc. Documents sur la technologie Close-To-Metal d'ATI, 2007. http://ati.amd.com/companyinfo/researcher/documents.html.

[ART] Site de la bibliothèque ARToolkit. http://www.hitl.washington.edu/artoolkit/.

[AZ95] M. Armstrong and A. Zisserman. Robust object tracking. In *Proc. Asian Conference on Computer Vision*, volume I, pages 58–61, 1995.

[Azu97] Ronald Azuma. A survey of augmented reality. *Presence : Teleoperators and Virtual Environments 6*, 4, pages 355–385, August 1997.

[BA04] Faycal Bensaali and Abbes Amira. Design and efficient FPGA implementation of an RGB to YCrCb color space converter using distributed arithmetic. In *Field-Programmable Logic and Applications*, pages 991–995, 2004.

[BBB+97] Chandrajit Bajaj, Jim Blinn, Jules Bloomenthal, Marie-Paule Cani-Gascuel, Alyn Rockwood, Brian Wyvill, and Geoff Wyvill. *Introduction to Implicit Surfaces*. Morgan-Kaufmann, 1997.

[BG01] Samuel Boivin and André Gagalowicz. Image-based rendering of diffuse, specular and glossy surfaces from a single image. In *Proceedings of the ACM SIGGRAPH 2001*, 2001.

[BHZK05] Mario Botsch, Alexander Hornung, Matthias Zwicker, and Leif Kobbelt. High-quality surface splatting on today's gpus. In *Eurographics Symposium on Point-Based Graphics 2005*, pages 17–24, 2005.

[BJ98] Michael J. Black and Allan D. Jepson. Eigentracking : Robust matching and tracking of articulated objects using a view-based representation. *Int. J. Comput. Vision*, 26(1), pages 63–84, 1998.

[BK04] M. Botsch and L. Kobbelt. A remeshing approach to multiresolution modeling. In *Eurographics Symposium on Geometry Processing 2004*, pages 189–196, 2004.

[BKSH06] G. Bianchi, B. Knörlein, G. Székely, and M. Harders. High precision augmented reality haptics. In *Eurohaptics 2006*, 2006.

[BM04] Simon Baker and Iain Matthews. Lucas-kanade 20 years on : A unifying framework. *Int. J. Comput. Vision*, 56(3), pages 221–255, 2004.

[Bou] 2D3 Boujou. http://www.2d3.com/jsp/index.jsp.

[BSP+93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. Derose. Toolglass and magic lenses : the see-through interface. In *SIGGRAPH '93 : Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80, New York, NY, USA, ACM Press, 1993.

[BV99] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH '99 : Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co., 1999.

[CB05] Vincent Charvillat and Adrien Bartoli. Feature-based estimation of non-rigid registrations . In *10th International Fall Workshop - Vision, Modeling, and Visualization 2005*, Erlangen, Germany, Univ. Erlangen, November 2005.

[CCLW05] B. Cope, P.Y.K. Cheung, W. Luk, and S. Witt. Have GPUs made FPGAs redundant in the field of video processing ? In *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*, pages 111–118, 2005.

[CET01] T. Cootes, G. Edwards, and C. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6, pages 681–685, 2001.

[CMPC06] A.I. Comport, E. Marchand, M. Pressigout, and F. Chaumette. Real-time markerless tracking for augmented reality : the virtual visual servoing framework. *IEEE Trans. on Visualization and Computer Graphics*, 12(4), pages 615–628, July 2006.

[CNSD93] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality : the design and implementation of the cave. In *SIGGRAPH '93 : Proceedings of the 20th annual conference on*

Computer graphics and interactive techniques, pages 135–142, New York, NY, USA, ACM, 1993.

[CSA00] Marco La Cascia, Stan Sclaroff, and Vassilis Athitsos. Fast, reliable head tracking under varying illumination : An approach based on registration of texture-mapped 3d models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(4), pages 322–336, 2000.

[dBvKO00] Mark de Berg, Marc van Kreveld, and Mark Overmars. *Computational Geometry : Algorithms and Applications*. Number ISBN : 3-540-65620-0. Springer-Verlag, 2nd edition, 2000.

[DC99] Tom Drummond and Roberto Cipolla. Real-time tracking of complex structures for visual servoing. In *Workshop on Vision Algorithms*, pages 69–84, 1999.

[DC02] Tom Drummond and Roberto Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7), pages 932–946, July 2002.

[DCC07] Christophe Dehais, Vincent Charvillat, and Jean Conter. Interactive augmentation of photographs depicting prehistoric engravings. In *Euromedia 07*, Delft, Netherlands, April 2007.

[DCM06] Christophe Dehais, Vincent Charvillat, and Géraldine Morin. 3D Visual Tracking Using a Point-based Model. In K Kobbelt, editor, *Vision, Modeling and Visualization, AAchen, 22/11/2006-24/11/2006*, pages 41–48, http ://www.iospress.nl, Aka/ IOS Press, November 2006.

[DDCM04] Christophe Dehais, Matthijs Douze, Vincent Charvillat, and Géraldine Morin. Augmented Reality through real-time tracking of video sequences using a panoramic view . In *Int. Conf. on Pattern Recognition, Cambridge, 23/08/04-26/08/04*. IAPR, August 2004.

[Deb98] Paul E. Debevec. Rendering synthetic objects into real scenes : Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of SIGGRAPH 98*, July 1998.

[DGN03] E. Dubois, P. D. Gray, and L. Nigay. ASUR++ : Supporting the design of mobile mixed systems. *Interacting With Computers, Special issue on Mobile HCI*, 15, pages 497–520, 2003.

[DM96] Frédéric Dufaux and Fabrice Moscheni. Background mosaicking for low-bitrate video coding. In *proceedings of ICIP*, volume 1, pages 673–676, Lausanne, Switzerland, september 1996.

[Dou04] Matthijs Douze. *Estimation d'homographies inter-images- Cas des mosaïques et du suivi en temps réel-Applications en réalité augmentée* . Thèse de doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, décembre 2004.

[DRMS07] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. MonoSLAM : Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007. accepted for publication.

[ESN06] Chris Engels, Henrik Stewénius, and David Nistér. Bundle adjustment rules. In *SPRS Symposium on Photogrammetric Computer Vision (PCV)*, 2006.

[Far02] Gerald Farin. *Curves and Surfaces for CAGD*. Number ISBN 1-55860-737-4. Morgan-Kaufmann, 5th edition, 2002.

[Fau93] Olivier Faugeras. *Three-Dimensional Computer Vision*. MIT Press, November 1993.

[FC06] Ondrej Fialka and Martin Cadik. FFT and convolution performance in image filtering on GPU. *Tenth International Conference on Information Visualization*, pages 609–614, 2006.

[Fia04] Mark Fiala. ARTag revision 1, a fiducial marker system using digital techniques. Technical Report NRC 47419, NRC Institute for Information Technolog, November 2004.

[FL01] Olivier Faugeras and Quang-Tuan Luong. *The Geometry of Multiple Images*. MIT Press, March 2001.

[Fre08] Free Software Fundation. Site de la bibliothèque de calcul numérique GSL, 2008. http://www.gnu.org/software/gsl/.

[Gar99] Michael Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Computer Science Department, Carnegie Mellon University, 1999.

[GBP04] Gael Guennebaud, Loïc Barthe, and Mathias Paulin. Deferred splatting. *Computer Graphics Forum*, 23(3), pages 653–660, septembre 2004. (EG2004 Proceedings).

[GG07] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. In *Proceedings of Siggraph 2007*, 2007.

[GGHM05] Nico Galoppo, Naga K. Govindaraju, Michael Henson, and Dinesh Manocha. LU-GPU : Efficient algorithms for solving dense linear systems on graphics hardware. In *Proceedings of the ACM/IEEE SC|05 Conference*, 2005.

[GL91] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore and London, second edition, 1991.

[GS03] P. Gurdjos and P. Sturm. Methods and geometry for plane-based self-calibration. In *Proc. of the International Conference on Computer Vision and Pattern Recognition*, pages 491–496, 2003.

[Gue05] Gaël Guennebaud. *Algorithmes pour le rendu temps-réel de haute qualité des géométries basées points*. PhD thesis, Université Paul Sabatier Toulouse 3, 2005.

[Har92] C. Harris. *Tracking with rigid objects*. MIT Press, 1992.

[HB98] G.D. Hager and P.N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(10), pages 1025–1039, 1998.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH '92 : Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 71–78, New York, NY, USA, ACM Press, 1992.

[HLN96] W. Hoff, T. Lyon, and K. Nguyen. Computer vision-based registration techniques for augmented reality. In *Proc. of Intelligent Robots and Computer Vision*, 1996.

[HS88] C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2003.

[IAB+96] Michal Irani, P. Anandan, J. R. Bergen, Rakesh Kumar, and Steve Hsu. Efficient representation of video sequences and their applications. *Signal Processing : Image Communication*, 8, pages 327–351, may 1996.

[IB98] Michael Isard and Andrew Blake. CONDENSATION – conditional density propagation for visual tracking. *Int. J. Computer Vision*, 1(29), pages 5–28, 1998.

[Inc07] Sony Computer Entertainment Inc. Site du jeu The Eye of Judgment pour console Playstation 3 (tm), 2007. http://www.eyeofjudgment.com/.

[Int08] Intel Corp. Site de la bibliothèque OpenCV, 2008. http://www.intel.com/technology/computing/opencv/.

[JD02] F. Jurie and M. Dhome. Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7), pages 996–1000, 2002.

[JZLA04] Yunde Jia, Xiaoxun Zhang, Mingxiang Li, and Luping An. A miniature stereo vision machine (msvm-iii) for dense disparity mapping. In *ICPR '04 : Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1*, pages 728–731, Washington, DC, USA, IEEE Computer Society, 2004.

[KB04] L. Kobbelt and M. Botsch. A survey of pointbased techniques in computer graphics. *Proceedings of ACM SIGGRAPH*, 2004.

[KD03] Georg Klein and Tom Drummond. Robust visual tracking for non-instrumented augmented reality. In *Proc. Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pages 113–122, Tokyo, October 2003.

[KD04] Georg Klein and Tom Drummond. Sensor fusion and occlusion refinement for tablet-based AR. In *Proc. Third IEEE and ACM International Symposium*

on Mixed and Augmented Reality (ISMAR'04), pages 38–47, Arlington, VA, November 2004.

[Khr07a] Khronos Group. OpenGL developer web site, 2007. www.opengl.org.

[Khr07b] Khronos Group. Section sur le langage GLSL du site officiel d'OpenGL, 2007. http://opengl.org/documentation/glsl/.

[KM06] Georg Klein and David Murray. Full-3d edge tracking with a particle filter. In *Proc. British Machine Vision Conference (BMVC'06)*, Edinburgh, BMVA, September 2006.

[KM07] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes : A high resolution 3d surface construction algorithm. In *SIGGRAPH '87 : Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, ACM Press, 1987.

[LDR00] Céline Loscos, George Drettakis, and Luc Robert. Interactive virtual relighting of real scenes. *IEEE Transactions on Visualization and Computer Graphics*, 6(3), July-September 2000.

[Lev03] David Levin. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization*, pages 37–49, 2003.

[LF06] Vincent Lepetit and Pascal Fua. Keypoint recognition using randomized trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(9), pages 1465–1479, 2006.

[LHSD05] Johnny C. Lee, Scott E. Hudson, Jay W. Summet, and Paul H. Dietz. Moveable interactive projected displays using projector based tracking. In *UIST '05 : Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 63–72, New York, NY, USA, ACM, 2005.

[LK81] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 674–679, April 1981.

[LLF05] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *Conference on Computer Vision and Pattern Recognition*, San Diego, CA, June 2005.

[LMLN01] G. Lienhart, R. Männer, R. Lay, and K.H. Noffz. An FPGA-Based video compressor for H.263 compatible bit streams. In *International Symposium on Field Programmable Gate Arrays 01*, 2001.

[Low85] David G. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, Norwell, MA, USA, 1985.

[Low92] David G. Lowe. Robust model-based motion tracking through the integration of search and estimation. *Int. J. Comput. Vision*, 8(2), pages 113–122, 1992.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60, pages 91–110, 2004.

[LPC+00] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project : 3d scanning of large statues. In *Proceedings of Siggraph 2000*, pages 131–144, July 2000.

[LS81] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of computation*, 37(155), July 1981.

[LVD98] Jean-Marc Lavest, Marc Viala, and Michel Dhome. Do we really need an accurate calibration pattern to achieve a reliable camera calibration ? In *ECCV '98 : Proceedings of the 5th European Conference on Computer Vision-Volume I*, pages 158–174, London, UK, Springer-Verlag, 1998.

[LW85] M. Levoy and T. Whitted. The use of points as a display primitive. Technical Report TR85-022, Dept. of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1985.

[MDJ04] Lucie Masson, Michel Dhome, and Jurie Jurie. Robust real time tracking of 3D objects. In *17th International Conference on Pattern Recognition*, 2004.

[MK94] Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE Transactions on Information Systems*, E77-D(12), December 1994.

[MLD+07] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Generic and real time structure from motion. In *British Machine Vision Conference (BMVC)*, 2007.

[Mor80] Hans Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. In *tech. report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University & doctoral dissertation, Stanford University.* September 1980. Available as Stanford AIM-340, CS-80-813 and republished as a Carnegie Mellon University Robotics Institue Technical Report to increase availability.

[MS01] Krystian Mikolajczyk and Cordelia Schmid. Indexing based on scale invariant interest points. In *Proceedings of the 8th International Conference on Computer Vision, Vancouver, Canada*, pages 525–531, 2001.

[MS02] Krystian Mikolajczyk and Cordelia Schmid. An affine invariant interest point detector. In *Proceedings of the 7th European Conference on Computer Vision, Copenhagen, Denmark*, pages 128–142. Springer, 2002. Copenhagen.

[NNB04] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004*, 1, pages 652–659, 2004.

[nVi07a] nVidia Corporation. Section sur la technologie CUDA du site de la firme nVidia, 2007. http://developer.nvidia.com/object/cuda.html.

[nVi07b] nVidia Corporation. Section sur le langage Cg du site de la firme nVidia, 2007. http://developer.nvidia.com/page/cg_main.html.

[Pau03] Mark Pauly. *Point Primitives for Interactive Modeling and Processing of 3D Geometry*. PhD thesis, Federal Institute of Technology (ETH) of Zurich, 2003.

[PE02] Fernando Pereira and Touradj Ebrahimi, editors. *The MPEG-4 Book*. Pearson Education, Upper Saddle River, NJ, 2002.

[PGK02] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *VIS '02 : Proceedings of the conference on Visualization '02*, pages 163–170, Washington, DC, USA, IEEE Computer Society, 2002.

[PHMG06] Juri Platonov, Hauke Heibel, Peter Meier, and Bert Grollmann. A mobil markerless ar system for maintenance and repair. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'06)*, Santa Barbara (CA), USA, October 2006.

[Pix] PixelFarm PFTrack. http://www.thepixelfarm.co.uk/.

[PKA+05] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutré, Markus Gross, and Leonidas J. Guibas. Meshless animation of fracturing solids. In *In proceedings of ACM SIGGRAPH*, Los Angeles, CA, 2005.

[PM06] M. Pressigout and E. Marchand. Real-time 3d model-based tracking : combining edge and texture information. In *IEEE International Conference on Robotics and Automation, ICRA 2006*, pages 2726–2731, 2006.

[RC02] I. Rudomin and J. Castillo. Realtime clothing : geometry and physics, 2002.

[RD05] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005.

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[Rea] RealViz Match Mover. http://www.realviz.com/.

[Ree83] W. T. Reeves. Particle systems — a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2), pages 91–108, 1983.

[RHHL02] Szymon Rusinkiewicz, Olaf Hall-Holt, and Marc Levoy. Real-time 3d model acquisition. In *SIGGRAPH '02 : Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 438–446, New York, NY, USA, ACM Press, 2002.

[RL87] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection.* John Wiley & sons, 1987.

[RL00] Szymon Rusinkiewicz and Marc Levoy. QSplat : A multiresolution point rendering system for large meshes. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[RLDL07] Eric Royer, Maxime Lhuillier, Michel Dhome, and Jean-Marc Lavest. Monocular vision for mobile robot localization and autonomous navigation. *International Journal of Computer Vision*, 74(3), pages 237–260, September 2007.

[RTSD03] Patrick Reuter, Ireneusz Tobor, Christophe Schlick, and Sébastien Dedieu. Point-based modelling and rendering using radial basis functions. In *GRAPHITE '03 : Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 111–118, New York, NY, USA, ACM, 2003.

[SB97] Stephen M. Smith and J. Michael Brady. SUSAN-a new approach to low level image processing. *Int. J. Comput. Vision*, 23(1), pages 45–78, 1997.

[SDF05] A. Shahrokni, T. Drummond, and P. Fua. Fast texture-based tracking and delineation using texture entropy. In *International Conference in Computer Vision*, October 2005.

[SFPG06] Sudipta N. Sinha, Jan-Michael Frahm, Marc Pollefeys, and Yakup Genc. GPU-based video feature tracking and matching. Technical Report 06-012, Department of Computer Science, UNC Chapel Hill, May 2006.

[SLZ01] Ying Shan, Zicheng Liu, and Zhengyou Zhang. Model-based bundle adjustment with application to face modeling. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 644–651, 2001.

[SMB00] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2), pages 151–172, 2000.

[SNT+98] Y. Sato, M. Nakamoto, Y. Tamaki, T. Sasama, I. Sakita, Y. Nakajima, M. Monden, and S. Tamura. Image guidance of breast cancer surgery using 3-d ultrasound images and augmented reality visualization. *Medical Imaging, IEEE Transactions on*, 17(5), pages 681–693, 1998.

[SOB+04] Tobias Sielhorst, Tobias Obst, Rainer Burgkart, Robert Riener, and Nassir Navab. An augmented reality delivery simulator for medical training. In *Workshop AMI-ARCS'04*, 2004.

[SPF+07] John E. Stone, James C. Phillips, Peter L. Freddolino, David J. Hardy, Leonardo G. Trabuco, and Klaus Schulten. Accelerating molecular modeling applications with graphics processors. *Journal of Computational Chemistry*, 2007.

[ST94] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, June 1994.

[TCD+00] B. Thomas, B. Close, J. Donoghue, J. Squires, P. De Bondi, M. Morris, and W. Piekarski. ARQuake : an outdoor/indoor augmented reality first person application. In *The Fourth International Symposium on Wearable Computers, 2000*, pages 139–146, 2000.

[TMHF00] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment – a modern synthesis. In B. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms : Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer-Verlag, 2000.

[TO06] Leonardo Trujillo and Gustavo Olague. Using evolution to learn how to perform interest point detection. In *ICPR '06 : Proceedings of the 18th International Conference on Pattern Recognition*, pages 211–214, Washington, DC, USA, IEEE Computer Society, 2006.

[Tom97] C. Tomlin. Control of systems on lie groups. Number UCB/ERL M97/50, 1997.

[Var74] V. Varadarajan. Lie groups, lie algebras and their representations. *Graduate Texts in Mathematics*, 102, 1974.

[VDCM06] Bertrand Vandeportaele, Christophe Dehais, Michel Cattoën, and Philippe Marthon. Orient-Cam, a camera that knows its orientation and some applications. In *Ibero-American Congress on Pattern Recognition, Cancun, Mexico, 14/11/06-17/11/06*, 2006.

[VLF04a] L. Vacchetti, V. Lepetit, and P. Fua. Stable real-time 3D tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10), pages 1391–1391, 2004.

[VLF04b] Luca Vacchetti, Vincent Lepetit, and Pascal Fua. Combining edge and texture information for real-time accurate 3D camera tracking. In *ISMAR '04 : Proceedings of the Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pages 48–57, Washington, DC, USA, IEEE Computer Society, 2004.

[Wel78] Robert B. Welch. *Perceptual Modification : Adapting to Altered Sensory Environments*. Academic Press (1978), 1978.

[WOSL01] S. Wiedenmaier, O. Oehme, L. Schmidt, and H. Luczak. Augmented reality (AR) for assembly processes - an experimental evaluation. pages 185–186, 2001.

[WS05] I. Wald and H.-P. Seidel. Interactive ray tracing of point-based models. In *Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings*, pages 9–16, 2005.

[Wu06] Changchang Wu. Webpage on SIFT implementation on GPU, 2006. http://cs.unc.edu/~ccwu/siftgpu/.

[WWWC04] Rafal Wojciechowski, Krzysztof Walczak, Martin White, and Wojciech Cellary. Building virtual and augmented reality museum exhibitions. In *Web3D '04 : Proceedings of the ninth international conference on 3D Web technology*, pages 135–144, New York, NY, USA, ACM, 2004.

[Zha00] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), pages 1330–1334, November 2000.

[ZPBG02] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8, pages 223–238, July-September 2002.

[ZPKG02] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3d : An interactive system for point-based surface editing. In *Proceedings of ACM Siggraph 2002*, 2002.

[ZPvBG01] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 371–378. ACM Press / ACM SIGGRAPH, 2001.

[ZRB+04] Matthias Zwicker, Jussi Räsänen, Mario Botsch, Carsten Dachsbacher, and Mark Pauly. Perspective accurate splatting. In *GI '04 : Proceedings of Graphics Interface 2004*, pages 247–254, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Canadian Human-Computer Communications Society, 2004.

[ZTTS06] Gernot Ziegler, Art Tevs, Christian Theobalt, and Hans-Peter Seidel. On-the-fly point clouds through histogram pyramids. In *Proceedings of Vision Modeling and Visualization 2006 (VMV06)*, 2006.