Méthodes de rendu à base de vidéos et applications à la réalité Virtuelle

Vincent Nozick

Université de Marne-la-Vallée

THÈSE

pour obtenir le grade de

Docteur de l'Université de Marne-La-Vallée

Spécialité : Informatique

présentée et soutenue publiquement par

M. Vincent Nozick

le mercredi 7 juin 2006

Méthodes de Rendu à Base de Vidéos et Applications à la Réalité Virtuelle

Video-Based Rendering and Virtual Reality

Directeur de thèse
Didier Arquès

Jury		
Rapporteurs :	Pascal Estraillier,	Professeur à l'Université de la Rochelle
	Bernard Péroche,	Professeur à l'Université de Lyon I
Examinateurs :	Didier Arquès,	Professeur à l'Université de Marne-la-Vallée
	Daniel Laurent	Professeur à l'Université de Marne-la-Vallée
	Laurent Lucas,	Professeur à l'Université de Reims
	Henri Maître,	Professeur à l'ENST
Invité	Benoît Piranda,	Maître de conférence à l'Université de Marne-la-Vallée

à mes parents,

Remerciements

J’adresse toute ma gratitude à Pascal Estraillier, Laurent Lucas et Henri Maître et Bernard Péroche d’avoir accepté de faire parti de mon jury. J’exprime aussi ma sincère reconnaissance à Daniel Laurent pour l’honneur qu’il me fait d’être membre de mon jury.

J’adresse un profond remerciement à Didier Arquès, mon directeur de thèse, pour m’avoir donné la chance d’effectuer une thèse au sein de l’équipe SISAR et pour la confiance qu’il m’a accordée.

Je tiens à remercier Sylvain Michelin pour sa patience et sa bonne humeur. Ses conseils et son expérience m’ont largement aider à ne pas perdre de vue mes objectifs. Je souhaite également remercier Benoît Piranda pour l’attention qu’il a su porter à mon travail. Ses conseils de préparation et ses recommandations de dernière minute en situation de stress m’ont été d’un grand secours.

Je ne saurais assez remercier Venceslas Biri qui m’a apporté dès le début de ma thèse ses encouragements et son soutien.

Je souhaite adresser un grand merci à tous les membres de l’équipe SISAR qui, par leur attitude chaleureuse et leur sympathie quotidienne ont rendu ces années de thèse très agréables. Je tiens à remercier particulièrement Cyril Pichard, Olivier Derpierre, Pascal Chaudeyrac, Pierre Boulenguez, Patrice Bouvier, François de Sorbier et Ismaël Grimeau pour leur aide, leur bonne humeur et leur amitié.

Un grand merci à Christophe Leroy pour son amitié et son aide durant toute la fin de mon cursus universitaire.

J’adresse enfin des remerciements particuliers à mes parents pour tout ce qu’ils ont fait et font pour moi.

Résumé

Mots-clés : rendu à base de vidéos, GPU, réalité virtuelle.

Etant donné un ensemble de caméras filmant une même scène, le rendu à base de vidéos consiste à générer de nouvelles images de cette scène à partir de nouveaux points de vue. L'utilisateur a ainsi l'impression de pouvoir déplacer une caméra virtuelle dans la scène alors qu'en réalité, toutes les caméras sont fixes.

Certaines méthodes de rendu à base de vidéos coûteuses en temps de calcul se basent sur une reconstruction 3d de la scène et produisent des images de très bonne qualité. D'autres méthodes s'orientent plutôt vers le rendu temps réel. C'est dans cette dernière catégorie que s'inscrit la méthode de *Plane Sweep* sur laquelle porte la majeure partie de nos travaux.

Le principe de la méthode des *Plane Sweep* consiste à discrétiser la scène en plans parallèles et à traiter séparément chaque point de ces plans afin de déterminer s'ils se trouvent ou non sur la surface d'un objet de la scène. Les résultats obtenus permettent de générer une nouvelle image de la scène à partir d'un nouveau point de vue. Cette méthode est particulièrement bien adaptée à une utilisation optimale des ressources de la carte graphique ce qui explique qu'elle permette d'effectuer du rendu en temps réel.

Notre principale contribution à cette méthode concerne la façon d'estimer si un point d'un plan représente la surface d'un objet. Nous proposons d'une part un nouveau mode de calcul permettant d'améliorer le résultat visuel tout en rendant la navigation de la caméra virtuelle plus souple. D'autre part, nous présentons une adaptation de la méthode des *Plane Sweep* permettant de gérer les occlusions partielles.

Compte tenu des applications du rendu à base de vidéos en réalité virtuelle, nous proposons une amélioration des *Plane Sweep* appliquée à la réalité virtuelle avec notamment la création de paires d'images stéréoscopiques permettant de visualiser en relief la scène reconstruite. Notre amélioration consiste à calculer la seconde vue à moindre coût alors qu'une majorité des méthodes concurrentes sont contraintes d'effectuer deux rendus indépendants. Cette amélioration est basée sur un partage des données communes aux deux vues stéréoscopiques.

Enfin, dans le cadre de l'utilisation des *Plane Sweep* en réalité virtuelle, nous présentons une méthode permettant de supprimer les mouvements pseudoscopiques. Ces mouvements pseudoscopiques apparaissent lorsque l'observateur se déplace devant une image stéréoscopique, il ressent alors une distorsion des proportions de la scène virtuelle et voit les objets se déplacer de façon anormale. La méthode de correction que nous proposons est applicable d'une part à des méthodes classiques de rendu d'images de synthèse et d'autre part à la méthode des *Plane Sweep*.

Toutes les méthodes que nous présentons utilisent largement les possibilités du processeur de la carte graphique à l'aide des *shader programs* et génèrent toutes des images en temps réel. Seuls un ordinateur grand public, un dispositif d'acquisition vidéo et une bonne carte graphique sont suffisants pour les faire fonctionner. Les applications des *Plane Sweep* sont nombreuses, en particulier dans les domaines de la réalité virtuelle, du jeu vidéo, de la télévision 3d ou de la sécurité.

Abstract

Key words : Video-Based Rendering, Shader Programs, Virtual Reality.

Given a set images of the same scene, the goal of video-based rendering methods is to compute new views of this scene from new viewpoints. The user of this system controls the virtual camera's movement through the scene. Nevertheless, the virtual images are computed from static cameras.

A first approach is based on a reconstruction of the scene and can provide accurate models but often requires lengthy computation before visualization. Other methods try to achieve real-time rendering. Our main contribution to video-base rendering concerns the plane sweep method which belongs to the latter family.

The plane sweep method divides space in parallel planes. Each point of each plane is processed independently in order to know if it lies on the surface of an object of the scene. These informations are used to compute a new view of the scene from a new viewpoint. This method is well suited to an implementation using graphic hardware and thus to reach real-time rendering.

Our main contribution to this method concerns the way to consider whether a point of a plane lies on the surface of an object of the scene. We first propose a new scoring method increasing the visual quality of the new images. Compared with previous approaches, this method implies fewer constraints on the position of the virtaul camera, i.e. this camera does not need to lie between the input camera's area. We also present an adaptation of the plane sweep algorithm that handles partial occlusions.

According to video-based rendering practical applications in virtual reality, we propose an improvement of the plane sweep method dealing with stereoscopic images computation that provides visualization of the virtual scene in relief. Our enhancement provides the second view with only low additional computation time whereas most of the others techniques require to render the scene twice. This improvement is based on a sharing of the informations common to the two stereoscopic views.

Finally, we propose a method that removes pseudoscopic movements in a virtual reality application. These pseudoscopic movements appear when the observer moves in front of the stereoscopic screen. Then the scene proportions seem to be distorted and the observer sees the objects of the scene moving in an anormal way. The method we propose is available either on a classical stereoscopic rendering method or on the Plane Seep algorithm.

Every method we propose widely uses graphic harware through to shader programs and provides real-time rendering. These methods only require a standard computer, a video acquisition device and a powerful enough graphic card. There exists a lot of practical

applications of the plane sweep method, especially in fields like virtual reality, video games, 3d television or security.

Table des matières

Université de Marne-la-Vallée 1
Remerciements 5
Résumé 7
Abstract 9
Table des matières 11
Liste des figures 15
Liste des annexes 21
Introduction 23
Première partie Rendu à base d'images 27
1. Vision par Ordinateur, Dispositifs de Capture 28
1.1 Géométrie projective 28
1.1.1 Deux dimensions : $\mathbb{P}^2$ 28
1.1.2 Trois dimensions : $\mathbb{P}^3$ 29
1.1.3 Homographie dans $\mathbb{P}^2$ 29
1.2 Camera projective 31
1.2.1 Matrice de projection 31
1.2.2 Extraction des paramètres d'une matrice de projection 33
1.2.3 Calibrage de la caméra 33
1.2.4 Le « Gold Standard Algorithm » 36
1.2.5 Autres méthodes de calibrage 37
1.2.6 Correction de la distorsion radiale 37
1.3 Géométrie épipolaire 40
1.3.1 Matrice fondamentale 40
1.3.2 Droites épipolaires, épipôles 41
1.4 Localisation d'un point dans l'espace 42
1.4.1 Localisation d'un point dans l'espace par triangularisation 42
1.4.2 Projeté d'un point sur un plan 43
1.5 Mise en place d'un dispositif d'acquisition vidéo 44
1.5.1 Types de caméras 44

1.5.2 Disposition des caméras 44
1.5.3 Illumination 45
1.5.4 Calibrage de la couleur et balance des blancs 46
1.5.5 Gestion d'un réseau, compression du flux vidéo 47
1.6 Bilan 47
2 Rendu à base d'images : état de l'art 49
2.1 Rendu à base d'image réaliste non temps réel 50
2.1.1 Premier pas vers le rendu à base d'images : les cartes de disparités 51
2.1.2 Voxels et Poxels 56
2.1.3 Image-based Priors 61
2.1.4 Morphing 62
2.1.5 Projective Grid Space – caméras non calibrées 64
2.1.6 Bilan 65
2.2 Rendu à base d'images temps-réel 65
2.2.1 3d à partir d'une seule image 66
2.2.2 Imposteurs 69
2.2.3 QuickTimeVR 72
2.2.4 Plenoptic Modeling 74
2.2.5 Light Field Rendering 77
2.2.6 Lumigraph 79
2.2.7 Méthodes de reconstruction 81
2.2.8 Bilan 84
2.3 Rendu à base de vidéos : play-back interactif 85
2.3.1 Virtualized Reality™ 85
2.3.2 Layered Representation 87
2.3.3 Murs de caméras 89
2.3.4 Light Fields et Lumigraph 92
2.3.5 Bilan 93
2.4 Rendu à base de vidéos en direct 94
2.4.1 « 3D TV » 94
2.4.2 Light Fields dynamiques, Lumigraphs dynamiques 95
2.4.3 Visual Hulls 96
2.4.4 Plane Sweep 101
2.4.5 Bilan 102

3 « Plane Sweep » : Notre contribution 103
3.1 Principe général 103
3.1.1 Première approche 103
3.1.2 Principe de fonctionnement 106
3.2 Etat de l'art 107
3.2.1 Méthode de Collins 107
3.2.2 Méthode de Yang 108
3.2.3 Méthode de Wotzel 109
3.2.4 Méthode de Geys 110
3.2.5 Méthode de Photo Hulls 112
3.2.6 Bilan 112
3.3 Nouvelle approche des Plane Sweep 113
3.3.1 Calcul de scores 113
3.3.2 Mipmapping 116
3.3.3 Implémentation 116
3.3.4 Mode opératoire 119
3.3.5 Résultats 121
3.4 Gestion des occlusions 125
3.4.1 Problématique 125
3.4.2 Sigma-Clipping 127
3.4.3 Algorithme dynamique 129
3.4.4 Résultats 129
3.5 Plane Sweep : points forts et faiblesses 132
3.5.1 Contraintes sur la scène 132
3.5.2 Manipulation et usage 133
3.5.3 Résultat visuel 134
3.5.4 Matériel 134
Seconde partie Rendu stéréoscopique à base de Vidéos et Réalité Virtuelle 135
4 Plane Sweep et Rendu Stéréoscopique 137
4.1 Perception du relief 137
4.1.1 Vision monoculaire 137
4.1.2 Vision stéréoscopique 137
4.2 Rendu stéréoscopique en synthèse d'images 141
4.3 Plane Sweep et rendu stéréoscopique 144

4.3.1 Modification de l'algorithme classique 144
4.3.2 Implémentation 146
4.3.3 Résultats 146
5 Réalité Virtuelle et Mouvements Pseudoscopiques 149
5.1 Mouvements pseudoscopiques en réalité virtuelle 149
5.1.1 Mouvements pseudoscopiques 149
5.1.2 Position orthostéréoscopique 150
5.2 Correction des mouvements pseudoscopiques : méthodes existantes 152
5.3 Correction des mouvements pseudoscopiques en synthèse d'images 153
5.3.1 Création d'une paire d'images en position orthostéréoscopique 153
5.3.2 Analogies entre la scène réelle et la scène virtuelle 154
5.3.3 Correction des mouvements pseudoscopiques en synthèse d'image 157
5.3.4 Implémentation 158
5.3.5 Résultats 159
5.4 Adaptation aux Plane Sweep 161
Conclusion 163
Bibliographie 167

Liste des figures

Figure 1 : homographie. 29
Figure 2 : repère de la caméra et repère de la scène. 32
Figure 3 : mire de calibrage. 34
Figure 4 : distorsion tangentielle et radiale. 37
Figure 5 : correction de la distorsion radiale. 38
Figure 6 : exemple de correction de distorsion radiale. 39
Figure 7 : géométrie épipolaire. 40
Figure 8 : triangularistion. 42
Figure 9 : *flare*. 45
Figure 10 : *lens flare*. 45
Figure 11 : *Color Checker* de Macbeth. 46
Figure 12 : segmentation de couleurs et application au calcul de cartes de disparité. 55
Figure 13 : flux optique. 55
Figure 14 : voxels : appariement de deux couples de pixels de même couleur. 56
Figure 15 : voxels : appariement de quatre couples de pixels sans occlusion. 57
Figure 16 : voxels : appariement de quatre couples de pixels avec occlusions entre pixels de couleur différente. 57
Figure 17 : voxels : appariement de quatre couples de pixels avec occlusions entre pixels de même couleur. 58
Figure 18 : répartition des voxels par couches. 58
Figure 19 : répartition des voxels par couches, application aux vue panoramiques. 59
Figure 20 : voxels : image originale. 60
Figure 21 : voxels : reconstruction avec plusieurs niveaux de discrétisation. 60
Figure 22 : poxels, reconstruction gérant les éléments semi-transparents. 61
Figure 23 : Image-Based Priors. 62
Figure 24 : morphing. 63
Figure 25 : warping par maillage. 63
Figure 26 : Projective Grid Space, principe de base. 64
Figure 27 : Tour into the Picture, extraction de premier plan et d'arrière plan. 66

Figure 28 : Tour into the Picture, dettection des lignes de fuites. 67
Figure 29 : Tour into the Picture, modélistaion par plans. 67
Figure 30 : Tour into the Picture,création d'une nouvelle vue en déplaçant les points de fuite. 67
Figure 31 : *billboards sreen-aligned*. 69
Figure 32 : *billboards world oriented.* 70
Figure 33 : *billboards* combinés 70
Figure 34 : classement des méthodes d'imposteurs. 72
Figure 35 : principe de création d'une image panoramique. 73
Figure 36 : création d'une image panoramique : images de départ. 73
Figure 37 : création d'une image panoramique, image générée. 73
Figure 38 : vue de dessus d'un parcours parmi les nœuds d'un réseau d'images panoramiques. 74
Figure 39 : fonction Plenoptic en coordonnées sphériques. 75
Figure 40 : relation épipolaire entre deux images panoramiques cylindriques. 75
Figure 41 : droites épipolaires sur unr image panoramique cylindrique. 76
Figure 42 : Light Field, une fonction à quatre paramètres. 77
Figure 43 : rendu par Light Field Rendering 78
Figure 44 : quatre Light Fields disposés en carré autour de la scène 79
Figure 45 : Lumigraph, interpolation de deux rayons. 80
Figure 46 : images de départ. 81
Figure 47 : calcul de points de correspondance. 81
Figure 48 : relation épipolaires entre images. 82
Figure 49 : reconstruction grossière. 82
Figure 50 : autocalibrage et reconstruction métrique. 83
Figure 51 : carte de profondeur dense. 83
Figure 52 : maillage et placage de textures. 84
Figure 53 : précision de la triangularisation en fonction de la distance entre les caméras. 86
Figure 54 : Virtualized Reality, gestion des discontinuities spatiales. 87
Figure 55 : Layered Representation, gestion des discontinuités spatiales. 88
Figure 56 : Layered Representation, hiérarchie des traitements. 88
Figure 57 : Layered Representation, segmentation des couleurs. 89
Figure 58 : Layered Representation, carte de disparité générée. 89
Figure 59 : Layered Representation, image générée. 89

Figure 60 : mur de caméras (Stanford Camera Array).....90
Figure 61 : image priseavec une ouverture du diaphragme très faible.....91
Figure 62 : image priseavec une ouverture du diaphragme très grande.....91
Figure 63 : simulation d'ouverture du diaphragme.....92
Figure 64 : Distributed Light Field, système de client/serveur.....96
Figure 65 : Visual Hulls, principe général.....97
Figure 66 : Image Based Visual Hulls, creation d'une nouvelle image.....99
Figure 67 : Image Based Visual Hulls, problème de visibilité.....100
Figure 68 : Image Based Visual Hulls, decoupage de l'image en plusieurs zones.....100
Figure 69 : Visual Hulls et textures projetées.....101
Figure 70 : Image-Based Priors.....104
Figure 71 : Plane Sweep, première approche.....105
Figure 72 : Plane Sweep, le point *M* appartient à la surface d'un objet.....105
Figure 73 : Plane Sweep, le point *M* n'appartient pas à la surface d'un objet.....105
Figure 74 : Plane sweep, détail d'une itération.....106
Figure 75 : Plane Sweep, projetés des images de departs sur quatres plans situés à des profondeurs différentes.....107
Figure 76 : méthode de Collins sur les images en niveaux de gris.....108
Figure 77 : méthodes de Yang.....109
Figure 78 : méthode de Woetzel, une des images de départ.....110
Figure 79 : méthode de Woetzel, carte de profondeur générée.....110
Figure 80 : méthode Geys.....111
Figure 81 : méthode des Photo Hulls.....112
Figure 82 : Plane Sweep, calcul de score.....114
Figure 83 : les points caractéristiques de la mire sont localisés à l'aide d'un bras articulé. 119
Figure 84 : choix du plan *min* et du plan *max*.....120
Figure 85 : Plane Sweep, dispositif d'acquisition.....120
Figure 86 : quatres images de départ.....121
Figure 87 : images générées.....121
Figure 88 : Plane Sweep et mipmapping.....122
Figure 89 : Plane Sweep, influence du nombre de plans sur la qualité visuelle des images générées.....123
Figure 90 : création d'une carte de profondeur.....123
Figure 91 : occlusion totale.....126

Figure 92 : occlusion partielle. 126
Figure 93 : auto-occlusion classique. 127
Figure 94 : auto-occlusion sur le bord d'un objet. 127
Figure 95 : occlusion partielle et Sigma Clipping. 128
Figure 96 : comparatif des méthodes standard, de Sigma Clipping et dynamique. 130
Figure 97 : zones où le sigma clipping prend effet. 131
Figure 98 : zones où la méthode dynamique prend effet. 131
Figure 99 : arrière plan monochrome, source d'erreurs. 133
Figure 100 : parallaxe. 138
Figure 101 : angle de parallaxe. 138
Figure 102 : parallaxe positive. 139
Figure 103 : parallaxe négative. 139
Figure 104 : parallaxe nulle. 140
Figure 105 : axes optiques convergents. 140
Figure 106 : axes optiques parallèles. 140
Figure 107 : apparition de parallaxe verticale. 141
Figure 108 : système de deux caméras stéréoscopiques. 142
Figure 109 : caméras virtuelles proches. 143
Figure 110 : caméras virtuelle éloignées. 143
Figure 111 : pyramide de projection (glFrustum). 144
Figure 112 : Plane Sweep et stéréoscopie. 145
Figure 113 : images stéréoscopiques générées. 146
Figure 114 : séquence d'images stéréoscopiques. 147
Figure 115 : mouvements pseudoscopiques vers l'avant. 150
Figure 116 : mouvements pseudoscopiques latéraux. 150
Figure 117 : distorsion de la profondeur. 151
Figure 118 : distorsion de la largeur (et de la hateur). 151
Figure 119 : paramètres d'un systèmes de deux caméras stéréoscopiques. 153
Figure 120 : paramètres d'un systèmes de deux caméras stéréoscopiques excentré par rapport à l'écran. 155
Figure 121 : système de deux caméras stéréoscopiques vu de coté. 156
Figure 122 : scène liée à l'écran. 157
Figure 123 : scène liée à l'observateur. 157
Figure 124 : observateur devant l'écran. 160

Figure 125 : scène liée à l'observateur. 160
Figure 126 : scène liée à l'écran. 160
Figure 127 : scène liée à l'écran vue par l'observateur. 160
Figure 128 : correction des mouvements pseudoscopiques et méthode des Plane Sweep... 161
Figure 129 : attributs des vertex shaders. 196
Figure 130 : attributs des frgments shaders. 196
Figure 131 : vertex shaders et fragments shaders dans le pipeline OpenGL. 197

Liste des annexes

Annexes 183

Décomposition en valeurs singulières 185

Historique des techniques de rendu d'images en relief 188

Fragment shaders et vertex shaders dans le pipeline openGL 194

Introduction

Les méthodes de rendu d'images de synthèse ne cessent de s'améliorer, tant en ce qui concerne les images photo-réalistes que les images générées en temps réel. On parvient ainsi à créer des films en images de synthèse dont le réalisme est parfois saisissant. Dans un même temps, les techniques de rendu d'images de synthèse en temps réel atteignent elles aussi un niveau de rendu impressionnant compte tenu des contraintes temps réel qui leur sont imposées. Ce constat est particulièrement frappant concernant les jeux vidéo dont les créateurs ne cessent de chercher de nouvelles méthodes pour les rendre plus fluides et augmenter la sensation d'immersion.

C'est dans cet état d'esprit qu'est apparue l'idée d'intégrer des photos dans des images de synthèse pour les rendre plus réalistes. Ce principe consiste à plaquer une texture sur un modèle géométrique prédéfini. Pour aller plus loin dans cette direction, certains chercheurs ont proposé d'extraire de l'information contenue dans les photos pour reconstruire ce modèle géométrique. Les photographies de la scène sont donc utilisées d'une part pour la création du modèle géométrique et d'autre part pour la gestion des couleurs lors du rendu d'une nouvelle image.

Une autre approche du problème a permis à certains chercheurs de proposer des méthodes pouvant se passer d'un modèle géométrique. Ces méthodes tentent d'estimer le champs de lumière aux alentours d'une scène à partir d'un grand nombre de caméras (éventuellement plusieurs centaines). Les nouvelles images sont alors calculées directement à partir des photographies de la scène. Ces méthodes se passent donc de modèle géométrique et donc de reconstruction de la scène.

Il existe actuellement une multitude de méthodes permettant de générer de nouvelles vues d'une scène. Ces méthodes utilisent au mieux les avantages respectifs de ces deux approches et génèrent des images de façon plus ou moins rapide et plus ou moins précise. En effet, alors que certaines méthodes nécessitent un grand nombre de calculs, d'autres méthodes plus rapides permettent d'effectuer du rendu en temps réel.

Grâce à l'évolution des technologies et des algorithmes de vision par ordinateur, certaines méthodes sont devenues suffisamment rapides pour pouvoir traiter des vidéos plutôt que des photos, donnant ainsi accès à la restitution de scènes dynamiques. Ces méthodes de rendu à base de vidéos se classent en deux catégories. La première nécessite un prétraitement sur les vidéos avant de permettre une navigation en temps réel dans la scène recréée. Il s'agit alors d'une visualisation en différé. La seconde catégorie de méthodes traite l'information suffisamment rapidement pour permettre une navigation en direct (*on-line*). C'est dans cette dernière catégorie que s'inscrit la méthode des « *Plane Sweep* » sur laquelle porte la majeure partie de mes travaux.

Je propose en effet une nouvelle approche de cette méthode, notamment au niveau du traitement des données, permettant d'augmenter la qualité visuelle du rendu et permettant une navigation plus fluide dans la scène virtuelle. Je propose en outre une seconde modification ayant attrait aux occlusions partielles. Celles-ci apparaissent sur l'image générée notamment au niveau des contours des objets. Il s'agit d'un problème commun à une grande partie des méthodes de reconstruction auxquelles la méthode des *Plane Sweep* ne fait pas exception. Nous proposons une adaptation des *Plane Sweep* permettant de gérer ces occlusions partielles tout en conservant le rendu en temps réel. Cette amélioration repose sur un traitement individuel des pixels de l'image finale. Ce traitement est particulièrement prédisposé à une adaptation utilisant au mieux le processeur de la carte graphique.

Le rendu à base de vidéo est le croisement de plusieurs disciplines comme la reconstruction géométrique, la synthèse d'images, la vision par ordinateur, l'optique, la réalité virtuelle ou encore la psychologie. Ses applications sont donc nombreuses notamment dans les domaines du jeu vidéo, de la télévision 3d, de la téléconférence, de la simulation ou de la sécurité.

Dans la seconde partie de notre étude, je me suis particulièrement intéressé aux applications en réalité virtuelle et plus particulièrement à la création de paires d'images stéréoscopiques permettant de visualiser des images en relief. La majorité des méthodes de rendu à base d'images nécessitent deux rendus de la scène pour générer une paire d'images stéréoscopiques (une image pour chaque œil). Je propose une adaptation des *Plane Sweep* permettant d'accélérer la création de paires d'images stéréoscopiques en générant la seconde vue à moindre coût. Cette optimisation est basée sur un partage des données relatives à la scène communes aux deux vues stéréoscopiques.

Enfin, j'aborderai une partie plus spécifique à la réalité virtuelle qui concerne les mouvements pseudoscopiques lors de la visualisation d'images stéréoscopiques sur un grand écran. Ces mouvements pseudoscopiques apparaissent lorsque l'observateur se déplace par rapport à l'écran. Si l'image affichée n'est pas modifiée, l'observateur verra les objets de la scène se déplacer de façon anormale et sa perception du relief s'en trouvera perturbée. Je propose une méthode simple et souple d'utilisation permettant de corriger cette distorsion de la perception géométrique de la scène en relief. Cette méthode s'applique d'une part aux techniques de synthèse d'images classiques dédiées au rendu stéréoscopique et d'autre part à la méthode des *Plane Sweep*. Elle est applicable sur ces techniques de façon simple et impose très peu de nouvelles contraintes sur les paramètres de caméras.

Les travaux traitant de ma nouvelle approche des *Plane Sweep* présentés dans ce mémoire ont été publiés dans Nozick et al. [NMA05, NMA06]. Quant à ma méthode de correction des mouvements pseudoscopiques, elle été publiée dans Nozick et Biri [NB03]. Il est prévu de poursuivre ces travaux de recherche sur le rendu à base de vidéos dans le cadre d'un post-doc dans le laboratoire du professeur Hideo Saito de l'Université de Keio, à Tokyo au Japon.

Première partie
Rendu à base d'images

1. Vision par Ordinateur, Dispositifs de Capture

Ce chapitre introduit les outils nécessaires à une bonne compréhension des méthodes de rendu à base d'images, de rendu à base de vidéos et de reconstruction. Il permet aussi de définir les notations que nous utiliserons tout au long de ce document.

Les méthodes de rendu à base d'images utilisent très largement les outils disponibles en vision par ordinateur. Parmi eux, nous détaillerons particulièrement les méthodes de calibrage de caméras permettant de gérer les informations de position entre la scène et la caméra. Nous proposons aussi une vue d'ensemble de la géométrie épipolaire décrivant quelques propriétés des relations entre deux images d'une même scène. Enfin, nous détaillons les connaissances relatives à la capture vidéo d'une scène, notamment la gestion et la disposition des caméras.

1.1 Géométrie projective

La géométrie projective est un outil puissant largement utilisé en vision par ordinateur ou en synthèse d'image. Plutôt que d'utiliser les coordonnées cartésiennes, elle utilise les coordonnées homogènes qui permettent de gérer des notions de points à l'infini et facilitent les transformations linéaires.

1.1.1 Deux dimensions : $\mathbb{P}^2$

Dans $\mathbb{P}^2$ (coordonnées homogènes dans le plan), un point du plan se note $\mathbf{x} = (x,y,w)^\top$ et une droite $\mathbf{l} = (a,b,c)^\top$. Voici quelques propriétés des points et des droites de $\mathbb{P}^2$:

- $\mathbf{x} \in \mathbf{l}$ si et seulement si $\mathbf{x}^\top\mathbf{l} = \mathbf{l}^\top\mathbf{x} = 0$ (produit scalaire de $\mathbb{R}^3$)
- $\mathbf{x}$ et $k\mathbf{x}$ ($k \in \mathbb{R}^*$) représentent le même point.

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} x/w \\ y/w \\ 1 \end{pmatrix}$$

  où x/w et y/w représentent les coordonnées cartésiennes du point dans le plan pour $w \neq 0$.

- $\mathbf{l}$ et $k\mathbf{l}$ ($k \in \mathbb{R}^*$) représentent la même droite.
- La droite $\mathbf{l}$ passant par deux points $\mathbf{x_1}$ et $\mathbf{x_2}$ vaut : $\mathbf{l} = \mathbf{x_1} \times \mathbf{x_2}$ (où $\times$ représente le produit vectoriel de $\mathbb{R}^3$).
- Le point d'intersection $\mathbf{x}$ de 2 droites $\mathbf{l_1}$ et $\mathbf{l_2}$ se calcule avec : $\mathbf{x} = \mathbf{l_1} \times \mathbf{l_2}$. Notez le caractère dual entre les points et les droites dans $\mathbb{P}^2$.
- Trois points alignés $\mathbf{x_1}$, $\mathbf{x_2}$ et $\mathbf{x_3}$ vérifient $\det[\mathbf{x_1}\ \mathbf{x_2}\ \mathbf{x_3}] = 0$.
- Trois droites concourantes $\mathbf{l_1}$, $\mathbf{l_2}$ et $\mathbf{l_3}$ vérifient $\det[\mathbf{l_1}\ \mathbf{l_2}\ \mathbf{l_3}] = 0$.
- Le point $(x,y,0)^\top$ correspond à un point à l'infini aussi appelé point idéal (*ideal point* en anglais).

- Deux droites parallèles se coupent en un point à l'infini.

1.1.2 Trois dimensions : $\mathbb{P}^3$

$\mathbb{P}^3$ (coordonnées homogènes dans l'espace) présente les mêmes propriétés que $\mathbb{P}^2$ mais le caractère dual intervient entre les points et les plans et non entre les points et les droites. Une droite dans $\mathbb{P}^3$ peut être représentée comme l'intersection de 2 plans. Pour plus de détails, voir les matrices de Plücker décrites en détail dans Hartley et Zisserman [HZ04].

1.1.3 Homographie dans $\mathbb{P}^2$

Les homographies, transformations projectives des plans, sont très largement utilisées en vision par ordinateur ou en rendu à base de vidéos. Comme illustré sur la figure 1, pour transformer une image, il suffit d'appliquer une homographie à tous ses pixels. Une telle transformation peut s'écrire sous forme matricielle de la façon suivante :

$$\boldsymbol{x}' = H\boldsymbol{x}$$

où $\mathbf{x} \in \mathbb{P}^2$ est un point de l'image de départ et $\mathbf{x}' \in \mathbb{P}^2$ un point de l'image d'arrivée.

Figure 1 : homographie.

Une homographie est une matrice 3×3 et possède donc 9 éléments indépendants. Cette matrice est utilisée en coordonnées homogènes, elle est donc invariante par changement d'échelle et on peut alors la multiplier par une constante réelle non nulle sans changer ses effets sur une image. Une homographie a donc 8 degrés de liberté et elle est déterminée par un minimum de 4 correspondances de points (qui ont chacun 2 degrés de liberté).

Le calcul d'une homographie s'effectue à partir d'un ensemble de correspondances $\mathbf{x_i} \leftrightarrow \mathbf{x_i}'$ (i = 1, …, nombre de correspondances) entre des pixels de l'image de départ et des pixels de l'image d'arrivée. La matrice H doit alors vérifier $\mathbf{x_i}' = H\ \mathbf{x_i}$ pour chaque correspondance $\mathbf{x_i} \leftrightarrow \mathbf{x_i}'$. Pour trouver la matrice H vérifiant $\mathbf{x_i}' = H\ \mathbf{x_i}$, il est préférable de partir de $\mathbf{x_i}' \times H\ \mathbf{x_i} = \boldsymbol{0}$ (où × correspond au produit vectoriel de $\mathbb{R}^3$). En séparant les lignes de H, on obtient :

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} = \begin{pmatrix} \mathrm{h}_1^{\mathrm{T}} \\ \mathrm{h}_2^{\mathrm{T}} \\ \mathrm{h}_3^{\mathrm{T}} \end{pmatrix}$$

Posons

$$\mathbf{x_i} = \begin{pmatrix} x_i \\ y_i \\ w_i \end{pmatrix} \text{ et } \mathbf{x'_i} = \begin{pmatrix} x'_i \\ y'_i \\ w'_i \end{pmatrix}$$

On obtient alors

$$\begin{pmatrix} x'_i \\ y'_i \\ w'_i \end{pmatrix} \times \begin{pmatrix} \mathbf{h_1^T x_i} \\ \mathbf{h_2^T x_i} \\ \mathbf{h_3^T x_i} \end{pmatrix} = \begin{pmatrix} y'_i \mathbf{h_3^T x_i} - w'_i \mathbf{h_2^T x_i} \\ w'_i \mathbf{h_1^T x_i} - x'_i \mathbf{h_3^T x_i} \\ x'_i \mathbf{h_2^T x_i} - y'_i \mathbf{h_1^T x_i} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

soit

$$\begin{pmatrix} 000 & -w'_i \mathbf{x_i^T} & y'_i \mathbf{x_i^T} \\ w'_i \mathbf{x_i^T} & 000 & x'_i \mathbf{x_i^T} \\ -y'_i \mathbf{x_i^T} & x'_i \mathbf{x_i^T} & 000 \end{pmatrix} \begin{pmatrix} \mathbf{h_1} \\ \mathbf{h_2} \\ \mathbf{h_3} \end{pmatrix} = \begin{pmatrix} 000 & -w'_i \mathbf{x_i^T} & y'_i \mathbf{x_i^T} \\ w'_i \mathbf{x_i^T} & 000 & x'_i \mathbf{x_i^T} \\ -y'_i \mathbf{x_i^T} & x'_i \mathbf{x_i^T} & 000 \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Il se trouve que la troisième ligne du système est une combinaison linéaire des deux premières, on peut donc s'en passer. Le système à résoudre doit prendre en compte les *n* correspondances de points et il devient alors :

$$\begin{pmatrix} 000 & -w'_1 \mathbf{x_1^T} & y'_1 \mathbf{x_1^T} \\ w'_1 \mathbf{x_1^T} & 000 & -x'_1 \mathbf{x_1^T} \\ \vdots & \vdots & \vdots \\ 000 & -w'_n \mathbf{x_n^T} & y'_n \mathbf{x_n^T} \\ w'_n \mathbf{x_n^T} & 000 & -x'_n \mathbf{x_n^T} \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

L'équation obtenue est en général surdéterminée (le nombre d'équations est supérieur au nombre d'inconnues) et peut être résolue au sens des moindres carrés par une décomposition en valeurs singulières. Pour plus de détails sur les homographies et leur calcul par des méthodes robustes, le lecteur peut se référer à l'ouvrage de Hartley et Zisserman [HZ04].

1.2 Camera projective

Par caméra, nous entendons aussi bien un appareil photo, une caméra vidéo, un œil humain, une caméra *fish-eye*, une « *pushbroom camera* » pour les satellites ou les photocopieuses ou bien encore un radiotélescope ou un microscope électronique. Il est clair que l'on ne peut pas modéliser tous ces dispositifs de la même façon et nous nous limiterons aux appareils photo et aux caméras vidéo. On décrit ces dernières avec le modèle de caméra « trou d'épingle » (*pinehole camera* en anglais). Ce modèle, largement utilisé en vision par ordinateur, décrit aussi bien la projection perspective que la projection orthogonale. Il fonctionne à partir des concepts de la géométrie projective et par conséquent il nécessite l'emploi d'une matrice de projection.

Ce modèle de camera perspective est relativement similaire au modèle de caméra utilisé en synthèse d'images. La principale différence vient surtout de son mode de calcul : alors qu'en synthèse d'images on connaît la position de la caméra et on en déduit sa matrice de projection, il faut, en vision par ordinateur, extraire la matrice de projection à partir d'une image prise avec une caméra réelle. Cette caméra réelle a une position et une orientation dans le repère de la scène ainsi que des caractéristiques qui lui sont propres (distance focale, angle de vue…). C'est durant le processus de calibrage que l'on extrait ces informations de l'image pour en déduire une matrice de projection.

Les paragraphes suivants traitent d'une part des caractéristiques du modèle de caméra perspective et notamment de la matrice de projection. D'autre part, ils détaillent l'étape de calibrage.

1.2.1 Matrice de projection

Une matrice de projection projette un point de l'espace sur une image, autrement dit elle projette un point de $\mathbb{P}^3$ sur un point de $\mathbb{P}^2$ (figure 2). La matrice de projection P est donc une matrice 3×4. P n'est pas une matrice 4×4 comme souvent en synthèse d'image. En effet, certaines bibliothèques graphiques comme OpenGL par exemple utilisent aussi la matrice de projection pour le clipping, c'est-à-dire pour la gestion des facettes qui débordent sur le bord de l'image et qu'il ne faut pas afficher. La matrice de projection 4×4 correspond en fait à une homographie 3d de la pyramide de projection sur le cube unitaire suivie d'une projection orthogonale. Durant l'homographie 3d, l'information de profondeur du point projeté n'est pas perdue.

Soit $\mathbf{X}\in\mathbb{P}^3$ un point de la scène et $\mathbf{x}\in\mathbb{P}^2$ le pixel de l'image correspondant au projeté de $\mathbf{X}$. La projection se note :

$$\mathbf{x}=P\mathbf{X}$$

En coordonnées homogènes, on obtient :

$$\begin{pmatrix} x/w \\ y/w \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{bmatrix} p_{11} p_{12} p_{13} p_{14} \\ p_{21} p_{22} p_{23} p_{24} \\ p_{31} p_{32} p_{33} p_{34} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix}$$

où les p_{ij} sont les éléments de P d'indice (i,j).

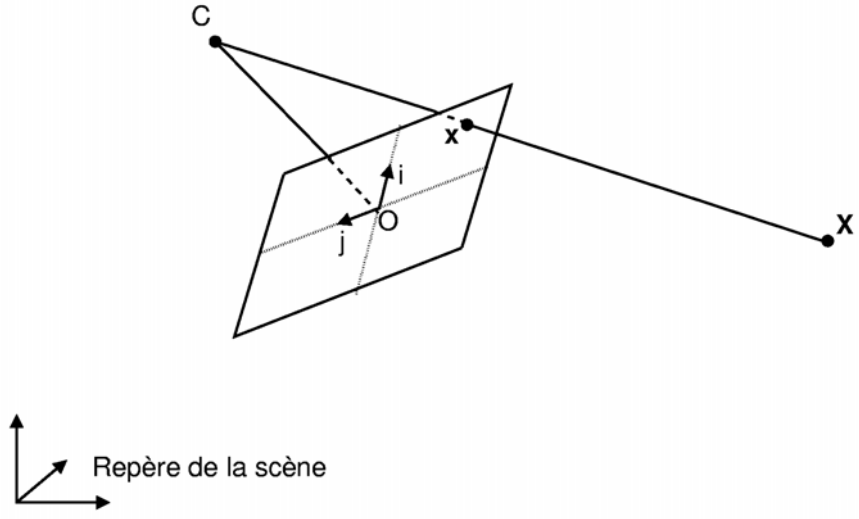


Figure 2 : repère de la caméra et repère de la scène.

La matrice P est invariante par changement d'échelle et est de rang 3, elle a donc 11 degrés de liberté. On peut la décomposer de la façon suivante :

$$P = [M|-MC]$$

où M est une matrice 3×3 et C correspond à la position (non homogène) du centre de la caméra dans le repère de la scène. La matrice M peut s'écrire comme le produit de deux matrices :

$$M = KR$$

où K est une matrice 3×3 triangulaire supérieure du type :

$$\begin{bmatrix} \alpha_i & s & x_0 \\ 0 & \alpha_j & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

avec :

- α_i : distance focale sur l'axe $(0i)$ du plan de projection (en unité de pixels).
- α_j : distance focale sur l'axe $(0j)$ du plan de projection (en unité de pixels).
- s : (*skew* en anglais) facteur de "verticalité" du plan de projection. Si $s \neq 0$, le plan de projection est un losange. Cela peut arriver sur certaines caméras CCD où les capteurs ne sont pas alignés sous forme de rectangle.

Les valeurs α_i et α_j peuvent être différentes si les pixels de la caméra CCD ne sont pas carrés. Cela revient alors à dire que la distance focale sur l'axe (*0i*) est différente de celle sur l'axe (*0j*). La matrice *R* est une matrice 3×3. Elle correspond à une matrice de rotation définissant l'orientation de la caméra.

1.2.2 Extraction des paramètres d'une matrice de projection

A partir de la matrice de projection *P*, il est possible d'extraire les paramètres de la caméra. Pour cela, il suffit de décomposer la matrice *P* en *P=[M|-MC]* avec *M=KR*.

1.2.2.1 Matrices *M*, *K* et *R*

Pour une matrice de projection *P*=[*M*|-*MC*], l'extraction de la matrice *M* est immédiate, il s'agit juste de prendre les trois premières colonnes de *P*. La matrice *M* se décompose en *M=KR* où *K* est triangulaire supérieure. On identifie *K* et *R* en effectuant une décomposition RQ sur *M*. La décomposition RQ pour des matrices 3×3 est spécialement simple [HZ04]. En ce qui concerne la matrice *K*, il ne faut pas oublier de la diviser par k_{33}.

1.2.2.2 Centre *C* de la caméra

Le centre de la caméra obéit à la propriété suivante : $P\mathbf{C}=\mathbf{0}$ (Hartley et Zisserman [HZ04]). On en déduit les relations suivantes :

$$x = \det([\mathbf{p_2}\ \mathbf{p_3}\ \mathbf{p_4}])$$

$$y = -\det([\mathbf{p_1}\ \mathbf{p_3}\ \mathbf{p_4}])$$

$$z = \det([\mathbf{p_1}\ \mathbf{p_2}\ \mathbf{p_4}])$$

$$w = -\det([\mathbf{p_1}\ \mathbf{p_2}\ \mathbf{p_3}])$$

où les $\mathbf{p_i}$ correspondent à la $i^{ème}$ colonne de *P*

1.2.3 Calibrage de la caméra

Pour calculer la matrice *P*, il est nécessaire de disposer d'un ensemble de correspondances $\mathbf{X_i} \leftrightarrow \mathbf{x_i}$ (points/pixels). Il est possible d'obtenir ce genre de correspondances à partir d'une photo d'une mire dont les dimensions sont connues (figure 3).

Soit $\mathbf{P_1}^\top$ le vecteur de dimension 4 correspondant à la première ligne de la matrice *P*, $\mathbf{P_2}^\top$ celui correspondant à la seconde ligne et $\mathbf{P_3}^\top$ à la 3ème ligne.

$$P=\begin{bmatrix}\mathbf{P_1^T}\\ \mathbf{P_2^T}\\ \mathbf{P_3^T}\end{bmatrix}=\begin{bmatrix}p_{11}\,p_{12}\,p_{13}\,p_{14}\\ p_{21}\,p_{22}\,p_{23}\,p_{24}\\ p_{31}\,p_{32}\,p_{33}\,p_{34}\end{bmatrix}$$

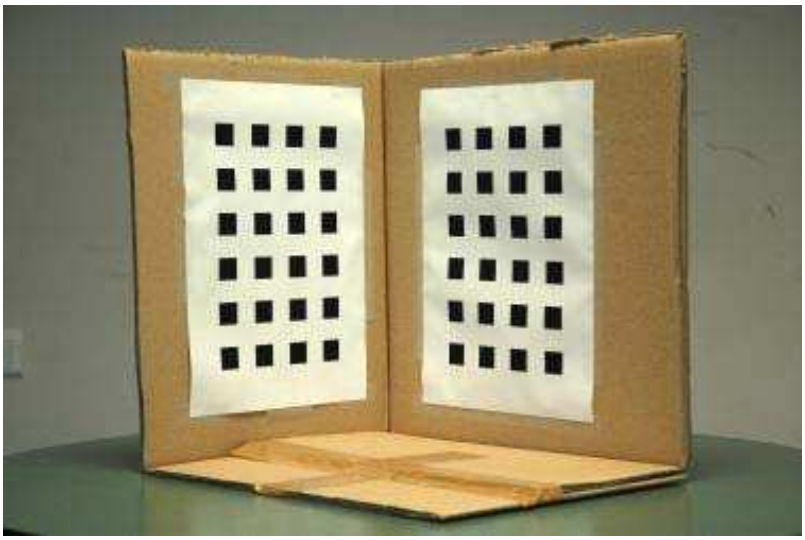


Figure 3 : mire de calibrage.

En appliquant la formule de la matrice de projection $\mathbf{x_i} = P\mathbf{X_i}$, on obtient :

$$\begin{pmatrix} x_i \\ y_i \\ w_i \end{pmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ W_i \end{pmatrix} \text{ qui équivaut à } \begin{pmatrix} x_i \\ y_i \\ w_i \end{pmatrix} = \begin{bmatrix} \mathbf{P_1^T} \\ \mathbf{P_2^T} \\ \mathbf{P_3^T} \end{bmatrix} \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ W_i \end{pmatrix} = \begin{bmatrix} \mathbf{P_1^T} \\ \mathbf{P_2^T} \\ \mathbf{P_3^T} \end{bmatrix} \mathbf{X}_i$$

soit

$$\begin{pmatrix} x_i \\ y_i \\ w_i \end{pmatrix} = \begin{pmatrix} \mathbf{X_i}.\mathbf{P_1^T} \\ \mathbf{X_i}.\mathbf{P_2^T} \\ \mathbf{X_i}.\mathbf{P_3^T} \end{pmatrix}$$

$$\begin{pmatrix} x_i \\ y_i \\ w_i \end{pmatrix} \times \begin{pmatrix} \mathbf{X_i}.\mathbf{P_1^T} \\ \mathbf{X_i}.\mathbf{P_2^T} \\ \mathbf{X_i}.\mathbf{P_3^T} \end{pmatrix} = \mathbf{0}$$

(l'opérateur $\times$ représente le produit vectoriel de $\mathbb{R}^3$)

$$\begin{bmatrix} \mathbf{0^T} & -w_i\mathbf{X_i^T} & y_i\mathbf{X_i^T} \\ w_i\mathbf{X_i^T} & \mathbf{0^T} & -x_i\mathbf{X_i^T} \\ -y_i\mathbf{X_i^T} & x_i\mathbf{X_i^T} & \mathbf{0^T} \end{bmatrix} \begin{pmatrix} \mathbf{P_1} \\ \mathbf{P_2} \\ \mathbf{P_3} \end{pmatrix} = \mathbf{0}$$

On ne garde que les deux premières équations puisque la troisième est une combinaison linéaire des deux premières. On obtient ainsi :

$$\begin{bmatrix} \mathbf{0^T} & -w_i\mathbf{X_i^T} & y_i\mathbf{X_i^T} \\ w_i\mathbf{X_i^T} & \mathbf{0^T} & -x_i\mathbf{X_i^T} \end{bmatrix} \begin{pmatrix} \mathbf{P_1} \\ \mathbf{P_2} \\ \mathbf{P_3} \end{pmatrix} = \mathbf{0}$$

avec

$$\begin{pmatrix} \mathbf{P_1} \\ \mathbf{P_2} \\ \mathbf{P_3} \end{pmatrix} = \begin{pmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{pmatrix}$$

On utilise ce système pour tous les couples $\mathbf{X_i} \leftrightarrow \mathbf{x_i}$.

$$\begin{bmatrix} \mathbf{0}^\mathbf{T} & -w_1\mathbf{X}_1^\mathbf{T} & y_1\mathbf{X}_1^\mathbf{T} \\ w_1\mathbf{X}_1^\mathbf{T} & \mathbf{0}^\mathbf{T} & -x_1\mathbf{X}_1^\mathbf{T} \\ \mathbf{0}^\mathbf{T} & -w_2\mathbf{X}_2^\mathbf{T} & y_2\mathbf{X}_2^\mathbf{T} \\ w_2\mathbf{X}_2^\mathbf{T} & \mathbf{0}^\mathbf{T} & -x_2\mathbf{X}_2^\mathbf{T} \\ & \vdots & \\ \mathbf{0}^\mathbf{T} & -w_n\mathbf{X}_n^\mathbf{T} & y_n\mathbf{X}_n^\mathbf{T} \\ w_n\mathbf{X}_n^\mathbf{T} & \mathbf{0}^\mathbf{T} & -x_n\mathbf{X}_n^\mathbf{T} \end{bmatrix} \begin{pmatrix} \mathbf{P_1} \\ \mathbf{P_2} \\ \mathbf{P_3} \end{pmatrix} = \mathbf{0}$$

La matrice obtenue est une matrice $2n \times 12$ où n est le nombre de correspondances utilisées. Il faut utiliser au moins 5,5 points pour satisfaire les 11 degrés de liberté de la matrice P. Il est toutefois fortement recommandé d'en utiliser beaucoup plus afin d'obtenir un résultat plus précis. Cette méthode est connue sous le nom de DLT (*Direct Linear Transformation*). La décomposition en valeurs singulières, notée SVD (de l'anglais *Singular Value Decomposition*) permet de résoudre ce genre de systèmes surdéterminés (où le nombre d'équations est supérieur au nombre d'inconnues) en utilisant des méthodes de moindres carrés. La SVD permet de résoudre ce type de systèmes ($A\mathbf{x}=\mathbf{0}$) sans donner la solution triviale $\boldsymbol{x}=\mathbf{0}$ qui n'est pas spécialement intéressante. Pour plus de détails sur la SVD, le lecteur peut se reporter à l'annexes 1.

Il est possible de résoudre ce système directement avec la matrice pseudo-inverse en posant $p_{34} = 1$. En effet, la matrice P est invariante par facteur d'échelle. Le système $A\mathbf{x}=\mathbf{0}$ peut alors se réécrire avec une colonne en moins pour A et une ligne en moins pour $\boldsymbol{x}$. Cette méthode est cependant peu stable numériquement dans le cas où p_{34} est proche de 0.

1.2.4 Le « Gold Standard Algorithm »

Le *Gold Standard Algorithm* (Hartley et Zisserman [HZ04]) est une adaptation de la méthode DLT décrite dans le paragraphe précédent (§1.2.3). Elle comporte deux étapes supplémentaires : la normalisation des données pour rendre le système plus stable numériquement et une minimisation non linéaire de l'erreur. Le *Gold Standard Algorithm* se décompose ainsi :

- Calculer la transformation T (matrice 3×3) permettant de cadrer les pixels utilisés pour le calibrage au centre de l'image et opérant une homothétie pour que la distance moyenne entre ces points et l'origine de l'image soit $\sqrt{2}$. Pour cela, il faut calculer le barycentre (b_x,b_y) des pixels et la distance moyenne d entre les pixels et ce point. La matrice T s'écrit :

$$T = \begin{bmatrix} \frac{\sqrt{2}}{d} & 0 & -\frac{\sqrt{2}}{d} b_x \\ 0 & \frac{\sqrt{2}}{d} & -\frac{\sqrt{2}}{d} b_y \\ 0 & 0 & 1 \end{bmatrix}$$

  et son inverse :

$$T^{-1} = \begin{bmatrix} \frac{d}{\sqrt{2}} & 0 & b_x \\ 0 & \frac{d}{\sqrt{2}} & b_y \\ 0 & 0 & 1 \end{bmatrix}$$

- Appliquer la transformation T à tous les pixels de la liste de corrélation.
- Trouver une transformation U (matrice 4×4) permettant de centrer les points de la scène utilisés pour le calibrage sur l'origine du repère et opérant une homothétie pour que la distance moyenne entre ces points et l'origine du repère soit $\sqrt{3}$. Son mode de calcul est similaire à celui de T.
- Appliquer la transformation U à tous les points de la liste de corrélation.
- Calculer P' en appliquant la méthode DLT du paragraphe précédent sur les points et pixels normalisés.
- A partir de l'estimation de P', minimiser l'erreur $\sum_i d(x_i,P'X_i)^2$ avec une méthode de résolution d'équations non linéaires telle que la méthode de Levenberg-Marquardt.
- $P=T^{-1}P'U$

1.2.5 Autres méthodes de calibrage

La méthode de Zhang [Z00] est utilisée dans de nombreuses applications tant elle est efficace et simple à mettre en œuvre. Cette méthode utilise une série de clichés d'une mire composée de points coplanaires. Pour chacun de ces clichés, une homographie entre la mire (dans ses proportions réelles) et son projeté est calculée. En combinant toutes ces homographies, il est possible de calculer les paramètres de la caméra. Après une première évaluation de ces paramètres, une évaluation des coefficients de correction de distorsion radiale est effectuée. Pour cela, il suffit d'utiliser comme points de correspondance les projetés des points de la mire par homographie avec les points détectés sur la mire. Une fois ces paramètres définis, il suffit de minimiser l'erreur globale avec une méthode de résolution d'équations non linéaires telle que la méthode de Levenberg-Marquardt en agissant sur tous les paramètres préalablement calculés. Il existe plusieurs implémentations de cette méthode de calibrage accessibles sur internet en Matlab ou sous OpenCV [I00].

Dans le cas où il est nécessaire de calibrer plusieurs caméras, soit celles-ci sont proches les unes des autres et une mire commune peut être utilisée, soit les caméras sont très espacées et le calibrage se fait en deux temps. Il faut d'abord définir les paramètres intrinsèques de chaque caméra (traitée individuellement), puis calculer la localisation de chaque caméra les unes par rapport aux autres avec des points de correspondance. Dans une application au rendu à base de vidéos, certains réalisateurs n'hésitent pas à transformer le studio de tournage en mire géante afin de simplifier la procédure de calibrage.

1.2.6 Correction de la distorsion radiale

Le modèle de caméra décrit dans les paragraphes précédents (chapitre 1.1.3) est un modèle de caméra linéaire et ne tient pas compte des défauts des optiques des caméras, notamment de la distorsion géométrique des images. Cette distorsion peut être de deux types : tangentielle ou radiale et dans les deux cas, elle révèle un caractère non linéaire. La figure 4 illustre ces deux distorsions en représentant en traits pleins des parties de l'image non distordue et en pointillés leurs correspondants distordus. La distorsion tangentielle est souvent négligeable et il n'est en général pas nécessaire de la corriger. Ce n'est pas le cas de la distorsion radiale dont les effets sont particulièrement visibles sur les objectifs bas de gammes, les objectifs à focale faible (grands angles) ou sur les zooms.

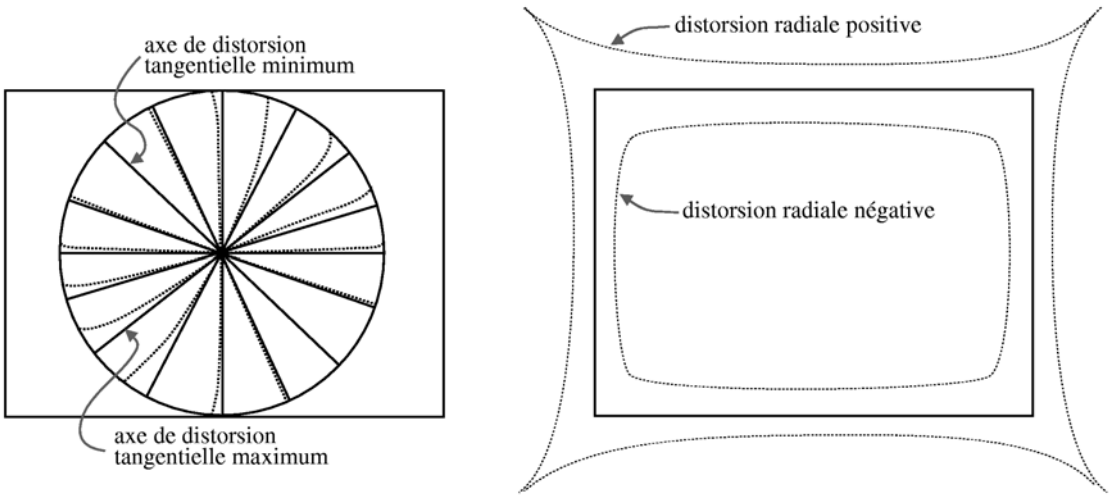


Figure 4 : distorsion tangentielle et radiale.

La distorsion radiale peut être modélisée par un polynôme vérifiant certaines contraintes (Faugeras et Luong [FL04]). Soit (x_c,y_c) le centre de distorsion, il s'agit en général du « point principal » (de l'anglais *principal point*) qui correspond à l'intersection de l'axe optique avec l'image. Il arrive que le centre de distorsion soit décentré par rapport à l'image dans le cas d'assemblages d'optiques de mauvaise qualité. Soit (x_i,y_i) un ensemble de points sur l'image distordue et $(\hat{x}_i, \hat{y}_i)$ les points correspondants sur l'image rectifiée (figure 5).

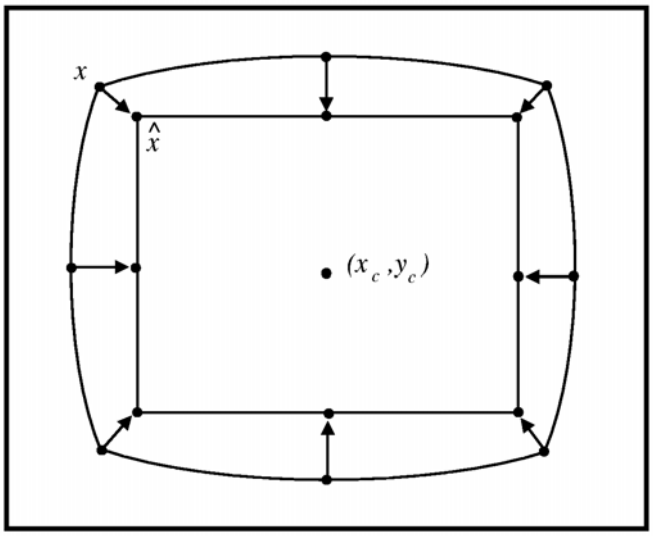


Figure 5 : correction de la distorsion radiale.

La relation entre ces deux ensembles de points peut être définie par un polynôme (Ray [R02]). Nous utilisons dans ces équations un polynôme de degré 4 en ne gardant que les puissances paires, ce qui est réputé pour être suffisant (Faugeras et Luong [FL04]). Il est toutefois possible d'utiliser un polynôme de degrés 6 (Forsyth et Ponce [FP02]), toujours en ne gardant que les puissances paires, ou bien d'utiliser toutes les puissances d'un polynôme de degré 4 (Hartley et Zisserman [HZ04]).

La relation entre ces deux ensembles de points peut s'écrire de la façon suivante :

$$\begin{cases} \hat{x}_i = x_i + \Delta x_i \\ \hat{y}_i = y_i + \Delta y_i \end{cases}$$

avec

$$\begin{cases} \Delta x_i = (x_i - x_c)(k_1 r^2 + k_2 r^4) \\ \Delta y_i = (y_i - y_c)(k_1 r^2 + k_2 r^4) \end{cases}$$

et

$$r = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2}$$

soit

$$\begin{cases} \hat{x}_i = x_i + (x_i - x_c)(k_1 r^2 + k_2 r^4) \\ \hat{y}_i = y_i + (y_i - y_c)(k_1 r^2 + k_2 r^4) \end{cases}$$

La relation précédente se note aussi :

$$\begin{cases} \dfrac{\hat{x}_i - x_i}{x_i - x_c} = k_1 r^2 + k_2 r^4 \\ \dfrac{\hat{y}_i - y_i}{y_i - y_c} = k_1 r^2 + k_2 r^4 \end{cases}$$

Pour *n* points de correspondance, il suffit donc de résoudre le système suivant :

$$\begin{bmatrix} r_1^2 & r_1^4 \\ r_1^2 & r_1^4 \\ \vdots & \vdots \\ r_n^2 & r_n^4 \\ r_n^2 & r_n^4 \end{bmatrix} \begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} \dfrac{\hat{x}_1 - x_1}{x_1 - x_c} \\ \dfrac{\hat{y}_1 - y_1}{y_1 - y_c} \\ \vdots \\ \dfrac{\hat{x}_n - x_n}{x_n - x_c} \\ \dfrac{\hat{y}_n - y_n}{y_n - y_c} \end{pmatrix}$$

Cette méthode permet de définir la fonction transformant un point distordu en point corrigé (figure 6). Notons que la fonction inverse n'est pas triviale à trouver et nécessite la résolution de polynômes de degré 3. Il est parfois intéressant de calculer la correction inverse qui transforme un point corrigé en point distordu. Pour cela, il suffit d'intervertir les (x_i,y_i) et les $(\hat{x}_i, \hat{y}_i)$ dans le système à résoudre et de remplacer *r* par $\hat{r} = \sqrt{(\hat{x}_i - x_c)^2 + (\hat{y}_i - y_c)^2}$. Les correspondances de pixels peuvent être établies en s'inspirant de la méthode de calibrage de Zhang [Z00] qui consiste à associer des points distordus avec leur projection idéale calculée par homographie.

image distordue

image corrigée

Figure 6 : exemple de correction de distorsion radiale.

Il existe d'autres méthodes moins contraignantes permettant de corriger la distorsion radiale comme celle proposée par Devernay et Faugeras [DF01]. Cette méthode redresse les courbes supposées droites dans l'image corrigée. Encore faut il que des droites figurent sur l'image à corriger. De plus, ces droites doivent être identifiées manuellement.

1.3 Géométrie épipolaire

Il existe une relation géométrique entre 2 images d'une même scène. Cette relation est décrite par la géométrie épipolaire et permet de donner une indication de la position (en pixels) d'un objet sur une image connaissant la position de ce même objet sur l'autre image. Cette relation entre les deux images peut s'exprimer sous forme matricielle à l'aide de la "matrice fondamentale" *F*.

1.3.1 Matrice fondamentale

Etant donné une de correspondance $\mathbf{x}^{\mathbf{A}} \leftrightarrow \mathbf{x}^{\mathbf{B}}$ où $\mathbf{x}^{\mathbf{A}}$ et $\mathbf{x}^{\mathbf{B}}$ sont les coordonnées homogènes en pixels d'un même point sur les images *A* et *B*, la matrice fondamentale *F* obéit à la formule :

$$\mathbf{x}^{\mathbf{B}\top} F \mathbf{x}^{\mathbf{A}} = 0$$

où *F* est une matrice 3×3. En géométrie projective à 2 dimensions ($\mathbb{P}^2$), une droite du plan et un point sont de même nature. On peut donc considérer le vecteur $F\mathbf{x}^{\mathbf{A}}$ comme une droite et le fait d'avoir $\mathbf{x}^{\mathbf{B}\top} F \mathbf{x}^{\mathbf{A}} = 0$ signifie que le point $\mathbf{x}^{\mathbf{B}}$ se trouve sur la droite $F\mathbf{x}^{\mathbf{A}}$ et réciproquement que le point $\mathbf{x}^{\mathbf{A}}$ se trouve sur la droite $(\mathbf{x}^{\mathbf{B}\top} F)^{\top} = F^{\top}\mathbf{x}^{\mathrm{B}}$.

Pour interpréter géométriquement la relation épipolaire entre deux images, considérons un pixel **x** de l'image *A* (figure 7). Ne possédant pas d'information sur la « profondeur » de ce pixel, c'est-à-dire sur la distance entre le pixel **x** et l'objet **X** qu'il représente, il n'est pas possible de définir à quel endroit sur la demie-droite $[\mathbf{C_A x})$ se trouve ce point **X**. Si l'on projette ces positions possibles de **X** sur l'image *B*, on obtient une droite, la droite épipolaire. Notons que la droite épipolaire correspond à l'intersection du plan $(\mathbf{X}, \mathbf{C_A}, \mathbf{C_B})$ avec le plan focal de la caméra *B* (figure 7). Ainsi, toutes les droites épipolaires de l'image *B* se croisent en un point $\mathbf{e_B}$ nommé épipôle. Il en va de même pour l'image *A* et son épipôle $\mathbf{e_A}$.

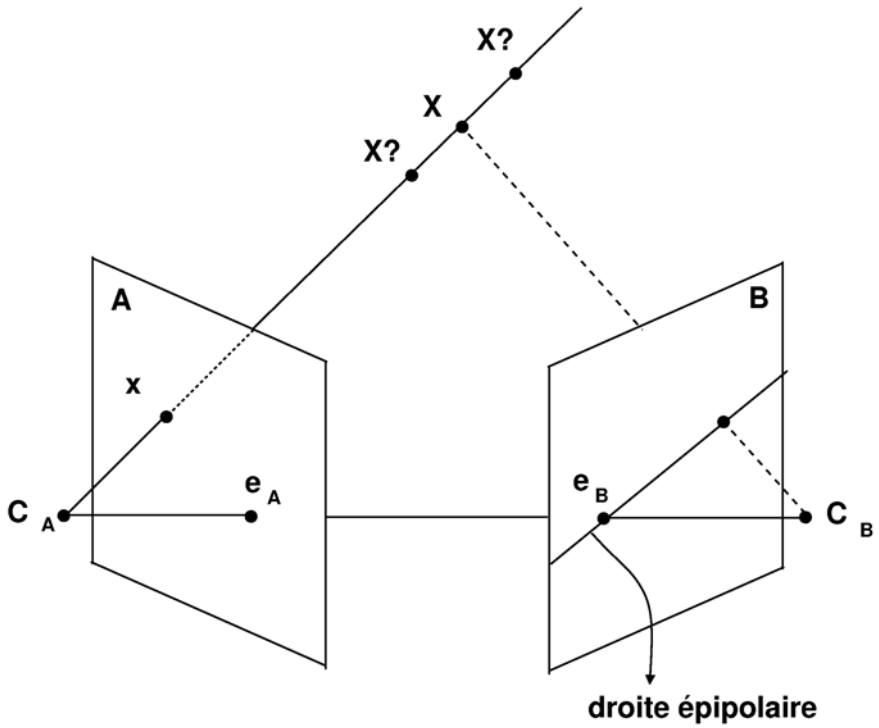


Figure 7 : géométrie épipolaire.

Pour le calcul de la matrice F, on procède de la même façon que pour la matrice de projection en partant de la formule $\mathbf{x}^{\mathbf{B}\top} F \mathbf{x}^{\mathbf{A}}=0$. En posant :

$$F = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}$$

Il faut pour chaque couple de pixels $\mathbf{x_i}^{\mathbf{A}} \leftrightarrow \mathbf{x_i}^{\mathbf{B}}$ satisfaire l'équation $\mathbf{x_i}^{\mathbf{B}\top} F \mathbf{x_i}^{\mathbf{A}}=0$:

$$x_i^A x_i^B f_{11} + x_i^A y_i^B f_{12} + x_i^A w_i^B f_{13} + y_i^A x_i^B f_{21} + y_i^A y_i^B f_{22} + y_i^A w_i^B f_{23} + w_i^A x_i^B f_{31} + w_i^A y_i^B f_{32} + w_i^A w_i^B f_{33} = 0$$

que l'on peut écrire sous forme matricielle :

$$\begin{bmatrix} x_1^A x_1^B & x_1^A y_1^B & x_1^A w_1^B & y_1^A x_1^B & y_1^A y_1^B & y_1^A w_1^B & w_1^A x_1^B & w_1^A y_1^B & w_1^A w_1^B \\ & & & & \vdots & & & & \\ x_n^A x_n^B & x_n^A y_n^B & x_n^A w_n^B & y_n^A x_n^B & y_n^A y_n^B & y_n^A w_n^B & w_n^A x_n^B & w_n^A y_n^B & w_n^A w_n^B \end{bmatrix} \begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

La matrice F est une matrice 3x3, elle possède donc 9 inconnues mais comme il s'agit d'une matrice homogène, elle est invariante par changement d'échelle. De plus, F satisfait Det $F = 0$ et n'a donc que 7 degrés de liberté. Cependant, si l'on utilise une méthode de résolution linéaire dans laquelle on ne peut pas inclure cette contrainte, il faut utiliser au moins 8 points de correspondance. Pour améliorer la précision des résultats, il est vivement conseillé d'en utiliser plus.

Comme pour le calcul de la matrice de projection, il est recommandé de normer et de centrer les données avant de résoudre le système pour des questions de stabilité numérique. Il est aussi conseillé (voir annexe 1) de « renforcer » la matrice fondamentale F en annulant sa plus petite valeur singulière. Cette étape s'effectue à l'aide de la SVD. Ce « renforcement » consiste à trouver la matrice F' la plus proche de F mais de rang inférieur et assure la propriété Det $F = 0$ qui garantit que les droites épipolaires se croisent aux épipôles (ce qui n'est pas forcément le cas sinon).

1.3.2 Droites épipolaires, épipôles

Connaissant le point $\mathbf{x}^{\mathrm{A}}$ de l'image A (ou le point $\mathbf{x}^{\mathrm{B}}$ de l'image B), on calcule les droites épipolaires $\mathbf{l}^{\mathrm{B}}$ (et $\mathbf{l}^{\mathrm{A}}$) de la manière suivante :

- $\mathbf{l}^{\mathrm{B}}=F\mathbf{x}^{\mathrm{A}}$
- $\mathbf{l}^{\mathrm{A}}=F^{\top}\mathbf{x}^{\mathrm{B}}$

Un épipôle est la projection du centre d'une des caméras sur l'autre caméra. Chaque droite épipolaire passe par son épipôle. Ceux-ci se calculent avec les relations suivantes :

- $F\mathbf{e}^{A}=0$
- $F^{T}\mathbf{e}^{B}=0$

Notons que si F est la matrice fondamentale entre l'image A et l'image B, alors F^{T} est la matrice fondamentale entre l'image B et l'image A.

1.4 Localisation d'un point dans l'espace

Etant données plusieurs caméras calibrées, il est possible de localiser un point dans l'espace à partir des coordonnées de son projeté sur les caméras. Cette méthode de triangularisation est souvent utilisée pour localiser un point dont les correspondances sur chaque image ont été établies. Il est aussi parfois nécessaire de localiser le projeté d'un pixel sur un plan dont on connaît les paramètres. Ces deux méthodes de localisation sont décrites dans les paragraphes suivants.

1.4.1 Localisation d'un point dans l'espace par triangularisation

Soient P_A et P_B deux matrices de projection associées aux images A et B. Connaissant un couple de pixels $\mathbf{x}^{A} \leftrightarrow \mathbf{x}^{B}$ correspondant au même objet dans la scène, il est possible, par triangulation, de retrouver les coordonnées 3d du point en question. Pour cela, on cherche l'intersection de la droite $(C_A,\mathbf{x}^{A})$ avec la droite $(C_B,\mathbf{x}^{B})$ où C_A et C_B sont les centres des deux caméras (figure 8).

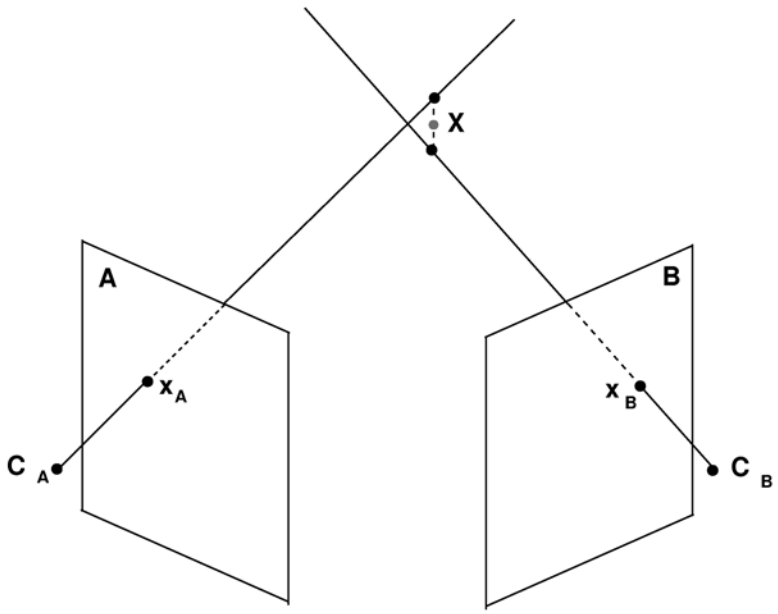


Figure 8 : triangularistion.

En pratique, ces deux droites ne se croisent jamais parfaitement, il faut donc trouver le point le plus proche des deux droites. Le calcul des intersections se fait à l'aide des matrices de projection des caméras. On pose les équations exprimant le fait que le point d'intersection $\mathbf{X}$ des deux droites se projette sur les pixels $\mathbf{x}^{A}$ et $\mathbf{x}^{B}$ puis on résout le système au sens des moindres carrés.

On obtient :

$$\mathbf{x^A} = P_\mathrm{A}\mathbf{X} \qquad \text{et} \qquad \mathbf{x^B} = P_\mathrm{B}\mathbf{X}$$

soit

$$\mathbf{x^A} \times P_\mathrm{A}\mathbf{X} = \mathbf{0} \qquad \text{et} \qquad \mathbf{x^B} \times P_\mathrm{B}\mathbf{X} = \mathbf{0}$$

En posant

$$\mathbf{x^A} = \begin{pmatrix} x^A \\ y^A \\ w^A \end{pmatrix}, \quad \mathbf{x^B} = \begin{pmatrix} x^B \\ y^B \\ w^B \end{pmatrix} \quad \text{et} \quad \mathbf{X} = \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix}$$

on obtient le système

$$\begin{cases} x^A(p_2^A X) - y^A(p_1^A X) = 0 \\ y^A(p_3^A X) - w^A(p_2^A X) = 0 \\ x^B(p_2^B X) - y^B(p_1^B X) = 0 \\ y^B(p_3^B X) - w^B(p_2^B X) = 0 \end{cases}$$

où les p_i^A correspondent aux i$^{\text{èmes}}$ lignes de la matrice P_A (et réspectivement pour P_B). Il suffit alors de résoudre

$$\begin{bmatrix} x^A p_2^A - y^A p_1^A \\ y^A p_3^A - w^A p_2^A \\ x^B p_2^B - y^B p_1^B \\ y^B p_3^B - w^B p_2^B \end{bmatrix} \mathbf{X=0}$$

La SVD permet là encore de résoudre ce genre de système en proposant une alternative à la solution triviale **X=0**.

1.4.2 Projeté d'un point sur un plan

Soit P la matrice de projection d'une caméra et **x** la position d'un pixel sur cette caméra. Soit $\boldsymbol{D} = (a,b,c,d)^\mathsf{T}$ un plan de $\mathbb{R}^3$. Le projeté **X** de **x** sur ***D*** correspond à l'intersection du rayon passant par **x** et le centre de la caméra avec le plan ***D***. Ce point **X** appartient à ***D*** et vérifie donc $\mathbf{X}^\mathsf{T}\boldsymbol{D}=0$. De plus, **X** est le projeté de **x** donc **x**=P**X**. Pour trouver **X**, il suffit donc de résoudre le système :

$$\left[\mathrm{P}^T \mid D\right]^T \mathrm{X} = \begin{pmatrix} \mathrm{x} \\ 0 \end{pmatrix}$$

soit

$$\begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ a & b & c & d \end{bmatrix} X = \begin{pmatrix} x_i \\ x_j \\ 1 \\ 0 \end{pmatrix}$$

1.5 Mise en place d'un dispositif d'acquisition vidéo

L'acquisition des images utilisées pour la reconstruction ou le rendu à base de vidéos est soumise à une série de contraintes que nous allons énumérer dans les parties qui suivent. Ces contraintes concernent notamment le choix des caméras, leur paramétrage, leur disposition, ainsi que la gestion du flux vidéo qu'elles génèrent.

1.5.1 Types de caméras

Concernant les applications de rendu à base de vidéos ou de vision par ordinateur, le choix des caméras vidéo est important. Il est par exemple souhaitable d'éviter les caméras à trames entrelacées qui diminuent la précision des images et qui nécessitent du temps à désentrelacer. Il reste alors le choix entre les caméras commerciales et les caméras de vision. Les drivers des caméras commerciales les plus répandues sont souvent accessibles gratuitement sur internet. Les drivers des caméras de vision ne sont quant à eux pas toujours gratuits. Cependant ces caméras sont souvent plus paramétrables, notamment au niveau de la résolution et de la compression du flux vidéo. Sur la plupart des caméras de vision par ordinateur, le flux vidéo n'est pas compressé, le transfert est donc plus long mais le traitement des données est plus rapide.

Par ailleurs, pour certaines applications où la restitution des couleurs est primordiale, il est conseillé d'utiliser des caméras TRI-CCD munies d'un capteur par couleur primaire plutôt que la restitution par filtre de Bayer, plus courante. Enfin, certaines applications peuvent se contenter d'une webcam dont la distorsion radiale et la mauvaise restitution des couleurs sont justifiées par des prix relativement bas. L'utilisation de webcams souligne particulièrement l'emploi de l'USB 2.0 comme alternative au Firewire. Pour finir, le choix la résolution des caméras utilisées dépend essentiellement de la méthode de reconstruction employée.

1.5.2 Disposition des caméras

Suivant la méthode de reconstruction ou de rendu à base d'images choisie, l'utilisateur bénéficie de plus ou moins de liberté concernant la disposition des caméras par rapport à la scène et les unes par rapport aux autres. Il y a quatre grandes catégories de méthodes : celles qui supportent et même encouragent la disposition des caméras tout autour de la scène. Cette catégorie représente une minorité des méthodes. Ensuite, certaines méthodes imposent juste aux caméras de filmer à peu près dans la même direction, peu importe la focale et le champ de vision. Dans ce cas, les caméras peuvent éventuellement être disposées selon un arc de cercle ou bien très loin les unes des autres, du moment que la scène est observée à peu près du même angle. La catégorie suivante concerne les méthodes à peu près similaires mais utilisant une caméra de référence selon la position de la caméra virtuelle. Dans ce cas, les caméras doivent être disposées relativement près les unes des autres afin d'éviter les

« sauts » sur l'image virtuelle lors d'un changement de caméra de référence. Enfin, les méthodes dites de « murs de caméras » (de l'anglais *camera array*) utilisent un grand nombre de caméras disposées sur un plan. Cette disposition coplanaire n'est souvent pas anodine puisqu'elle permet de simplifier les méthodes d'appariement de points entre les images.

Notons enfin qu'il est préférable de disposer les caméras de telle sorte que la profondeur de champ couvre la totalité de la scène. Les caméras sont souvent positionnées loin de la scène pour éviter d'avoir recours à des objectifs grands angles pour lesquels la distorsion radiale doit impérativement être corrigée.

1.5.3 Illumination

Il est suffisant de voir la façon dont est traitée l'éclairage dans le cinéma ou bien dans les studios de télévision pour se rendre compte à quel point il est important dans la mise en scène. Le rendu à base de vidéos impose de nouvelles contraintes venant s'ajouter aux contraintes artistiques.

En effet, une scène bien éclairée permet de fermer le diaphragme et d'obtenir ainsi une plus grande profondeur de champ. Une scène bien éclairée permet aussi de filmer avec une exposition courte et d'éviter ainsi le flou induit par du mouvement. Enfin, une scène bien éclairée limite l'augmentation du gain de la caméra et donc le bruit sur l'image. Tous ces défauts de l'image, acceptables voire souhaités dans un film ne le sont pas dans le cadre d'une reconstruction ou de rendu à base de vidéos.

Dans la mesure du possible, il est préférable d'ajuster les sources de lumière de la scène de façon à obtenir une scène éclairée de façon homogène. Il est aussi prudent d'éviter de placer les sources de lumière face aux caméras. Dans le cas contraire, il apparaît parfois sur les images des taches lumineuses (*flare*) illustrée sur les figure 9 et figure 10. Par ailleurs, certaines méthodes de rendu à base d'images ne gèrent que les objets diffus (i.e. ne gèrent pas les reflets), il est alors important de bien positionner les sources lumineuses et si possible d'éclairer dans toutes les directions. Le cas échéant, l'utilisateur peut se servir un diffuseur, sorte de plaque opaque, permettant de rendre la lumière plus diffuse.

Figure 9 : *flare*.

Image de François de Sorbier

Figure 10 : *lens flare*.

1.5.4 Calibrage de la couleur et balance des blancs

Certaines méthodes de rendu à base d'images ou de vidéos, notamment les méthodes colorimétriques, sont très sensibles à la bonne restitution des couleurs sur chacune des caméras. La plupart des modèles commerciaux disposent de modes de balance des blancs préprogrammés et surtout d'un mode de paramétrage manuel. Celui-ci consiste simplement à filmer un objet blanc (une feuille blanche par exemple) dans les conditions d'éclairage de la scène. Il arrive que les caméras de vision par ordinateur ne soient pas pourvues de ce genre de fonctionnalités dans quel cas la balance des blancs revient à la charge de l'utilisateur.

Le calibrage des couleurs est souvent effectué à l'aide de palettes de couleurs disposées devant les caméras. La plus utilisée est sans doute le *Color Checker* de Greta Macbeth souvent appelé la palette Macbeth (figure 11). Il s'agit d'une palette de 24 couleurs étalonnées disponible dans le commerce. Il est fortement déconseillé d'effectuer le calibrage des couleurs avec une palette imprimée sur une imprimante grand public au risque d'utiliser des couleurs altérées. Certaines méthodes utilisent ce test pour calibrer de façon parfaite la couleur de chacune des caméras, en utilisant les données de référence relatives à chaque couleur de la palette. D'autres méthodes se contentent d'harmoniser les couleurs entre chaque caméra sans nécessairement faire en sorte que les couleurs finales correspondent parfaitement aux couleurs de référence. C'est le cas dans la méthode proposée par Joshi et al. [JW05] utilisée pour calibrer le mur de caméras de Stanford (*Stanford Camera Array*) ou bien dans la méthode de Nanda, et Cutler [NC01] utilisant eux aussi un très grand nombre de caméras.

Figure 11 : *Color Checker* de Macbeth.

Dans la même volonté d'harmoniser les couleurs des caméras entres elles, Marcus Magnor [M05] propose d'associer à chaque caméra i une matrice (3×4) M_i de correction des couleurs. Cette méthode utilise elle aussi les 24 couleurs du *Color Checker* de Macbeth. A chacune des 24 couleurs du *Color Checker* est associée la couleur moyenne (R_j G_j, B_j) de cette couleur vue par l'ensemble des caméras. C'est cet ensemble de couleurs R_j qui servira de référence. Soit (r_{ij} ,g_{ij}, b_{ij}) la $j^{ème}$ couleur du *Color Checker* vue par la $i^{ème}$ camera.

La matrice M_i doit, pour toutes les couleurs j du *Color Checker*, satisfaire au mieux l'équation :

$$\begin{pmatrix} R_j \\ G_j \\ B_j \end{pmatrix} = M_i \begin{pmatrix} r_{ij} \\ g_{ij} \\ b_{ij} \\ 1 \end{pmatrix}$$

Ce système se résout en utilisant le même genre d'équations que celles utilisées pour calibrer une caméra projective.

1.5.5 Gestion d'un réseau, compression du flux vidéo

Les dispositifs d'acquisition composés de plusieurs caméras utilisent souvent des PC intermédiaires chargés non seulement d'une correction de la distorsion radiale mais aussi parfois de prétraitements sur les images. Un grand soin doit être porté à la synchronisation des caméras sans quoi des artéfacts peuvent apparaître lors de la combinaison des images entres elles. Il existe toutefois des méthodes d'interpolation temporelle comme la méthode de Wang et Yang [WY05] permettant de simuler une synchronisation.

Pour alléger le flux vidéo à travers le réseau, certaines méthodes comme celle de Yang et al. [YEB02] arrivent à sélectionner dans une image les parties qui vont être utilisées et demandent au PC intermédiaire de ne transmettre que cette partie de l'image. Pelletier et al. [PSC05] proposent une méthode permettant d'alléger le flux vidéo sur le réseau en n'envoyant les images en pleine résolution qu'une fois sur 10. Les autres images sont transmises avec une résolution inférieure et sont reconverties en images pleine résolution à l'aide des images clés. Cette méthode allège énormément le réseau tout en assurant un minimum de perte.

Il est parfois nécessaire de stocker de grandes quantités de vidéos, représentant souvent la même scène vue d'un point de vue légèrement différent. La similarité des vidéos entres elles fait qu'il existe des méthodes de compression très efficaces, notamment celle proposée par Levoy et Hanrahan [LH96] ou bien celle introduite par Chen et al. [CBC02]. Les méthodes de compression doivent impérativement permettre de décompresser les vidéos très rapidement, sachant qu'il est souvent nécessaire de décompresser plusieurs flux simultanément.

Pour trouver des informations complémentaires, le lecteur peut se référer au papier de Matusik et al. [MP04] ainsi qu'à celui de Smolic et McCutchen [SM04]. Il est également intéressant d'examiner le protocole DCCP (*Data Congestion Control Protocol*) qui correspond au prochain protocole de streaming.

1.6 Bilan

Les parties précédentes décrivent les outils principaux de vision par ordinateur, notamment la modélisation d'une caméra et les informations qu'on peut en extraire ou encore la relation épipolaire entre deux images. La grande majorité des méthodes de rendu à base d'images, de rendu à base de vidéos ou de reconstruction utilise au moins l'un de ces deux outils. Il en va

de même pour le descriptif de la mise en place d'un dispositif d'acquisition vidéo qui est commun à quasiment toutes ces méthodes.

Ces éclaircissements et précisions permettent de poursuivre sur la partie de l'état de l'art concernant les méthodes de rendu à base d'images. Pour plus de précisions sur la géométrie projective ou sur la vision par ordinateur, le lecteur peut se référer au livre « *Multiple View Geometry in Computer Vision* » de Hartley et Zisserman [HZ04] qui fait référence dans le domaine.

2 Rendu à base d'images : état de l'art

Les méthodes de rendu en images de synthèse ne cessent de s'améliorer, tant en ce qui concerne les images photo-réalistes que les images générées en temps réel. Ces méthodes évoluent toujours avec le soucis d'augmenter d'avantage la sensation d'immersion et de fluidité durant la navigation.

Une étape supplémentaire dans le perfectionnement de ces techniques et dans l'augmentation de l'immersion consiste à utiliser directement des photos illustrant une scène réelle et à les introduire dans l'image synthétisée. C'est dans cette optique que sont apparues les textures 2d appliquées sur des maillages 3d représentant une scène réelle. Pour aller plus loin dans cette direction, l'idée d'utiliser uniquement des photos pour générer de nouvelles images plus réalistes a été posée. Cette nouvelle discipline comprend plusieurs branches, avec des approches radicalement différentes, mais ayant toutes le même objectif.

Certaines méthodes s'orientent vers de la reconstruction géométrique pour réaliser un maillage modélisant une scène réelle puis texturent ce maillage avec les mêmes images ayant servies à la reconstruction. Cette approche, nommée photogrammétrie, utilise très largement les outils de vision par ordinateur et génèrent des modèles de scènes réelles parfois saisissants de ressemblance. Un tel niveau de détail n'est accessible qu'après une grande quantité de calcul et les techniques réalisant de telles reconstructions sont particulièrement gourmandes en ressources.

Une autre approche du problème consiste à ne pas se baser sur la géométrie de la scène mais juste sur la répartition des images entres elles. Il devient alors possible de générer de nouvelles images intermédiaires de la même scène. Cette démarche, nommée « rendu à base d'images », est déclinée par toute une variété de méthodes, plus ou moins rapides, effectuant un rendu plus ou moins réaliste. Par abus de langage, le terme « rendu à base d'image » est souvent employé pour désigner l'ensemble des méthodes permettant de générer une nouvelle image d'une scène ou d'un objet à partir de photographies.

Il existe enfin une multitude de méthodes hybrides empruntant à la photogrammétrie et au rendu à base d'images leurs avantages respectifs.

Au fil de l'évolution de ces méthodes et du progrès technologique, notamment sur les performances des cartes graphiques, certaines méthodes ont pu s'adapter au traitement de vidéos représentant une scène dynamique plutôt qu'au traitement de scènes statiques. Il est possible de trier ces méthodes en deux catégories. Celles qui traitent préalablement les vidéos et qui permettent de générer en temps réel des nouvelles images de la scène. Il s'agit alors de play-back ou de différé. La seconde catégorie concerne les méthodes capables de traiter les vidéos et de générer de nouvelles vues à la volée. Il est alors question de rendu en direct ou on-line en anglais.

Quelle que soit la famille à laquelle une méthode appartient, elle possède nécessairement des caractéristiques communes aux autres méthodes ce qui facilite leur comparaison. L'un des premiers critères de comparaison est le nombre d'images ou de caméras nécessaires pour le bon fonctionement de la méthode. Ce nombre de caméras a un impact direct sur la qualité du rendu, la mobilité du système et les temps de calculs. La qualité du rendu et les ressources nécessaires pour y parvenir sont d'ailleurs deux autres caractéristiques concurrentes.

Certaines méthodes admettent des contraintes particulières concernant la position des caméras ou sur le type de matériaux constituant la scène qui génèrent éventuellement des reflets dépendant du point de vue. Enfin, un critère de regroupement évident est l'appartenance à un courant comme la photogrammétrie ou le rendu à base d'images ou bien encore l'aptitude à traiter des vidéos en direct ou en différé.

Nous allons organiser notre état de l'art autour de la comparaison du temps d'exécution des différentes méthodes car parmi tous les critères présentés, nous estimons qu'il est le plus significatif. Nous commencerons donc par les méthodes les plus robustes nécessitant de longs calculs. Ensuite, nous présenterons les méthodes ne fonctionnant qu'après un long prétraitement, parfois manuel, mais permettant ensuite de générer des nouvelles vues de la scène en temps réel. Les deux dernières parties traitent les méthodes supportant la vidéo. Là encore, nous commencerons par les méthodes nécessitant un prétraitement sur la vidéo avant de pouvoir interpoler les images en temps réel. Dans ce cas, la navigation est donc bien en temps-réel mais sur une vidéo en différé (play-back). Enfin, nous finirons par les méthodes capables de traiter les vidéos à la volée et de générer de nouvelles vues en direct (on-line). Nous constaterons une tendance générale qui montre que les méthodes les plus rapides font l'impasse sur la qualité visuelle.

Notons enfin que la notion de temps réel est liée à un mode de percetion. Dans notre cas, il s'agit de la vue ce qui suppose qu'une application peut être considérée comme temps réel si elle génère au moins une vingtaine d'images par seconde. Si le framerate de cette application se situe plutôt autours des dix images par secondes, on parle alors de rendu en temps interactif. Pour des applications sonnores ou tactiles, il faut atteindre des fréquences bien plus élevées pour pouvoir parler de temps réel.

2.1 Rendu à base d'image réaliste non temps réel

Un moyen efficace d'effectuer un rendu réaliste à partir d'images consiste à reconstruire la scène réelle avec des outils de vision par ordinateur. Les erreurs de reconstruction peuvent provenir d'erreurs de calibrage, de bruit sur les images et de mauvaises correspondances entre un même point sur plusieurs images. Certains points de la scène ne sont vus que par une seule caméra ce qui pose un problème pour établir leur profondeur et donc pour les localiser. Afin de remédier à ce genre de problèmes, il faut disposer d'un maximum d'images ce qui engendre des calculs supplémentaires. Dans le cas d'un nombre restreint de caméras, il est possible de procéder à des optimisations imposant des critères de continuité sur les objets ou sur leur couleur.

Dans cette première partie, nous commencerons par une introduction des outils permettant de créer une carte de disparité, équivalente à une carte de profondeur, constituant le premier pas vers une reconstruction. Ces outils sont toutefois utilisés dans toutes sortes de méthodes, y compris les plus rapides. Cependant les outils les plus sophistiqués ne peuvent fonctionner en temps réel. Notons aussi que dans cette partie, nous spécifions uniquement les modes de calcul de cette carte de disparité sans préciser le mode de rendu.

Nous aborderons ensuite une série de méthodes nécessitant une grande quantité de calculs, y compris pour la phase de rendu, ne leur permettant pas de générer des images en temps réel. C'est le cas des méthodes à base de voxels ou des *Image-Based Priors*. Nous présenterons aussi des méthodes de morphing complexes gérant correctement les effets de changement de

perspective. Nous finirons par une technique de reconstruction utilisant des caméras non calibrées associée à des techniques d'optimisation géométrique pour améliorer le rendu.

2.1.1 Premier pas vers le rendu à base d'images : les cartes de disparités

Etant données deux caméras et un point *M* de l'espace, la disparité entre les projetés de *M* sur les deux caméras est défini comme la distance qui les sépare dans le repère des images. Une carte de disparité établit donc des correspondances de pixels entre deux images d'une même scène. Ces correspondances sont le premier pas vers une reconstruction de la scène, c'est pourquoi tant de recherches ont été faites sur cette étape tout à fait critique.

Les méthodes de correspondance exploitant les informations sur le voisinage du pixel traité sont dites locales. Inversement, les méthodes utilisant des données sur toute l'image sont dites globales. Les méthodes locales peuvent être très efficaces mais sont assez sensibles aux occlusions et sont peu efficaces sur des régions de couleur uniforme. Quant aux méthodes globales, elles sont souvent assez gourmandes en calculs. Les paragraphes suivants traitent de ces deux méthodes ainsi que de leurs optimisations.

2.1.1.1 Correspondances de pixels et calcul de scores

Pour établir une correspondance entre deux pixels provenant de deux images d'une même scène, la plupart des méthodes utilisent des scores afin de déterminer pour un pixel de la première image le pixel de la seconde image qui lui « ressemble » le plus. Les scores permettent de trier les pixels candidats dans l'ordre de préférence ce qui rend possible l'application de stratégies capables de rejeter le pixel le moins ressemblant. Les scores permettent aussi d'évaluer la robustesse d'une correspondance de pixels.

Il existe plusieurs méthodes de calcul de score, la plus simple est la *square intensity difference* notée SD qui calcule le carré de la différence entre deux intensités lumineuses (couleurs en niveaux de gris par exemple). Pour deux pixels d'intensité p^A et p^B, la SD se note :

$$\mathrm{SD}(p^A, p^B) = (p^A - p^B)^2$$

La SD peut s'appliquer pour toutes les composantes d'un pixel en couleur :

$$\mathrm{SD}(p^A, p^B) = \sum_{c \in \{R,V,B\}} \left(p_c^A - p_c^B \right)^2$$

Il existe ces deux variantes de la SD pour la AD (*absolute intensity difference*) qui calcule la valeur absolue des différences d'intensité lumineuse entre deux pixels :

$$\mathrm{AD}(p^A, p^B) = |p^A - p^B|$$

Pour plus de robustesse, la SD et l'AD s'appliquent sur chaque pixel contenu dans une fenêtre centrée sur le pixel examiné, il s'agit de la SSD (*sum of square intensity difference*) introduite par Barnea et Silverman [BS72] et de la SAD (*sum of absolute intensity difference*). Celle-ci est utilisée dans les premières applications de calcul de carte de disparité temps-réel notamment par Kanade [K94] qui recherche le correspondant d'un pixel en appliquant la SAD sur la droite épipolaire de l'autre image. La SAD et la SSD produisent

cependant de nombreux minimums locaux et par conséquent les corrélations ne sont pas très robustes. La SSSD (*sum of SSD*) permet de gérer la SSD entre trois images ou plus.

Pour plus de robustesse, Kanade et Okutomi [KO94] utilisent des fenêtres adaptatives dont la taille varie en fonction du pixel traité. Scharstein et Szeliski [SS98] utilisent un procédé de grossissement de la fenêtre. Ce procédé est itératif et non linéaire. Les conditions d'arrêt s'adaptent aux caractéristiques de chaque fenêtre.

La corrélation croisée normalisée est une méthode de corrélation robuste aux variations de dynamique d'une image sur l'autre. Soient I^A et I^B les deux fenêtres carrées de coté N issues des images A et B. La comparaison de ces deux sous-images s'effectue à l'aide des opérateurs suivants :

- Le calcul des moyennes : $\bar{I} = \frac{1}{N^2}\sum_{u=0}^{N}\sum_{v=0}^{N} I(u,v)$
- Le « produit scalaire » : $\left\langle I^A, I^B \right\rangle = \frac{1}{N^2}\sum_{u=0}^{N}\sum_{v=0}^{N}\left(I^A(u,v)-\bar{I}^A\right)\left(I^B(u,v)-\bar{I}^B\right)$
- La norme : $\left|I\right| = \sqrt{\left\langle I, I \right\rangle}$

La corrélation croisée normalisée renvoie un score C^{AB} défini de la façon suivante :

$$C^{AB} = \frac{\left\langle I^A, I^B \right\rangle}{\left|I^A\right|\left|I^B\right|}$$

C^{AB} est compris entre –1 et 1. Plus C^{AB} est proche de 1, plus les images I^A et I^B se ressemblent. Un score de 0 indique que les pixels dont la couleur est proche de la couleur moyenne de leur image ne sont pas distribués de la même manière sur les deux images. Un score de –1 signifie que I^A est le négatif de I^B.

Crouzil et al. [CMC96] effectuent leurs corrélations à l'aide de dérivées et notamment du filtre de Shen-Castan. L'*Orientation Code Matching* (OCM) de Ullah et al. [UKI01] utilise lui aussi des dérivées sur les images traitées pour les calculs de corrélation.

Enfin, d'autres méthodes telles que celle de Bhat et Nayar [BN98] ou bien celle de Zabih et Woodfill [ZW94] utilisent des méthodes de mesures ordinales basées sur un classement des niveaux de gris des pixels de la fenêtre de corrélation et utilisent des transformations non paramétriques.

Globalement, plus les méthodes sont robustes face aux occlusions et au bruit, plus elles sont coûteuses en temps de calcul. Ainsi les méthodes du type SSD ou SAD pourront être utilisées dans un dispositif à rendu en temps réel. Les méthodes de la famille de la corrélation croisée normalisée sont des méthodes intermédiaires en performance et en robustesse. Elles sont largement utilisées pour des méthodes d'évaluation de relations épipolaires en générant un grand nombre de corrélations entre deux images, les mauvais appariements sont alors supprimés lors du calcul de la matrice fondamentale (Hartley et Zisserman [HZ04]). Enfin les méthodes plus robustes sont dédiées aux calculs précis de cartes de disparité. Elles présentent de meilleurs résultats que les méthodes plus rapides mais sont difficilement adaptables à un rendu temps réel. Si les performances en rapidité ne sont

pas primordiales, il est intéressant de les coupler avec des méthodes d'optimisation globale décrites dans la section suivante. Chambon et Crouzil [CC03] proposent un comparatif (en français) de toutes ces méthodes en détaillant à quel point certaines méthodes sont sensibles à la présence de données aberrantes ou au contraire robustes aux occultations.

2.1.1.2 Optimisation globale

Les méthodes de corrélation proposées dans le paragraphe précédent attribuent à chaque correspondance potentielle un score. Une stratégie locale consiste à choisir la correspondance de score optimal. Cette méthode, appelée *winner-take-all* (WTA), ne prend pas en compte ni la pertinence du score, ni le fait que d'autres correspondances potentielles puissent avoir un score très proche du meilleur score et ainsi être elles aussi de bons candidats.

Contrairement aux stratégies locales, les stratégies globales prennent en compte tous ces paramètres. Elles sont souvent formulées comme des problèmes de minimisation d'énergie utilisant un terme local et un terme global. Pour une disparité d, on peut les noter de la façon suivante :

$$E(d)=E_{data}(d)+\lambda E_{smooth}(d)$$

où $E_{data}(d)$ correspond au score local de d, $E_{smooth}(d)$ correspond à la variation de d par rapport à son entourage et λ à une constante. Geman et Geman [GG84] traitent ces équations à l'aide de lois de Baye. Geiger et Girosi [GG91] utilisent une méthode dite de *mean-field-annealing* afin de trouver un minimum de cette fonction d'énergie.

Quelle que soit la stratégie adoptée, les calculs de correspondances mettent en évidence trois cas :

- Un pixel **p** d'une image correspond à un pixel **p**′ de l'autre image et réciproquement.
- Plusieurs pixels d'une image sont associés au même pixel de l'autre image.
- Un pixel ne trouve pas de correspondant dans l'autre image.

Ces caractéristiques révèlent les effets d'obturation d'une image à l'autre. Certaines stratégies traitent ces obturations comme les mesures de corrélation partielle de Lan et Mohr [LM97] ou bien les R-estimateurs de Zhang [Z97]. Drouin et al. [DTR05] détectent les occlusions à partir de plusieurs images de la scène. Pour cela ils déterminent pour chaque point de la scène l'ensemble des caméras à partir desquelles il est visible. Dans le cas d'occlusions, ils déterminent les zones responsables de cette occlusion.

Pour augmenter la précision de la triangulation à moindre coût, il est recommandé d'effectuer un raffinement subpixel comme par exemple une interpolation subpixel à base de splines décrite par Tian et Huhns [TH86].

Enfin, un filtre médian ou une interpolation de surface (*surface fitting*) appliqué sur la carte de disparité peut éliminer un certain nombre d'erreurs de disparité. Comme le montrent Birchfield et Tomasi [BT98] qui utilisent une interpolation linéaire autour du pixel traité, cette optimisation ne rajoute pas beaucoup d'opérations supplémentaires à la méthode de calcul de disparité utilisée.

Pour plus de détails, Scharstein et Szeliski [SS02] proposent des tests de comparaison d'un grand nombre d'algorithmes de mise en correspondance ainsi qu'un programme de test utilisable par tous. Brown et al. [BBH03] proposent un comparatif détaillé des méthodes de corrélations et de stéréo traitant notamment de la gestion des occlusions ainsi que des méthodes de calcul de carte de disparité temps-réel.

Dans le cas où les caméras ne sont pas calibrées, il n'est pas possible d'utiliser la contrainte épipolaire lors du calcul des correspondances. La relation épipolaire peut toutefois être retrouvée à l'aide de l'algorithme RANSAC [FB81] (*RANdom SAmple Concensus*) introduit par Fischler et Bolles. Des points d'intérêt caractéristiques de la scène sont extraits des images à l'aide de détecteur de coins tels que le détecteurs de Harris [HS88], puis ces points sont grossièrement corrélés d'une image à l'autre. Ensuite, 8 correspondances sont choisies aléatoirement parmi les correspondances disponibles. Une première relation épipolaire est alors établie. La pertinence de cette relation est testée en calculant le nombre de correspondances en accord avec cette relation épipolaire. Un score est établi en dénombrant le nombre de pixels dont le correspondant se trouve à une distance inférieure à une distance limite par rapport à sa position prédite par la relation épipolaire proposée. Ces pixels sont appelés les *inliers*. Ensuite, 8 autres correspondances sont choisies aléatoirement pour tester une nouvelle relation épipolaire. Cette étape est répétée un grand nombre de fois et la relation épipolaire pourvue du plus grand nombre d'*inliers* est retenue. Dans le cas d'une égalité en nombre d'*inliers*, un score départage les relations concurrentes. Ce score est calculé en sommant pour chaque pixel la distance quadratique qui le sépare de sa position prédite. Une nouvelle relation épipolaire est alors calculée à partir de l'ensemble des correspondances valides pour la relation épipolaire victorieuse. Cette relation épipolaire peut éventuellement servir de contrainte pour une nouvelle application de l'algorithme RANSAC afin de trouver un plus grand nombre de correspondances valides. Le nombre d'itérations nécessaires à la méthode RANSAC pour obtenir un résultat « intéressant » peut être estimé par des outils statistiques. Hartley et Zisserman [HZ04] proposent une évaluation du nombre d'itérations N nécessaires pour trouver avec une probabilité p un échantillon de 8 correspondances valides à partir d'une estimation w de la proportion d'*inliers*.

$$N = \frac{\log(1-p)}{\log(1-w^s)}$$

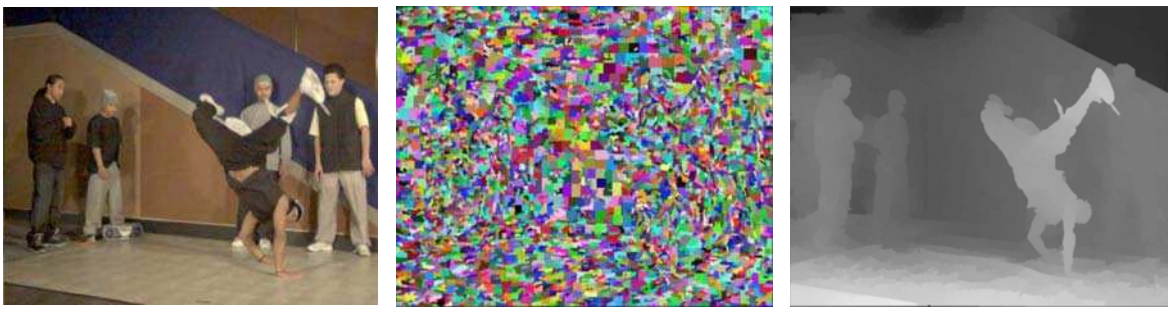
où s=8 car 8 correspondances sont utilisées pour le calcul de la relation épipolaire. Lacey et al. [LPT00] proposent une évaluation des performances de RANSAC sur le calibrage de cameras. La méthode MLESAC de Torr et Zisserman [TZ00] est une généralisation de RANSAC qui gère ses *inliers* non plus de façon binaire (*inlier* ou *outlier*) mais de façon statistique. MLESAC choisit donc comme solution optimale la relation qui maximise non plus le nombre d'*inliers* mais celle qui maximise un score global.

Les méthodes décrites utilisent les informations extraites des images dans leur globalité. Ces méthodes présentent donc de très bon résultats mais compte tenu du nombre d'opérations nécessaires à leur fonctionnement, il est actuellement impensable de les adapter au rendu temps réel.

2.1.1.3 Méthodes de segmentation de couleurs

Les méthodes de segmentation de couleurs consistent à découper les images traitées en petites zones ayant des caractéristiques colorimétriques communes [CM97]. Les zones sont

alors appareillées sur deux images afin de générer une carte de disparité. Tao et al. [TSK01] optimisent l'appareillement de ces zones en supposant que chaque zone est plane et en trouvant pour chacune d'elle une orientation dans l'espace. Zitnick et al. [ZBU04] limitent la taille de ces zones à un carré de 40 pixels de coté. Les zones de moins de 100 pixels sont fusionnées avec la zone voisine dont la couleur est la plus proche. Les résultats obtenus sont relativement précis (figure 12). Cette méthode est expliquée avec plus de détails dans le chapitre 2.3.2.



Avec l'aimable autorisation de Larry Zitnick [ZBU04]

Figure 12 : segmentation de couleurs et application au calcul de cartes de disparité.

2.1.1.4 Flux optique

Le flux optique est un champ de vecteurs qui décrit le mouvement apparent d'objets d'une image à une autre (figure 13). Une fois établi pour quelques points d'un objet rigide, il permet de retrouver plus facilement des correspondances entres images sur ce même objet. S'il s'agit d'une scène statique, il n'y a qu'un seul objet rigide dans la scène. Le flux optique permet aussi de retrouver la forme d'un objet dans le cas d'un champ de vecteur assez dense.

Parmi les premières méthodes de calcul de flux optique figure celle de Horn et Schunk [HS81] dont les possibilités seront détaillées par Barron et al. dans [BF92]. Black et Anandan [BA93] proposent une méthode robuste d'estimation du flux optique et proposent d'ailleurs le code source de leur méthode sur leur site internet. Zhang et Kambhamettu [ZK01] calculent directement le champ de vecteurs 3d à partir de plusieurs séquences vidéo prises de points de vue différents. Vedula et al. [VBR05] établissent le champ de vecteurs 3d à partir de chaque champ de vecteurs 2d des images de départ.

Figure 13 : flux optique.
Le flux optique est représenté par les segments noirs symbolisant le mouvement d'un pixel pour passer d'une image à l'autre.

Contrairement aux méthodes précédentes, les méthodes de flux optique tentent de gérer les appariements entre deux images en respectant une cohérence géométrique.

2.1.2 Voxels et Poxels

Seitz et Dyer [SD97] présentent une méthode basée sur l'utilisation de voxels (de l'anglais *volume element*). Cette méthode permet d'une part d'effectuer un appariement de pixels robuste entre plusieurs images d'une même scène et d'autre part de générer une reconstruction de cette scène par voxels à partir d'un ensemble d'images calibrées.

La reconstruction par voxels consiste à subdiviser une scène à reconstruire par un ensemble de voxels, généralement de forme cubique. Durant le processus de reconstruction, chacun de ces voxels est soit supprimé, soit coloré suivant sa cohérence avec les images de la scène. En général, cette étape ne s'effectue pas par un calcul de correspondances de pixels entre images de départ et doit donc faire face à un problème d'appareillement entre pixels.

Pour deux couples de pixels de même couleur (figure 14), deux possibilités sont envisageables :

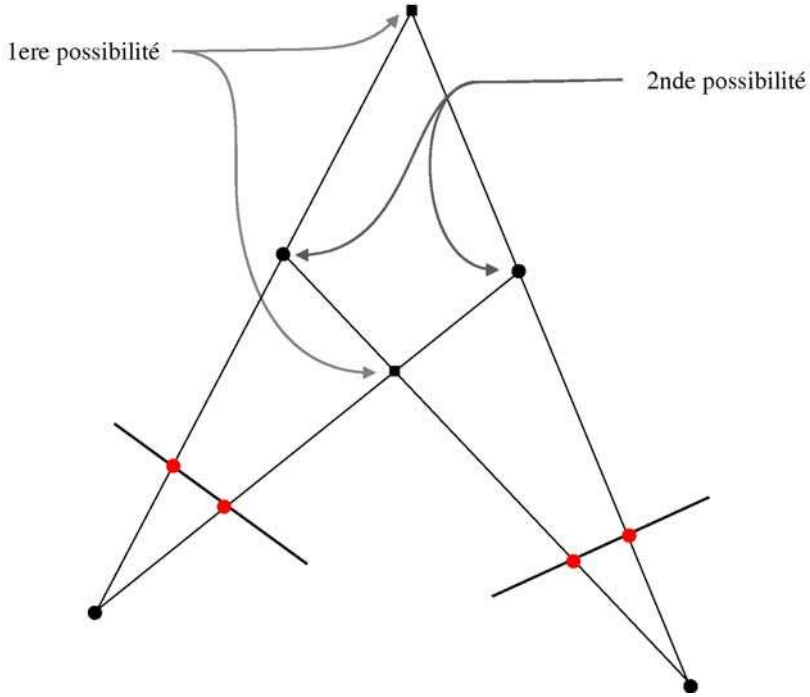


Figure 14 : voxels : appariement de deux couples de pixels de même couleur.

Ce problème se complexifie avec un couple de pixels supplémentaire. Les occlusions éventuelles multiplient les possibilités envisageables. Il est possible d'extraire trois familles de configurations dans cette situation. La première est juste une extension du cas précédent, il en existe donc 4 variations. Elle est illustrée par la figure 15 sur laquelle deux possibilités par couleurs sont dessinées : une avec des ronds pleins et l'autre avec des ronds vides.

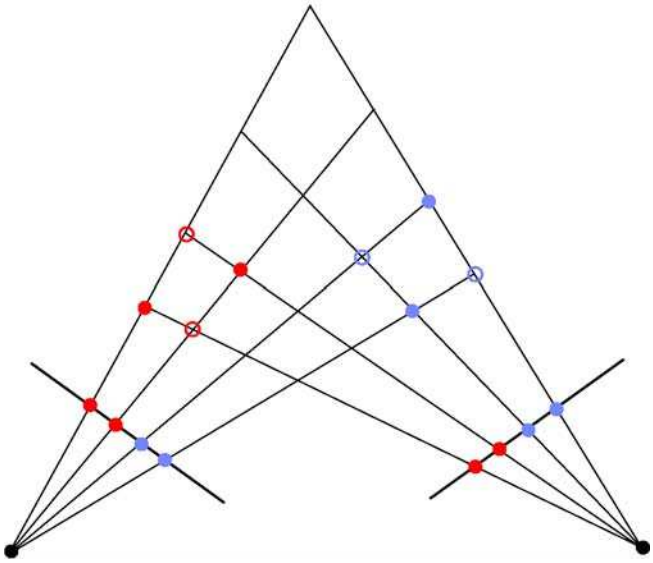


Figure 15 : voxels : appariement de quatre couples de pixels sans occlusion.

La figure 16 illustre une occlusion d'un voxel d'une couleur par un voxel d'une autre couleur. Les deux couples de voxels interfèrent entre eux. Cette configuration a bien plus de variations que la précédente.

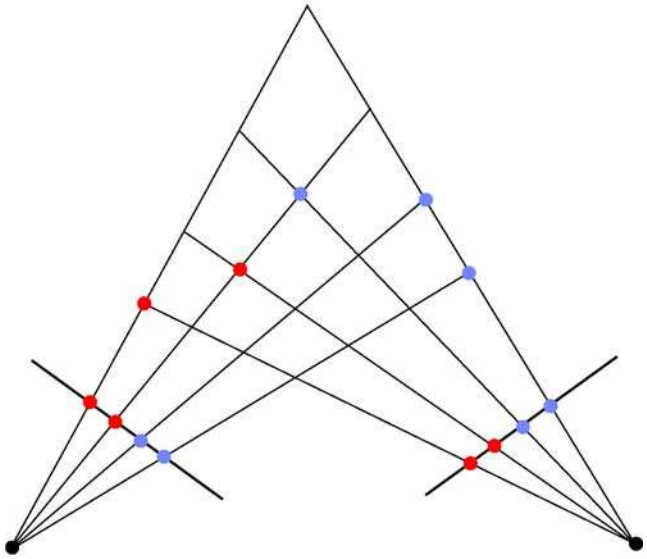


Figure 16 : voxels : appariement de quatre couples de pixels avec occlusions entre pixels de couleur différente.

Enfin, la figure 17 illustre des occlusions entre pixels de même couleur. Là encore il en existe de nombreuses variations.

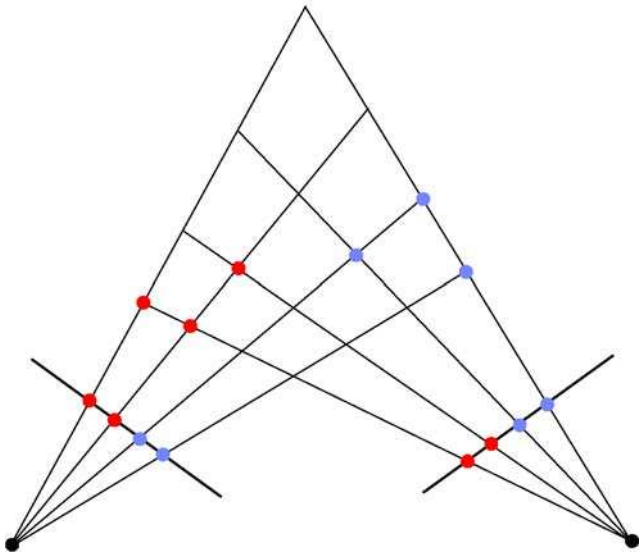


Figure 17 : voxels : appariement de quatre couples de pixels avec occlusions entre pixels de même couleur.

Le traitement des voxels doit aussi pouvoir traiter une combinaison de ces cas d'occlusion.

Afin de pouvoir gérer tous ces cas en une seule passe, Seitz et Dyer [SD97] proposent de regrouper les voxels de la scène en couches (*layers*) de telle sorte qu'un voxel *P* occlue un voxel *Q* implique que le numéro de *layer* de *P* soit inférieur à celui de *Q*. Il suffit ainsi de traiter les voxels par ordre de *layer* pour gérer les occlusions en une seule passe. Un exemple de cette répartition de *layers* est illustré sur la figure 18.

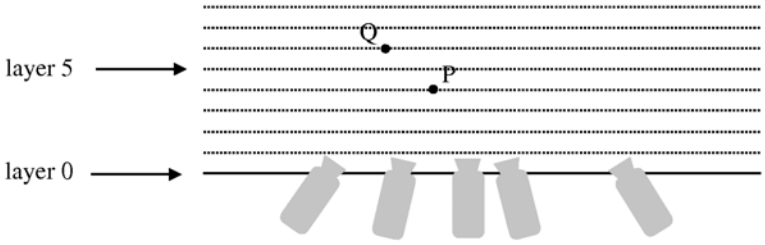


Figure 18 : répartition des voxels par couches.

Cette représentation en *layer* permet aussi de gérer les panoramiques comme le montre la figure 19, cependant certaines configurations sont interdites comme par exemple un ensemble de caméras disposées autour d'une scène. Il s'agit d'une forte contrainte sur la disposition des caméras.

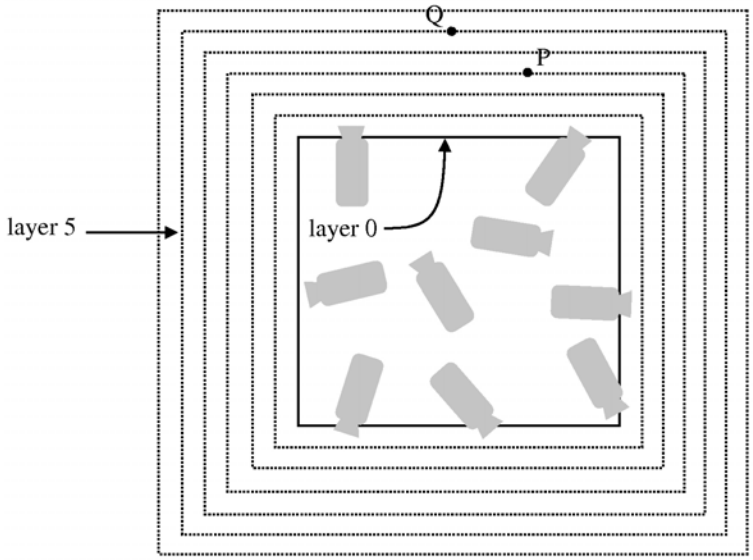


Figure 19 : répartition des voxels par couches,
application aux vue panoramiques.

L'utilisation de voxels provoque parfois un problème d'occlusion partielle entre plusieurs voxels pas tout à fait alignés. Westover [W90] règle ce problème à l'aide de traces (*footprint* dans son article) qui correspondent à l'ensemble des pixels d'une image sur lesquels se projette un voxel. Si un voxel est défini comme valide, sa trace est étiquetée « déjà lue » et ne sera pas prise en compte pour les voxels suivants.

L'algorithme de reconstruction s'effectue en une passe sur les voxels, après les avoir triés par *layers*. Il s'écrit de la façon suivante :

Pour chaque layer *L* par ordre de numéro croissant

- Pour chaque voxel *V*

  - calculer l'ensemble des traces de *V* sur chaque image parmi les pixels non lus

  - évaluer la corrélation entre les traces de chaque image

  - en cas de « bonne corrélation », empiler les pixels de la trace de V

- étiqueter les pixels empilés comme « déjà lus »

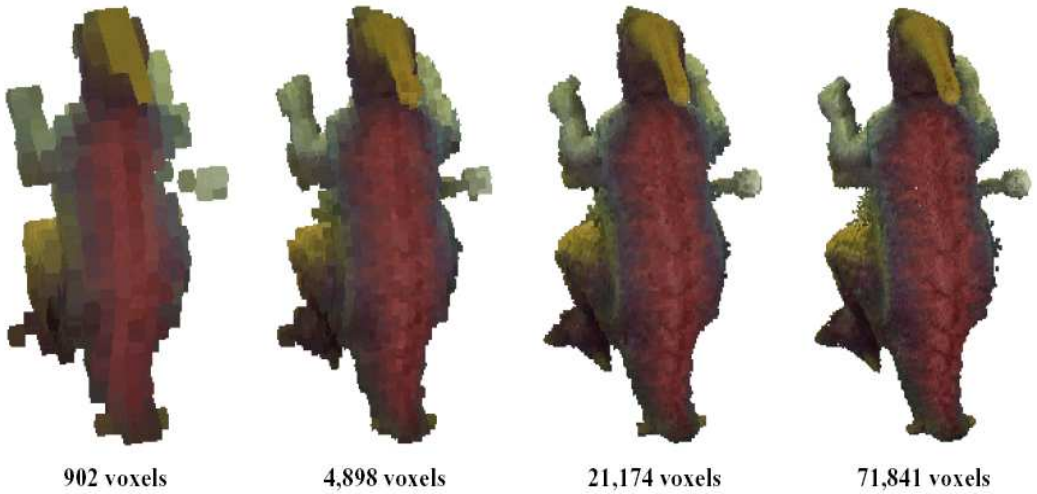
Il faut noter que les voxels retenus par cette méthode sont situés sur la surface de l'objet reconstruit ce qui permet d'éviter une explosion du nombre de voxels à afficher. Il faut aussi remarquer que cette méthode ne fonctionne que sur des scènes dépourvues de réfléxions spéculaires.

Le rendu de voxels est une opération très coûteuse en calculs dès lors que les voxels sont affichés sous forme de cubes. Même si les voxels ne sont distribués que sur les surfaces des objets, le nombre de voxels devient très vite élevé si l'on désire une bonne qualité visuelle.

Seitz et Dyer [SD97] obtiennent des scènes d'environ 70 000 voxels pour une discrétisation de la scène de 166x199x233, ce qui correspond à 800 000 polygones à afficher si chaque voxel est affiché sous forme de cube (pour 12 polygones par cube). Même avec des méthodes d'optimisation, il est très difficile d'atteindre un rendu en temps réel pour une scène complexe et réaliste. Cependant, pour une scène discrétisée grossièrement, la reconstruction et l'affichage peuvent être effectués en temps réel, dès lors que la disposition des *layers* ne change pas. Il s'agit dans ce cas d'une application d'exploration plutôt que de rendu réaliste. Ceci apparaît clairement dans la figure 21 représentant les rendus obtenus avec différents niveaux de discrétisation. La figure 20 montre une des images originales ayant servi à ces reconstructions.

Avec l'aimable autorisation de Steven Seitz [SD97]

Figure 20 : voxels : image originale.

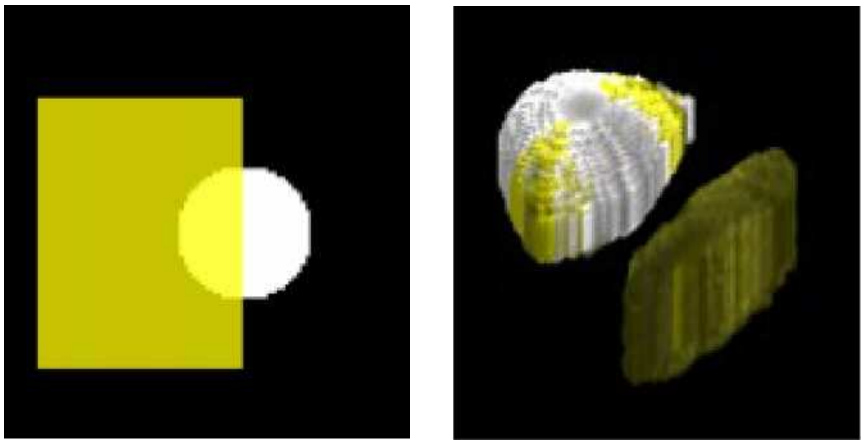


Avec l'aimable autorisation de Steven Seitz [SD97]

Figure 21 : voxels : reconstruction avec plusieurs niveaux de discrétisation.

De Bonet et Viola introduisent le concept de Poxel (*probabilistic voxel*) [DV99] capable de traiter les voxels transparents. La scène est discrétisée en voxels $V(x,y,z)$ de couleur $C(x,y,z)$ et les pixels d'une image P_k (k =1…nombre d'images) sont représentés sous forme de

vecteurs I_k. Soit R un rayon passant par le centre de projection d'une caméra et par un pixel de l'image associée. La couleur de ce pixel est un mélange pondéré des couleurs des voxels traversés par R. On attribue à un voxel un coefficient de « responsabilité » pour chaque pixel d'une image P_k. Cette relation peut s'écrire matriciellement sous la forme $I_k = R_kC$, où R_k est la matrice de responsabilité de l'image P_k. Si l'on représente toutes les images sous la forme d'un unique vecteur I et l'ensemble des responsabilités sous la forme d'une seule grosse matrice R, la relation précédente devient $I=RC$. La matrice R est une matrice creuse ce qui permet d'utiliser des méthodes d'optimisation pour la manipuler sans quoi il serait très difficile de s'en servir. Cette matrice obéit à une contrainte sur les voxels qui exprime le fait que les voxels subissant l'occlusion d'un voxel non transparent ont une responsabilité nulle. Grâce à cette contrainte, le système évolue récursivement jusqu'à ce qu'une solution stable pour R et C soit obtenue. La figure 22 montre la reconstruction d'une scène générée sous Pov-ray composée d'une sphère blanche placée derrière une plaque jaune semi-transparente. L'image de gauche montre une des 36 images de départ représentant cette scène. L'image de droite montre la reconstruction obtenue en discrétisant la scène en 128x128x128 voxels.



Avec l'aimable autorisation de Jeremy S. De Bonet [DV99]

Figure 22 : poxels, reconstruction gérant les éléments semi-transparents.

2.1.3 Image-based Priors

Certaines méthodes utilisent des outils statistiques pour traiter les données en entrée. C'est le cas des *image-based priors* de Fitzgibon et al. [FWZ03] basés sur des relations épipolaires entre images d'une même scène. Cette méthode permet de générer une nouvelle vue d'une scène à partir d'un ensemble d'une trentaine d'images calibrées prises de points de vue relativement proches les uns des autres.

Le principe général est assez simple : pour commencer, l'utilisateur définit la matrice de projection de la caméra correspondant au nouveau point de vue (figure 23). Ensuite, comme pour un lancer de rayons, la couleur de chaque pixel de la nouvelle image va être calculée à l'aide d'un rayon passant par le centre de la caméra et par le pixel traité. L'utilisateur définit une distance minimum et une distance maximum entre lesquelles un point M va parcourir le rayon par intervalles réguliers. Pour chaque position du point M sur le rayon, on calcule la couleur du projeté de M sur toutes les images d'entrée. Ces couleurs sont stockées puis comparées entre elles afin de définir, pour un rayon donné, la position du point M présentant la meilleure homogénéité de couleur sur les images en entrée. Notons que cette méthode revient à comparer des segments de droites épipolaires entre les images de départ.

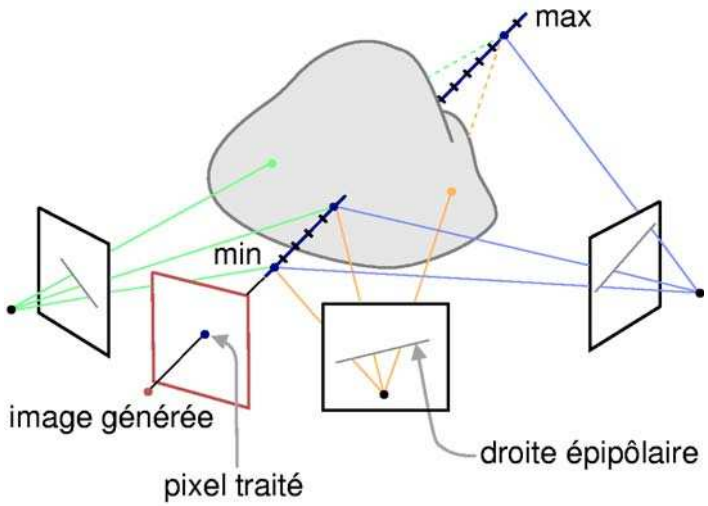


Figure 23 : Image-Based Priors.

La comparaison des couleurs s'effectue à l'aide des lois de Baye et des propriétés des priors. Ces outils statistiques permettent de faire des choix judicieux lors de situations ambiguës.

Cette méthode permet ainsi de réaliser de nouvelles vues photo-réalistes d'une scène à partir d'une trentaine d'images. Les auteurs utilisent une technique d'autocalibrage pour calibrer les caméras de façon automatique, sans quoi la méthode devient difficilement utilisable. Les temps de calcul sont toutefois assez longs. Dans leurs tests, les auteurs parlent de 0,1 seconde par pixel (sans spécifier les caractéristiques de la machine) avec un programme fonctionnant sous Matlab. Cette méthode est une bonne introduction à l'algorithme des *Plane Sweep* détaillé dans le chapitre 3.

2.1.4 Morphing

Parmi toutes les méthodes de rendu à base d'images, le morphing [BN92,GBC95,TE99] est sans doute la plus connue et la plus répandue, notamment dans le milieu du cinéma et de la publicité. On trouve même une multitude de logiciels de morphing pour le grand public. Le morphing consiste à générer des images intermédiaires entre deux images de référence en effectuant une double interpolation sur les formes d'une part et sur les couleurs d'autre part (figure 24). Les techniques de morphing sont souvent composées de *warping* et de morphing à proprement parler. Le *warping* est une transformation continue d'un objet graphique concernant sa géométrie, sa topologie et d'autres attributs. Le morphing est un mélange continu de formes et de couleurs entre un objet graphique source et un objet graphique de destination. Ainsi, il est nécessaire d'établir des correspondances entre l'objet de départ et l'objet d'arrivée et éventuellement d'utiliser des éléments de contrôle qui jouent le rôle d'objets graphiques intermédiaires. Les correspondances peuvent se présenter sous la forme de paires de pixels, de segments de droites ou de maillage.

Figure 24 : morphing.

Chen et Williams [CW93] utilisent un transfert linéaire de pixels, c'est-à-dire que chaque pixel se déplace de façon linéaire de son origine vers sa destination en suivant une ligne droite. Smithe [S90] ainsi que Ruprecht et Müller [RM95] utilisent des correspondances de points à partir desquelles ils construisent un maillage. C'est sur le maillage qu'est appliquée l'interpolation. Birkholz et Jackél [BJ03] définissent des courbes qui agissent sur la géométrie de leur environnement. Ils associent une courbe de départ à une courbe d'arrivée et calculent une interpolation de courbes pour les images intermédiaires.

Les images intermédiaires ainsi calculées donnent de bons résultats pour des transformations artistiques mais ne sont pas toujours pertinentes lors d'un morphing entre deux points de vue d'une même scène. En effet, la géométrie de la scène et la gestion des occlusions ne sont pas prises en compte dans les techniques de morphing classiques.

Seitz et Dyer proposent dans [SD96] une méthode de morphing interpolant aussi les matrices de projection des images de départ et d'arrivée. Ils montrent que dans le cas de caméras ayant la même focale et partageant le même plan image, le morphing obtenu est géométriquement correct et fonctionne sans connaître aucune information géométrique sur la scène autre que des correspondances de points. Cependant dans les autres configurations, les vues intermédiaires risquent de contenir des distorsions rendant le résultat peu réaliste.

Jiang et al. [JWB04] proposent une méthode de *warping* bénéficiant de l'accélération des cartes graphiques. Cette méthode nécessite une carte de profondeur de l'image initiale. A chaque pixel est associé un vertex dont les coordonnées 3d sont déterminées par les coordonnées 2d du pixel et sa coordonnée de profondeur. L'image est ainsi subdivisée en facettes (figure 25). Chaque vertex est envoyé dans le pipeline OpenGL où il est traité par un vertex program qui multiplie ses coordonnées homogènes par une matrice de warping. Cette matrice 4x4 est la même pour tous les pixels de l'image. Elle correspond à la combinaison des matrices model-view et projection entre l'image initiale et la nouvelle image. Les occlusions sont traitées directement par le z-test. Il est possible de voir apparaître des trous sur l'image générée dus aux éléments invisibles à partir de l'image initiale. Pour remédier à ce problème, il est possible d'appliquer le warping à plusieurs images de référence et de les mélanger.

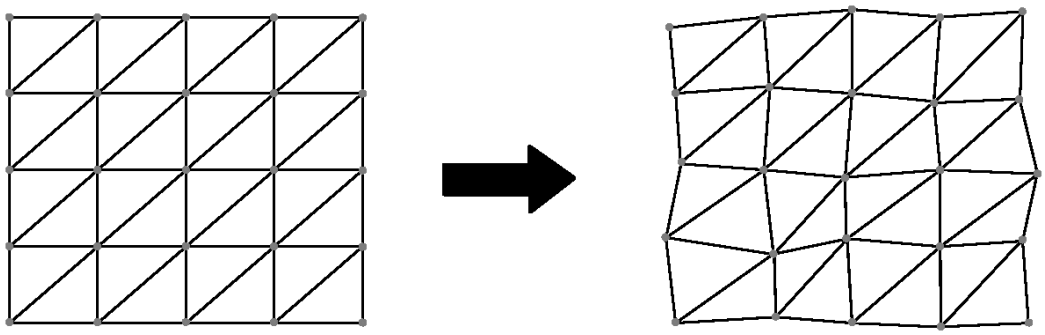


Figure 25 : warping par maillage.

2.1.5 Projective Grid Space caméras non calibrées

Saito et Kanade [SK99] présentent une méthode de reconstruction à partir d'images non calibrées en utilisant uniquement des combinaisons de relations épipolaires. Un point 3d est alors localisé par ses coordonnées (p,q) dans une première image et par une des composantes de ses coordonnées dans la seconde image. La relation épipolaire suffit alors à lever l'ambiguïté sur la seconde composante. La localisation de ce point 3d sur une nouvelle image nécessite de connaître les relations épipolaires entre cette image et les deux images de référence. Comme l'illustre la figure 26, le projeté du point (p,q,r) sur la nouvelle vue correspond à l'intersection des deux droites épipolaires reliant cette nouvelle vue aux deux images de référence. Pour optimiser la précision du calcul de cette intersection, il est préférable d'éviter de disposer les caméras sur un même plan. La phase de reconstruction est réalisée par extraction de silhouettes (*Visual Hulls*). L'objet à reconstruire est alors modélisé par voxels à l'aide de *marching cubes*.

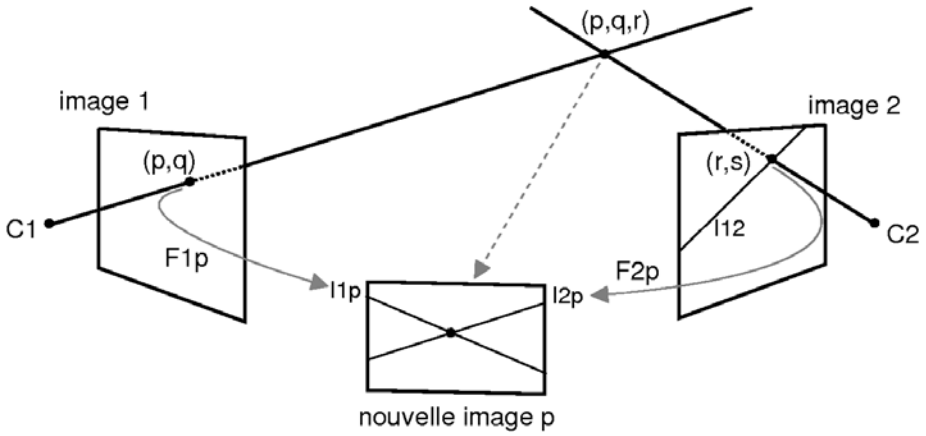


Figure 26 : Projective Grid Space, principe de base.

Yosuke et Saito [YS05] adaptent cette méthode pour du rendu à base de vidéos provenant de caméras mobiles non calibrées.

Yaguchi et Saito [YS06] améliorent le rendu en créant un maillage du modèle à partir de la modélisation par voxels. Ensuite, de façon itérative, chaque point du maillage est repositionné de telle sorte que sa nouvelle position maximise une fonction de score dépendant :

- de la corrélation des textures (corrélation croisée normalisée après une transformation affine des éléments d'image à comparer),
- de la continuité spatiale du maillage pour obtenir une surface plutôt « lisse »,
- de la régularité du maillage pour éviter d'obtenir des triangles disproportionnés,
- de l'adéquation avec les informations de silhouette de telle sorte qu'aucun sommet ne dépasse de l'enveloppe créée par l'extraction de silhouette. Ce critère concerne essentiellement les points situés sur une surface concave de l'objet.

Entre chaque itération, certains triangles du maillage sont fusionnés ou divisés de façon à alléger la contrainte sur la régularité du maillage. Plusieurs centaines d'itérations sont nécessaires pour converger vers un maillage stable.

2.1.6 Bilan

La première partie de cette section concerne les cartes de disparité et les méthodes permettant de les construire. Ces cartes de disparité ne permettent pas directement de générer une nouvelle vue de la scène cependant elles sont utilisées par un grand nombre de méthodes de rendu à base d'images ou de reconstruction. La gamme de ces outils de calcul de cartes de disparité est assez large puisqu'elle comprend des méthodes très rapides et peu robustes ainsi que des méthodes lentes et plutôt robustes. Dans la suite de l'état de l'art, nous détaillerons des méthodes faisant référence à ces techniques d'appariement.

Les autres méthodes décrites dans cette section permettent quant à elles de générer de nouvelles images de la scène. Les méthodes utilisant les voxels ou les poxels sont relativement coûteuses en temps de calcul et en espace mémoire. Les objets reconstruits sont donc difficiles à manipuler et leur résolution est rapidement limitée. Ces reconstructions sont toutefois relativement fidèles aux originaux.

Les *Image-Based Priors* sont eux aussi assez coûteux en calculs. Cependant ils permettent de générer une nouvelle image de la scène de façon relativement précise sans effectuer explicitement de manipulation géométrique.

En ce qui concerne les techniques de morphing, nous avons vu qu'il en existe certaines variante gérant les effets de perspective. Cependant les vues générées sont uniquement des vues intermédiaires entre les images de référence.

Les méthodes de *Projective Grid Space* fonctionnent sans les contraintes de calibrage habituelles mais les calculs supplémentaires ralentissent l'application de façon non négligeable. Les optimisations géométriques facultatives rajoutent un temps de calcul proche de 5 minutes par images. Notons toutefois que la qualité visuelle des images générées est largement superieure après ce traitement.

Enfin, il est intéressant de remarquer que la plupart des méthodes présentées dans ce chapitre sont réputées pour être robustes cependant seule la méthode des poxels est en mesure de gérer correctement les scène ayant des reflets spéculaires. Nous présentons dans la section suivante des méthodes de rendu à base d'images plus rapides à l'exécution et capables de gérer ce genre de reflets spéculaires.

2.2 Rendu à base d'images temps-réel

Ce chapitre traite des méthodes nécessitant un long prétraitement avant de pouvoir effectuer le rendu, qui lui est temps réel. Ce rendu en temps réel implique que ces méthodes sont capables de générer une nouvelle image dont le point de vue est déterminé de façon interactive par l'utilisateur.

Nous commencerons avec les méthodes n'utilisant qu'une seule image pour représenter la scène. Un long prétraitement rarement automatique précède le rendu. Nous décrirons ensuite des méthodes de rendu inspirées des imposteurs utilisés en synthèse d'images. Nous continuerons avec des méthodes de rendu à base d'images parmi lesquelles figurent les

méthodes de *Plenoptic*, le *Lumigraph* et le Light Field. Ces méthodes ont inspiré un grand nombre de chercheurs. Enfin, nous terminerons par une vue d'ensemble des méthodes de reconstruction d'une scène statique à partir d'une séquence vidéo.

2.2.1 3d à partir d'une seule image

La problématique de la reconstruction d'une scène 3d à partir d'une seule image réside dans la difficulté à attribuer une profondeur à chacun de ses pixels. Il existe deux approches pour résoudre ce problème, soit une carte de profondeur est établie manuellement ce qui requiert beaucoup de temps et d'énergie. Soit des éléments de géométrie sont extraits de l'image comme par exemple les lignes de fuite. En modifiant ces lignes de fuite, il devient alors possible de montrer la scène sous un autre point de vue.

2.2.1.1 Lignes de fuite

L'étude des lignes de fuite d'une image permet de s'affranchir en partie d'une carte de profondeur pour générer une nouvelle vue d'une scène à partir d'une seule image. C'est ce que montrent Horry et al. Dans leur *Tour into the Picture* [HAA97].

Comme les autres méthodes, celle-ci nécessite un travail préalable de la part de l'utilisateur. Pour commencer, il faut extraire de l'image les objets susceptibles de générer des occlusions lors d'un changement de point de vue de la caméra. La figure 27 montre une image à partir de laquelle on sépare les objets centraux de la scène et ceux de l'arrière-plan.

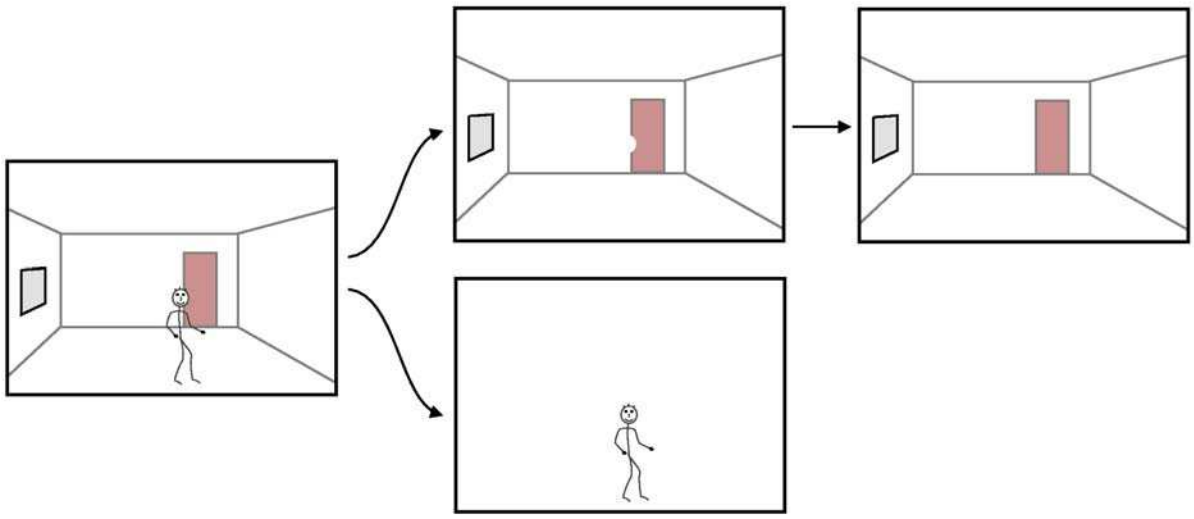


Figure 27 : Tour into the Picture, extraction de premier plan et d'arrière plan.

Cette étape engendre des trous sur l'arrière plan qu'il faut combler manuellement. Notons par exemple le trou laissé par la tête du personnage au niveau de la porte.

L'opération suivante consiste à déterminer les points de fuite de l'image à partir d'objets connus afin d'en extraire des propriétés projectives. Dans les images représentant des bâtiments avec beaucoup d'angles droits, il y a souvent trois points de fuite dont en général au moins un qui ne soit pas à l'infini ce qui est le cas dans notre exemple (voir figure 28). Si la scène photographiée manque de droites parallèles, il est possible d'utiliser les coniques

représentant des cercles ou sphères dans la scène originale pour déterminer les lignes de fuite.

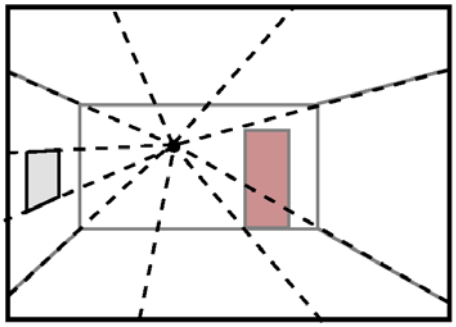


Figure 28 : Tour into the Picture, dettection des lignes de fuites.

L'utilisateur doit ensuite modéliser d'une part les objets de l'arrière plan et d'autre part les objets extraits de l'image (figure 29). Cette modélisation consiste à attribuer un plan à chaque objet de la scène et nécessite parfois des approximations sur le caractère coplanaire d'objets voisins. Pour des objets simples comme un personnage, un plan texturé (cf. *billboard* au chapitre 2.2.2.2) est souvent suffisant.

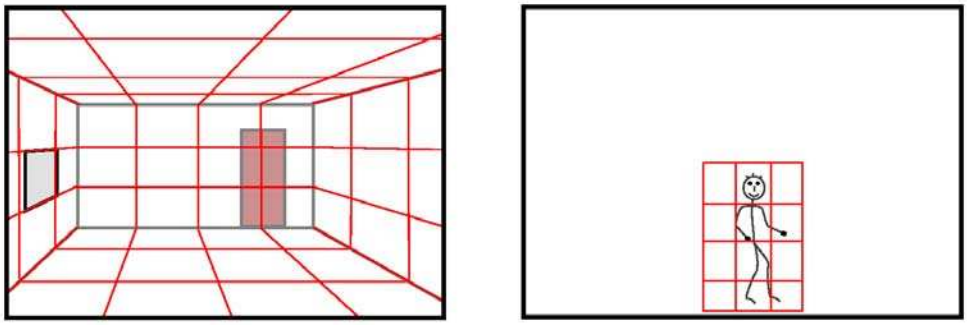


Figure 29 : Tour into the Picture, modélistaion par plans.

Finalement, pour générer une nouvelle image, il suffit de modifier les points de fuite de l'image en veillant à ce qu'ils restent compatibles avec les paramètres de la caméra virtuelle. Sur la figure 30, on peut constater que le point de fuite a été déplacé vers la droite ce qui change la perspective de l'image, il n'y a plus d'occlusion entre la tête du personnage et la porte.

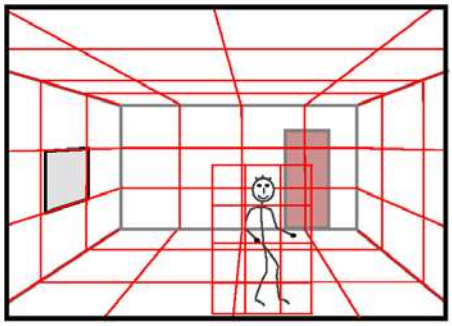


Figure 30 : Tour into the Picture,création d'une nouvelle vue en déplaçant les points de fuite.

Le rendu s'effectue en temps réel et les résultats sont assez spectaculaires. Il arrive toutefois que les objets représentés par un *billboard* manquent de profondeur. Cette méthode fonctionne aussi très bien avec des peintures dès lors qu'elles respectent les principes de la géométrie euclidienne et qu'elles présentent suffisamment d'indices pour en extraire les lignes de fuite.

Kang et al. [KPA01] proposent une adaptation du *Tour into The Picture* pour les images panoramiques.

2.2.1.2 Layers

Les méthodes à base de calques (*layers*) permettent de générer une scène 3d à partir d'une seule image en lui associant une carte de profondeur. Oh et al. [OCD01] proposent un éditeur et un visualiseur prenant en entrée une image et son angle de vue. La scène est représentée par une série de calques de profondeur prenant en compte la couleur, la profondeur, les normales et l'illumination. Cette carte de profondeur doit être faite manuellement ainsi que les parties cachées de la scène. Pour cela, deux outils sont proposés : le *clone broshing tool* permettant de faire des « copier coller » tenant compte de la perspective et le *texture illuminance decoupling filter* permettant de modifier l'illumination de la scène. D'autres outils pratiques sont fournis pour faciliter la création de la carte de profondeur, comme par exemple des modèles de sphères, de cônes, de cylindres ou bien de pyramides, ainsi que des outils pour modéliser le sol avec la perspective adéquate et des outils pour faire des murs à partir de leur plan 2d. La carte de profondeur des visages est faite à l'aide d'un modèle 3d de visage déformé et plaqué sur l'image. Cette méthode permet une navigation en temps-réel et un rendu photo réaliste mais nécessite un gros travail de la part de l'utilisateur. Les auteurs parlent de 10 heures de travail pour la segmentation et 3 heures pour la carte de profondeur pour une scène standard.

Dans leur *Layered Depth Images* [SGH98], Shade et al. utilisent plusieurs calques de profondeur. Ainsi, un pixel peut se voir attribuer plusieurs profondeurs en cas d'occlusion.

2.2.1.3 Automatic Photo Pop-up

Hoiemet al. [HEH05] proposent d'effectuer la reconstruction approximative d'une scène 3d à partir d'une seule image de façon entièrement automatique. L'image traitée doit toutefois satisfaire un certain nombre de contraintes. Elle doit notamment représenter une scène extérieure, contenir un et un seul sol ainsi qu'un et un seul ciel. Les autres éléments de la photo doivent appartenir à des plans verticaux, eux-mêmes posés directement sur le sol et positionnés à angle droit entre eux. Les bâtiments ayant une architecture classique conviennent parfaitement. Cette méthode fonctionne en quatre étapes. La première étape consiste à effectuer une extraction de superpixels [FH04], sortes de régions de couleur relativement homogènes. Ensuite, sont constituées des constellations formant des groupes de superpixels. Dans l'idéal, chaque constellation correspond à un objet différent. Pour plus de robustesse, plusieurs choix de constellations sont proposés. Ensuite, chaque constellation est identifiée comme appartenant au sol, au ciel ou à un objet. Cette étape utilise de la reconnaissance de forme ainsi que de l'intelligence artificielle et une base de données d'objets. Vient ensuite l'étape de modélisation commençant par une estimation de la ligne d'horizon. Le ciel est alors retiré de l'image, le sol modélisé et les éléments verticaux sont positionnés sur des *billboards*. Finalement, le rendu s'effectue en temps-réel. Le processus de création du modèle dure environ 1 minute 30 secondes pour une image 800×600 sur un

Athlon 2 GHz avec un programme codé sous Matlab. Cette méthode est entièrement automatique mais ne fonctionne pas sur un grand nombre d'images, même lorsque celles-ci répondent aux contraintes détaillées précédemment. Globalement, les approximations géométriques sont souvent trop fortes pour que le rendu soit visuellement satisfaisant.

2.2.2 Imposteurs

Les imposteurs sont de simples textures représentant un objet 3d. Leur emploi permet de faire régulièrement l'économie du rendu de l'objet 3d en question. Nous présentons dans ce chapitre des adaptations des imposteurs et des méthodes dérivées au rendu à base d'images. Nous les décrirons du plus simple au plus sophistiqué.

2.2.2.1 Sprites

Les *sprites* sont de simples images directement dessinées sur l'image traitée. L'exemple le plus simple est celui du curseur de la souris sur un écran d'ordinateur. Les *sprites* sont rarement issus d'images photographiques cependant cela peut être le cas, par exemple pour simuler le tableau de bord d'une voiture dans un jeu vidéo. Dans ce cas là, l'image du tableau de bord est réaliste mais on ne peut pas lui attribuer de modèle d'illumination pour réagir à l'éclairage de la scène. Associée au *stencil buffer* (permettant notamment de gérer des masques durant le rendu), cette méthode permet toutefois un rendu assez rapide.

2.2.2.2 Billboards

Le *billboard* est un dérivé du *sprite* possédant des propriétés plus évoluées. Il est généralement composé d'une texture plaquée sur un polygone. Les *billboards* sont souvent utilisés en synthèse d'images pour remplacer un objet complexe nécessitant un grand nombre de facettes. Il existe plusieurs sortes de *billboards* que l'on peut classer en trois catégories :

- **Les billboards disposés face au plan image de la caméra** (*screen-aligned*). Ce sont les plus simples et ils ne diffèrent pas beaucoup des *sprites*. La figure 31 représente la pyramide de projection d'une caméra vue de dessus, les *billboards*, en gris, sont disposés face au plan image de la caméra.

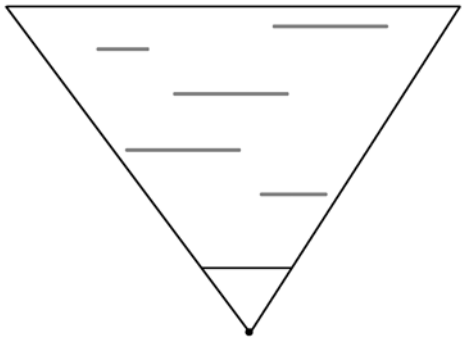


Figure 31 : *billboards sreen-aligned*.

Ces *billboards* perdent de leur efficacité si la caméra s'en approche trop et ils sont souvent victimes d'un effet « pancarte » qui trahit leur manque d'épaisseur.

- **Les billboards orientés vers le centre de la caméra** (*world-oriented*). Ils sont un peu plus élaborés que les *billboards screen-aligned* et représentent mieux les objets présentant une symétrie cylindrique. La figure 32 montre la pyramide de projection d'une caméra vue de dessus, les *billboards* sont orientés vers le centre de la caméra.

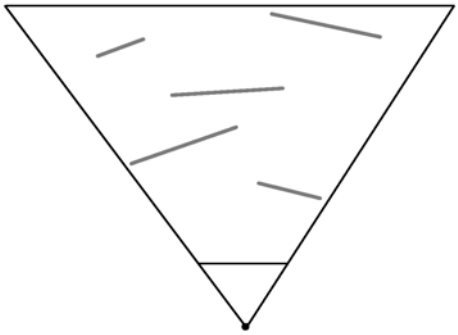


Figure 32 : *billboards world oriented.*

Ce genre de *billboard* peut, par exemple, être utilisé pour simuler des nuages vus par un avion. Cependant là encore, ils perdent leur crédibilité si l'on s'en approche trop ou pire, si on les traverse.

- **Les billboards indépendants de la caméra**. Ils sont généralement disposés dans la scène de telle sorte qu'ils soient efficaces à partir de plusieurs points de vue. La figure 33 montre deux *billboards* formant un arbre. On peut trouver ce genre de *billboards* dans des jeux vidéo pour représenter des objets lointains. Lorsque la caméra s'en rapproche, ils sont souvent remplacés par des objets à facettes plus complexes.

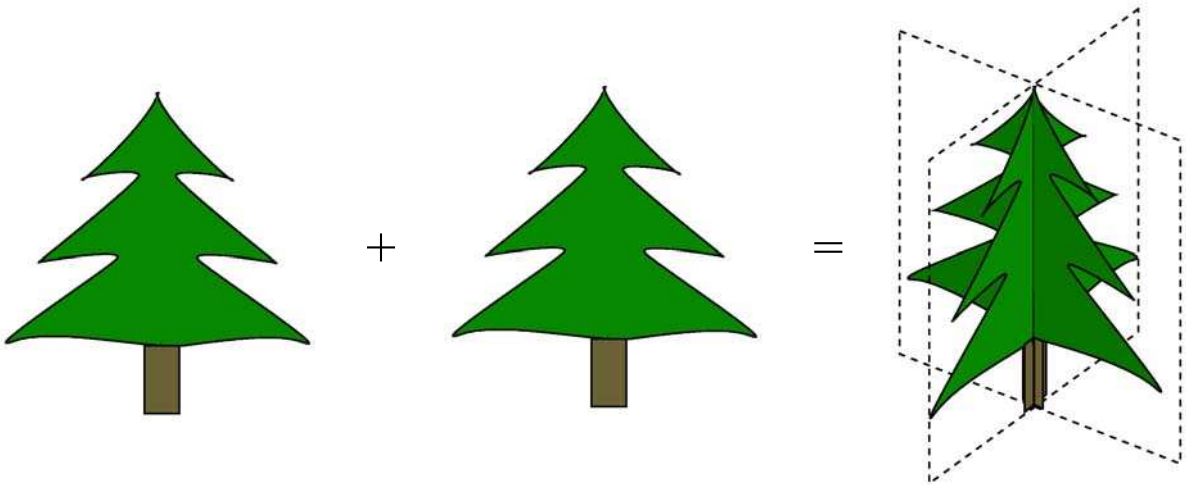


Figure 33 : *billboards* combinés.

En règle générale, les *billboards* ne sont efficaces que lorsque l'épaisseur de l'objet qu'ils remplacent est petite devant la distance qui les sépare de la caméra. Les textures utilisées ne proviennent pas nécessairement de photos et sont souvent créées « à la main ». Pour plus de précisions, le lecteur peut se référer à [AH02].

2.2.2.3 Imposteurs

Un imposteur [MS95] est un *billboard* créé dynamiquement et remis à jour suivant la position de la caméra. On l'utilise en synthèse d'images pour le rendu en temps-réel. Il permet d'accélérer le rendu en évitant d'afficher à chaque nouvelle image l'objet complexe qu'il représente. Le rendu de l'objet à remplacer est effectué hors écran (*off-screen*) directement dans une texture et selon le point de vue de la caméra. Pour le rendu de l'image suivante, si la caméra ne bouge pas, bouge peu ou effectue une rotation sur elle-même, l'imposteur n'est pas recalculé. De plus, la résolution de l'imposteur est souvent inférieure à celle de l'écran ce qui accélère davantage les calculs. Il existe trois conditions impliquant chacune la mise à jour de l'imposteur. La première condition concerne la résolution de l'imposteur. Celle-ci doit être paramétrée de telle sorte qu'aucun effet de « pixellisation » ne soit visible sur l'image générée et doit être minimale pour économiser de la mémoire et des calculs. La seconde et la troisième condition concernent les changements de points de vue. Si la caméra avance ou tourne autour de l'imposteur, il faut le mettre à jour. Ces trois conditions sont contrôlées par un simple calcul d'angles détaillé dans [AH02] et dans [SS96] qui effectue le rendu de scènes exclusivement composées d'imposteurs. Enfin, les imposteurs permettent parfois d'ajouter un effet de profondeur de champ dans une scène par simple filtre gaussien sur les textures. Pour plus de précisions sur les imposteurs, le lecteur peut se référer à [AL99, F01, WWS01].

Dans [LS97], Lengyel et Snyder représentent un objet complexe par un ou plusieurs imposteurs choisis judicieusement en fonction des articulations éventuelles de l'objet. Prenons par exemple un animal de profil constitué d'un imposteur pour la tête et d'un autre pour le corps. Si l'animal bouge la tête, il n'est pas nécessaire d'effectuer une mise à jour, il suffit simplement de repositionner l'imposteur représentant la tête. Les *billboards* affichés sont pourvus de calques : un pour les couleurs diffuses, un pour les couleurs spéculaires et éventuellement un pour les ombres ce qui là encore permet de réduire au minimum les calculs en minimisant les mises à jour.

Yamazaki et al. [YSK02] proposent de générer dynamiquement des *billboards* à partir d'un ensemble d'images plutôt qu'à partir d'un rendu en images de synthèse. L'objet à modéliser est filmé sur fond bleu avec des caméras calibrées. Une extraction de contours sur chaque image permet alors d'en calculer la géométrie approximative. Un nuage de *billboards*, ou micro-facettes, est alors disposé de façon cohérente avec cette géométrie. Chacune de ces micro-facettes est orientée vers la caméra et texturée avec les images initiales mélangées de façon judicieuse selon le point de vue de la caméra. Un objet peut être constitué de plus de dix mille micro-facettes, le but étant ici d'avoir un rendu photo-réaliste, notamment sur des objets difficilement modélisables comme des jouets en peluches ou des cheveux. Le rendu s'effectue en temps-réel.

2.2.2.4 Nailboards

En ajoutant à un *billboard* une carte de profondeur, on obtient un *nailboard* [S97] aussi appelé *depth sprite* [SGH98]. La texture associée au *nailboard* est au format RVBΔ où Δ représente la distance entre le polygone et l'objet réel. Cette différence de profondeur est directement intégrée dans le processus de rendu afin de mieux gérer l'occlusion entre le *nailboard* et un autre objet de la scène qui s'interpénètreraient. Tout comme le *bump mapping*, les *nailboards* permettent aussi d'adapter l'illumination du polygone texturé aux sources lumineuses de la scène. Shade et al. [SGH98] appliquent leur *depth sprite* à des images avec leurs *Layered Depth Images.*

2.2.2.5 Bilan

Finalement, nous venons d'explorer quelques méthodes permettant de remplacer un objet d'une scène par une représentation plus ou moins fidèle et plus ou moins coûteuse en calculs. Shade et al. [SGH98] définissent un classement de préférence déterminé suivant la distance entre l'objet à modéliser et le centre de la caméra. La figure 34 illustre ce classement sur lequel il apparaît que plus un objet est loin du centre de la caméra et moins l'approximation de sa géométrie doit être fine. Inversement, plus un objet est proche du centre de la caméra et plus il faut que son modèle lui soit fidèle. Ceci est dû au fait que plus un objet est mince relativement à la distance le séparant du centre de la caméra, moins son projeté bouge sur l'image lors d'un mouvement de la caméra. Par ailleurs, nous avons constaté que ce genre de méthodes n'utilise pas uniquement des images de synthèse mais aussi des images provenant de photographies.

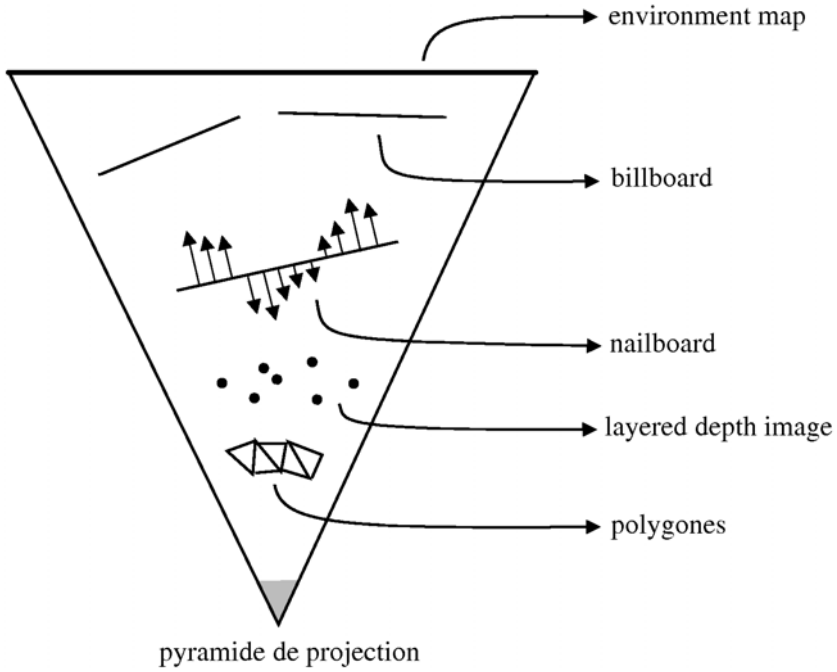


Figure 34 : classement des méthodes d'imposteurs.

2.2.3 QuickTimeVR

Chen [C95] propose l'application QuickTime®VR implémentée sur le logiciel QuickTime d'Apple. Cette méthode de rendu fonctionne avec un ensemble d'images panoramiques. A partir d'une dizaine de photos prises d'un même point de vue, il est possible de créer une image panoramique en projetant sur un cylindre (ou une sphère) les images de départ. Pour cela, il suffit de connaître la focale de la caméra et de déterminer des points de correspondance entre deux vues consécutives. Ce principe est illustré par la figure 35 qui représente une série d'images projetées sur un cylindre vu de dessus.

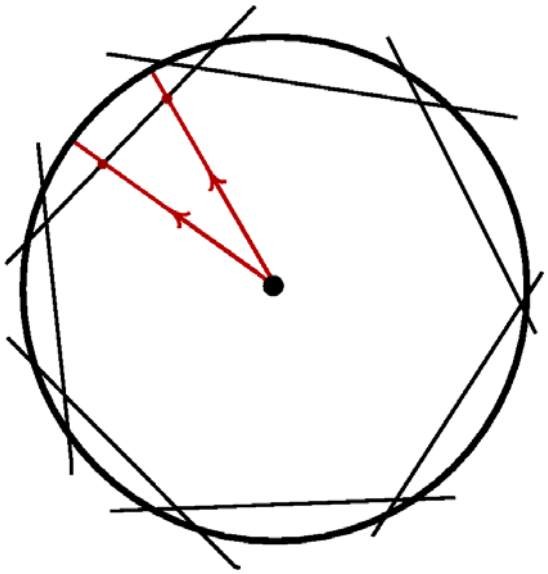


Figure 35 : principe de création d'une image panoramique.

La figure 36 montre trois images parmi les dix ayant servi à la création du panoramique de la figure 37.

Figure 36 : création d'une image panoramique : images de départ.

Figure 37 : création d'une image panoramique, image générée.

Chen utilise un réseau d'images panoramiques pour représenter une scène complète. Chaque image panoramique correspond à un nœud et l'utilisateur peut naviguer de nœud en nœud pour simuler les déplacements (figure 38). Ceux-ci sont toutefois restreints aux nœuds avoisinant le nœud courant. Les rotations sont calculées à l'aide d'*environment mapping* cylindrique consistant à texturer avec les images panoramique un cylindre centré sur le nœud traité. Cette technique permet de restituer une image perspective (où les lignes droites de la scène sont droites sur l'image) à partir d'une image panoramique. Enfin, le zoom est géré en utilisant des résolutions multiples pour chaque image panoramique à l'aide de *mipmapping* (gestion des niveaux de détails d'une image par la carte graphique).

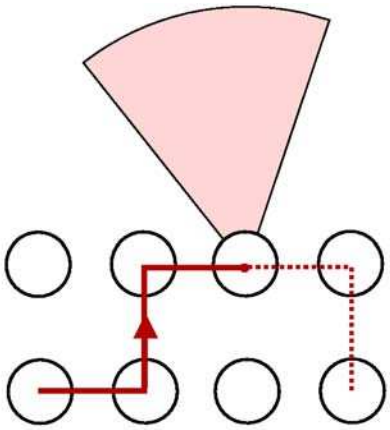


Figure 38 : vue de dessus d'un parcours parmi les nœuds d'un réseau d'images panoramiques.

QuickTime®VR permet donc de naviguer de façon discrète, c'est-à-dire de nœud en nœud, dans une scène virtuelle en temps réel. Cette méthode n'utilise aucune notion de géométrie en ce qui concerne la scène à représenter. De plus, la construction des vues panoramiques peut se faire de façon partiellement automatique en précisant la focale de la caméra ou bien de façon complètement automatique [AZP05].

Dans la lignée de *QuickTime*®VR, Kimber et al. proposent le *FlyAbout* [KFL01] constitué de quatre caméras disposées à angles droits de telle sorte qu'elles couvrent tout l'horizon. L'acquisition vidéo s'effectue en déplacement sur une route. Le *FlyAbout* génère ainsi une vidéo panoramique représentant ce que pourrait voir un observateur à bord du véhicule. Lors de la restitution, l'utilisateur peut orienter la caméra virtuelle et peut choisir un itinéraire lorsque le véhicule rencontre un croisement.

2.2.4 Plenoptic Modeling

McMillan et Bishop [MB95] introduisent le *Plenoptic Modeling*, issu de la fonction *Plenoptic* introduite par Adelson et Bergen [AB91] qui décrit à chaque endroit et dans chaque direction ce que peut voir un observateur (figure 39). Cette fonction *Plenoptic* se note :

$$p=P(\theta,\phi,\lambda,V_x,V_y,V_z,t)$$

avec :

- θ et ϕ : 2 angles en coordonnées sphérique correspondant à la direction regardée.
- λ : longueur d'onde observée.
- V_x,V_y,V_z : position de l'observateur.
- t : temps (pour les scènes dynamiques).

Dans le cas de scènes statiques, le paramètre *t* peut être supprimé.

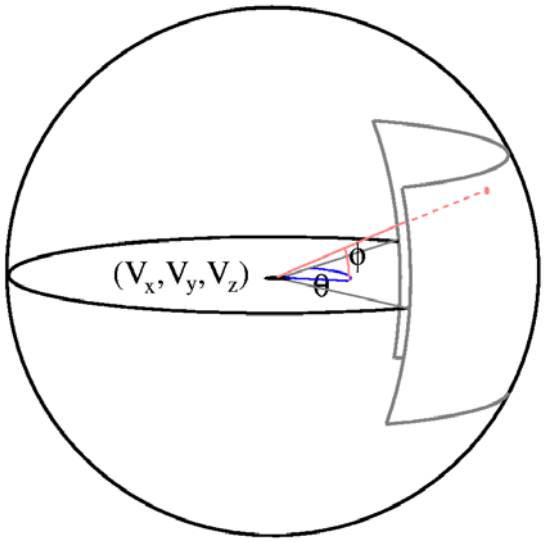


Figure 39 : fonction Plenoptic en coordonnées sphériques.

Comme dans *QuickTime*®VR [C95] de Chen, le rendu par *Plenoptic* fonctionne à partir d'images panoramiques pour lesquelles la fonction *Plenoptic* est naturellement connue (à un changement de référentiel près). La méthode de rendu consiste alors à générer une représentation continue de la fonction *Plenoptic* à partir d'échantillons discrets. En clair, il s'agit d'effectuer une interpolation entre les images panoramiques.

Le rendu par *Plenoptic* fonctionne en deux étapes. La première étape consiste à générer les images panoramiques cylindriques, à les calibrer les unes par rapport aux autres et à établir des points de correspondance entre elles. La seconde étape traite le déplacement dans la scène et le rendu.

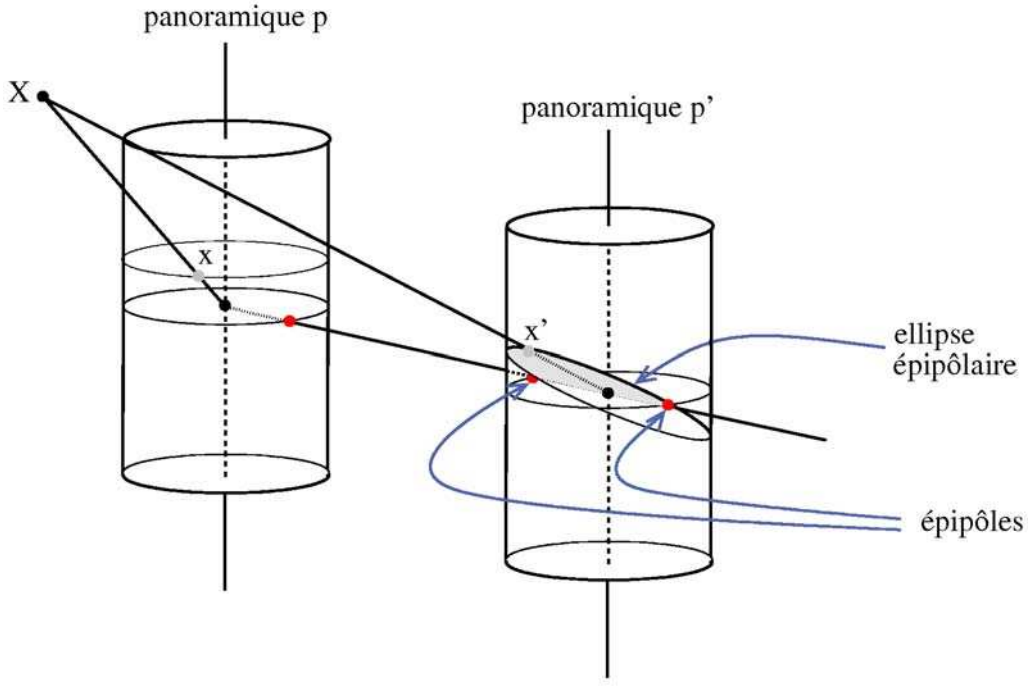


Figure 40 : relation épipolaire entre deux images panoramiques cylindriques.

Pour le calcul de points de correspondance entres images panoramiques, McMillan et Bishop établissent une relation épipolaire entre deux images panoramiques cylindriques. Ces relations épipolaires apparaissent sous forme de quadriques plutôt que sous forme de droites dans le cas d'images perspectives. La figure 40 montre la relation épipolaire entre deux images panoramiques cylindriques. La droite reliant les deux centres de projection des deux images panoramiques coupe chaque cylindre en deux points ce qui explique qu'une image panoramique ait deux épipôles. Le plan épipolaire passant par les deux centres de projection des deux images et par un pixel de la première image coupe le cylindre de la seconde image avec une conique. Lorsque les images sont mises à plat (figure 41), ces coniques deviennent des quadriques.

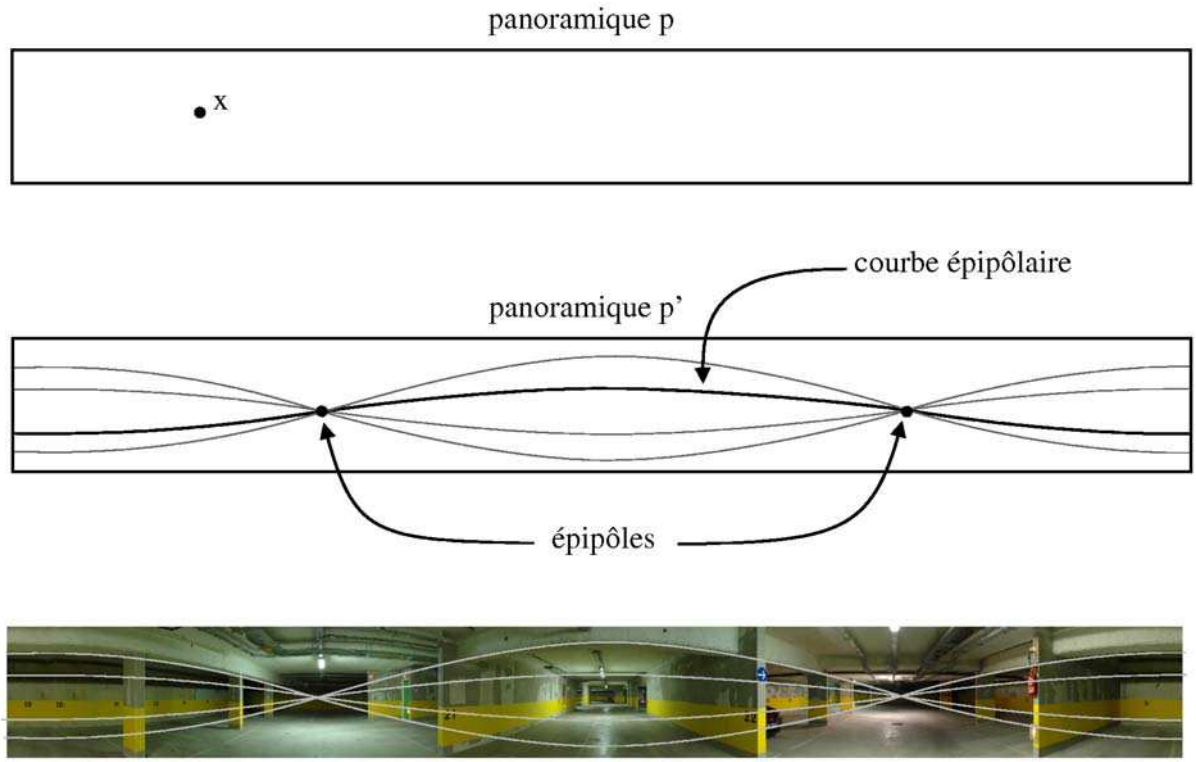


Figure 41 : droites épipolaires sur unr image panoramique cylindrique.

Cette contrainte épipolaire permet de calculer des correspondances plus rapidement et de façon plus robuste. Le rendu par *Plenoptic* requiert un maximum de corrélations entre les images panoramiques (si possible, tous les pixels) ce qui est coûteux en prétraitement et en espace mémoire.

L'étape de rendu consiste d'abord à générer un support cylindrique associé à l'emplacement virtuel de l'observateur. Ensuite, une image panoramique est construite sur ce support à partir des images panoramiques voisines. Les points corrélés de ces images panoramiques de référence sont triangularisés puis projetés sur le support cylindrique. Finalement, l'image panoramique générée est projetée sur un plan pour obtenir l'image et le point de vue désirés. Concrètement, ces trois opérations sont faites simultanément sans passer par une représentation panoramique cylindrique, ce qui engendre un gain de temps conséquent.

De cette façon, le rendu par *Plenoptic* permet de générer des images photo-réalistes et temps-réel dans n'importe quelle direction (sauf en direction verticale). Cette méthode

nécessite toutefois un long prétraitement pour générer les images panoramiques et calculer toutes les correspondances.

Pour plus de précision sur la fonction *Plenoptic*, le lecteur peut se reporter à [AB91] et [S02]. Koch et al. [KPH99] ainsi qu'à Aliaga et Carlbom [AC01] qui présentent une méthode de calcul de la fonction Plenoptic à partir de vidéos.

Seitz et Kutulakos [SK98] présentent dans leur *Plenoptic Image Editing* une méthode permettant d'éditer simultanément un ensemble d'images d'une même scène 3d. Grâce à leur technique de décomposition volumétrique de la fonction *plenoptic*, une modification de quelques images suffit à mettre à jour l'ensemble des images.

Shum et al. [SH99] réduisent le nombre de paramètres de la fonction *plenoptic* à trois en utilisant un procédé d'acquisition particulier. Plutôt que de calculer des images panoramiques dans une scène, l'acquisition s'effectue en déplaçant la caméra sur un cercle en veillant à ce qu'elle soit toujours orientée vers l'extérieur de façon radiale. Grâce à une table rotative indiquant l'orientation de la caméra, il n'est pas nécessaire de calibrer les images utilisées. Le rendu s'effectue en temps-réel et permet à l'utilisateur de se déplacer librement sur le cercle d'acquisition en regardant dans la direction de son choix.

2.2.5 Light Field Rendering

Tout comme le *Plenoptic Modeling* [MB95], le *Light Field Rendering* présenté par Levoy et Hanrahan [LH96] propose de représenter l'ensemble des rayons lumineux dans un volume déterminé à l'aide d'une fonction *L*. Contrairement à la fonction *Plenoptic* qui dépend de six paramètres, cette fonction n'en a que quatre. Elle est déterminée par deux plans situés en face de la scène et chaque rayon lumineux issu de cette scène est défini par son intersection (*u*,*v*) avec le premier plan et son intersection (*s*,*t*) avec le second plan (figure 42).

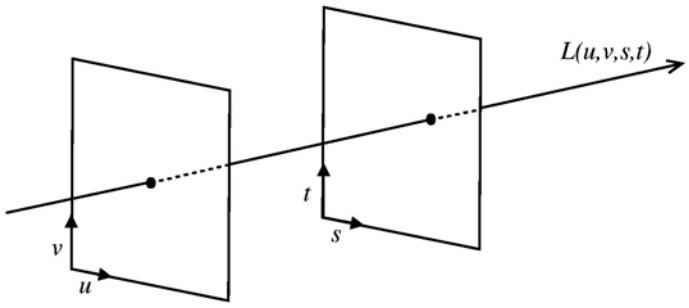


Figure 42 : Light Field, une fonction à quatre paramètres.

La fonction $L(u,v,s,t)$ renvoie la couleur (les composantes rouge, vert et bleu) du rayon passant par les points (u,v) du premier plan et (s,t) du second.

Cette méthode fonctionne en deux étapes, l'acquisition des données et la construction du Light Field puis le rendu. La première étape nécessite un dispositif particulier capable de traiter un ensemble assez dense d'images prises à partir d'un même plan. Pour cela, Levoy et Hanrahan utilisent une caméra sur bras articulé pilotée par ordinateur. Cette caméra prend des clichés à intervalles réguliers sur le plan (u,v). Chacune des images obtenues est projetée sur le plan (s,t) afin d'établir une correspondance pixel par pixel entre les deux plans.

Ce dispositif soulève trois problèmes : le premier est que la caméra doit s'orienter vers le centre de la scène pour éviter de perdre de l'information et de ce fait, les images capturées n'appartiennent plus au plan (u,v). Pour corriger cette erreur, il suffit d'appliquer aux images une homographie appropriée. Le second problème concerne les changements de focus lors de la mise au point de la caméra. Ceux-ci changent les paramètres intrinsèques de la caméra et fausse le calibrage. Pour y remédier, les auteurs augmentent l'éclairage de la scène et diminuent l'ouverture du diaphragme de la caméra afin d'augmenter sa profondeur de champ. Le fait de débrayer l'autofocus ne rend alors pas les images moins nettes. Enfin, le troisième problème concerne le stockage des données. En effet, le nombre d'images nécessaires est si grand que pour une simple scène, la mémoire utilisée dépasse facilement les 1,6 GB. Les données d'une image à l'autre étant toutefois assez similaires, l'ensemble des images se compresse bien. Levoy et Hanrahan proposent une méthode de compression atteignant des taux de compression de 100:1. Le processus d'acquisition des images est long, pour exemple, l'acquisition de 2048 photos prend environ 4 heures.

Pour effectuer le rendu, il suffit de placer dans la scène la caméra virtuelle composée de l'image à reconstruire et de son centre de projection. Comme pour un lancer de rayons, ce dispositif détermine un ensemble de rayons croisant les plans aux points (u,v) et (s,t). On attribue alors à chacun de ces rayons la couleur $L(u,v,s,t)$ (figure 43). Concrètement, la fonction $L(u,v,s,t)$ étant une fonction discrète, le rayon sélectionné est le rayon le plus proche du rayon demandé.

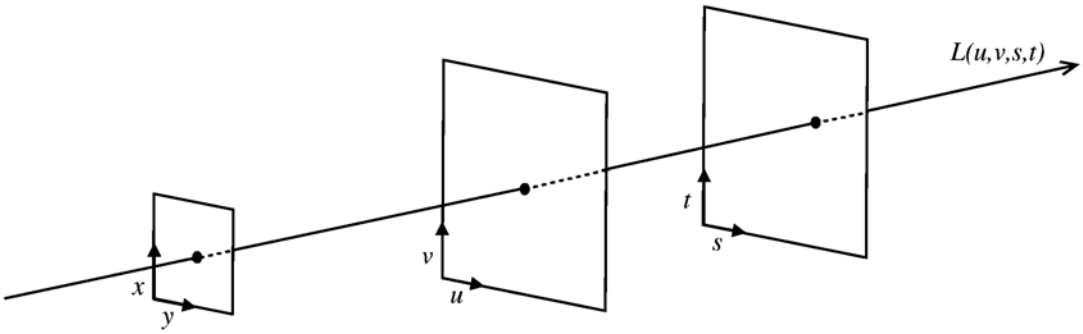


Figure 43 : rendu par Light Field Rendering.

Cette méthode permet de générer des images photo-réalistes en temps-réel. En utilisant un dispositif adéquat composé par exemple de quatre Light Fields entourant une scène (figure 44), il est possible de « faire le tour de la scène ». Enfin, cette méthode n'utilise aucune notion de géométrie en ce qui concerne la scène à représenter. Les principaux défauts de cette méthode sont d'une part la spécificité du matériel requis et d'autre part la durée des prétraitements et l'espace mémoire nécessaire.

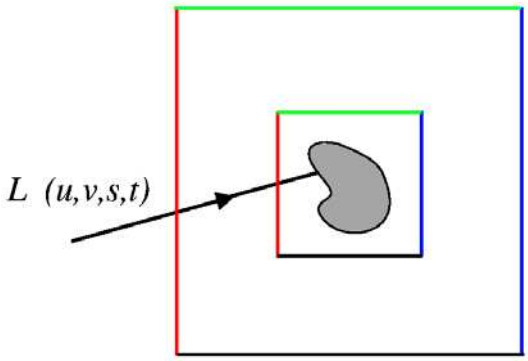


Figure 44 : quatre Light Fields disposés en carré autour de la scène

Le Surface Light Field introduit par [MRP98] et [NSI99] est une extension du Light Field appliqué à une surface quelconque plutôt qu'à un plan. Wood et al. [WAA00] définissent un protocole commençant par l'acquisition de points 3d représentant un objet. Cette acquisition d'effectue à l'aide d'un laser. Un maillage est alors calculé à partir du nuage de points. L'acquisition de photographies s'effectue à l'aide d'une caméra montée sur un bras articulé motorisé. Il en résulte un ensemble d'environ 500 images calibrées. L'utilisateur doit alors établir « manuellement » des correspondances entre les sommets du maillage et des ensembles d'images. Wood et al. utilisent alors une fonction attribuant une couleur RVB à chaque rayon émanant de chaque point de la surface du modèle à l'aide de lumisphères. Celles-ci sont uniquement définies sur les sommets du maillage et permettent par interpolation de générer une nouvelle vue de l'objet modélisé. Le rendu s'effectue en temps interactif grâce d'une part à un processus de compression basé sur une analyse en composante principale et d'autre part à un algorithme de contrôle de niveau de détails. Chen et al. [CBC02] développent une version utilisant des textures pour stocker de l'information et laissant à la carte graphique une grosse partie des calculs à effectuer. Le taux de compression atteint par cette méthode est très fort puisqu'il peut atteindre 5500:1. Cette méthode permet quasiment de multiplier par dix la vitesse d'affichage par rapport à la méthode non accélérée pour une qualité de compression moyenne. Les prétraitements demeurent toutefois relativement longs.

2.2.6 Lumigraph

Le *Lumigraph* proposé par Gortler et al. [GGS96] fonctionne sur le même principe que le *Light Field Rendering*. Cependant, alors que le Light Field ne tient pas compte de la géométrie de la scène, le *Lumigraph* essaie d'en tirer parti pour améliorer le rendu.

D'autres différences apparaissent notamment dans le mode d'acquisition des images. En effet, afin de s'affranchir d'une caméra sur moteur commandée par ordinateur, Gortler et al. proposent une méthode d'acquisition plus souple : on place l'objet filmé sur une mire de type « fond bleu » et on filme caméra à la main. Cette méthode restreint toutefois le contenu d'une scène à un objet principal. Ensuite, chacune des images est calibrée puis projetée sur les plans (s,t) et (u,v).

L'étape suivante consiste à effectuer un calcul approximatif de la forme 3d de l'objet de la scène. Pour cela, la silhouette de l'objet filmé est extraite grâce au fond bleu de la mire. Ensuite la scène est subdivisée en voxels (gérés par des octree) et seuls les voxels se projetant sur chaque silhouette sont gardés. Les voxels restants représentent la forme

approximative de l'objet. Notons que l'extraction de silhouettes ne permet pas de gérer les objets de forme concave et qu'il est donc recommander d'utiliser des objets convexes pour constituer la scène. Cette étape d'acquisition, de calibrage, d'extraction de silhouette et de calcul de la forme de l'objet est assez longue (une journée de calcul sur une SGI Indy en 1996).

C'est durant le rendu qu'intervient l'information de géométrie de la scène. La figure 45 montre un *Lumigraph* simplifié en deux dimensions, il n'y a plus que l'axe (*s*) à la place du plan (*s,t*) et l'axe (*u*) à la place du plan (*u,v*). Dans cet exemple, on cherche à calculer *L*(*s,u*) en pointillés. Le rayon le plus proche de (*s,u*) est (s_i,u_i), pourtant, le rayon (s_{i-1},u_i) correspond mieux à la solution cherchée. Connaissant la profondeur *z* à laquelle le rayon (*s,u*) intersecte la surface délimitant l'objet, il est possible de calculer pour un s_i donné, un *u'* qui correspond au mieux à la solution recherchée. Le théorème de Thalès établit la relation suivante :

$$u' = u + (s - s_i)\frac{z}{1-z}$$

Connaissant *u'*, il est possible d'interpoler de la meilleure façon entre u_{j-1} et u_j encadrant *u'*.

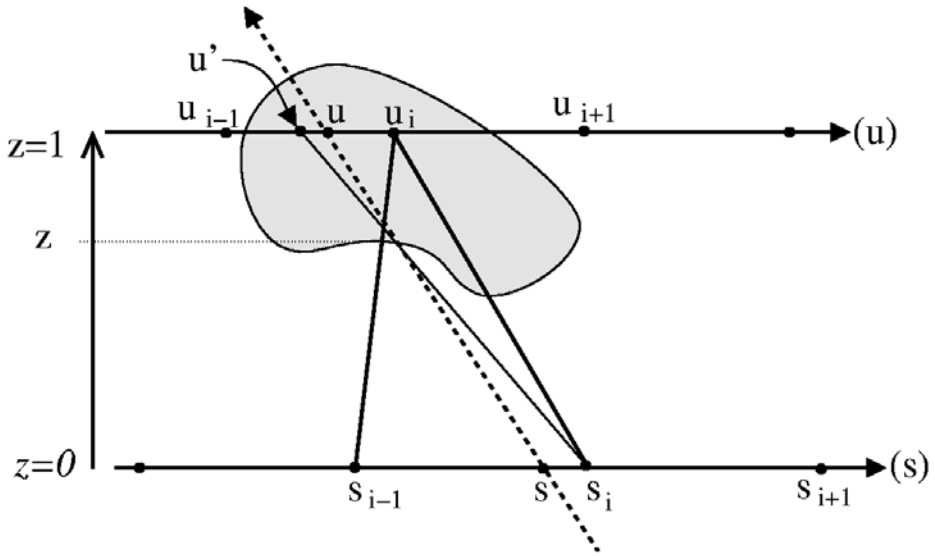


Figure 45 : Lumigraph, interpolation de deux rayons.

Le *Lumigraph* est donc une optimisation du Light Field permettant de s'affranchir de matériel contraignant. Il permet de générer en temps-réel des images photo-réalistes en utilisant la géométrie de l'objet filmé. Cette méthode impose néanmoins que la scène soit réduite à un objet dont la forme sera approximativement calculée. Elle nécessite en outre un long pré-traitement.

2.2.7 Méthodes de reconstruction

Comme leur nom l'indique, les méthodes de reconstruction créent un modèle 3d de la scène traitée et travaillent donc beaucoup sur la géométrie de la scène. Le nombre d'images en entrée peut varier d'une méthode à l'autre. Leur fonctionnement suit, à quelques variantes près, les étapes suivantes :

- **Acquisition des images** (figure 46). A la fin de l'acquisition, le programme dispose d'un ensemble d'images utilisables pour la reconstruction.

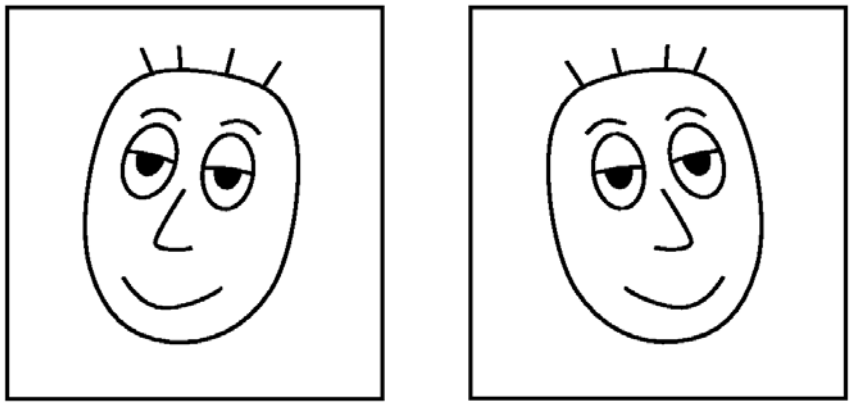


Figure 46 : images de départ.

- **Calcul de correspondances entre images** (figure 47) : il s'agit de trouver des pixels représentant les mêmes objets dans la scène. Cette étape commence en général par la détection de points d'intérêts, c'est-à-dire de points caractéristiques de l'image suffisamment discriminants pour être repérables (et identifiables) sur les autres images. Des détecteurs de coins sont souvent utilisés pour cette étape [HS88,DG93]. L'association de points d'intérêt entre deux images s'effectue grâce à des méthodes de corrélation [CC03]. Certaines méthodes proposent d'identifier des objets ou des motifs plutôt que des points d'intérêt. Dans le cas des objets, la comparaison est d'ordre géométrique et dans le cas de motifs, elle est colorimétrique et les motifs sont mis en correspondance par homographie. A la fin de cette étape, le programme dispose de correspondances entre les images. Il est courant que ces correspondances ne soient pas toutes bonnes et il est parfois intéressant d'avoir une idée de la proportion d'information erronée.

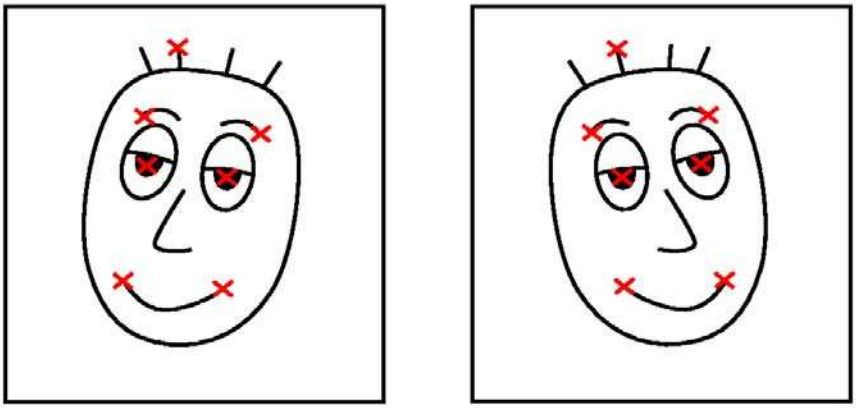


Figure 47 : calcul de points de correspondance.

- **Calcul de relations entre images** (figure 48) : cette étape consiste à calculer les relations épipolaires entre les images. Ce calcul permet aussi de supprimer les mauvaises correspondances de points grâce à des algorithmes de type RANSAC (voir Lacey et al. [LPT00] ou Torr et Zisserman [TZ00] pour MLESAC) en utilisant la contrainte épipolaire. A la fin de cette étape, le programme dispose donc de relations épipolaires entre images.

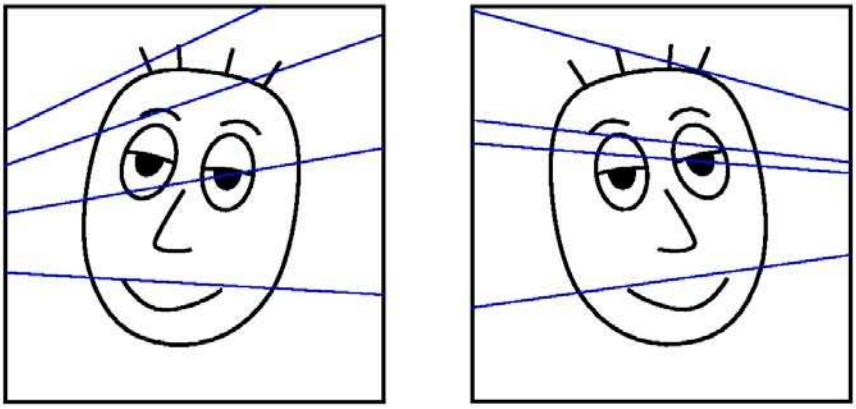


Figure 48 : relation épipolaires entre images.

- **Reconstruction grossière** (figure 49) : quelques correspondances entre les images suffisent à effectuer un calibrage perspectif (et grossier) des caméras. Par triangularisation, il est alors possible de localiser les points corrélés, la reconstruction sera toutefois perspective, c'est-à-dire qu'elle sera distordue. Pour la rendre métrique, il faut appliquer à la scène une homographie 3d ce qui est fait à l'étape suivante. Le calibrage s'effectue en commençant par les deux premières images puis en intégrant une par une les images supplémentaires de telle sorte qu'il ne soit pas nécessaire que les points d'intérêts le l'image à intégrer soient corrélés avec des points de la première image. A la fin de cette étape, les caméras sont calibrées grossièrement et pas de façon métrique. Le programme dispose alors d'une reconstruction 3d grossière des points utilisés pour les correspondances entre images.

Figure 49 : reconstruction grossière.

- **Autocalibrage** (figure 50) : c'est durant cette étape que l'on rend la reconstruction métrique. Il est alors nécessaire de faire quelques suppositions sur les paramètres intrinsèques de la caméra tels que supposer que les pixels du capteur de la caméra sont carrés ou bien que l'axe optique des caméras passe par le centre des images. Il est possible d'affiner le calibrage des caméras en vérifiant la cohérence de l'ensemble des

caméras par optimisation globale [TMH00, LA04]. A la fin de cette étape, les caméras sont calibrées correctement et le programme dispose d'une reconstruction métrique des points de l'étape précédente.

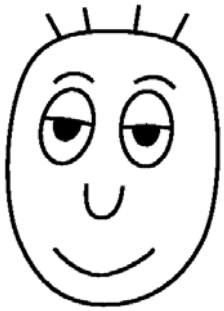


Figure 50 : autocalibrage et reconstruction métrique.

- **Carte de profondeur dense** (figure 51) : à partir des images calibrées de façon précise, il est possible de calculer un grand nombre de corrélations de pixels et il n'est plus nécessaire de se limiter à des corrélations entre deux images. C'est durant cette étape que l'on lève les ambiguïtés dues aux occlusions. A la fin de cette étape, tous les pixels de chaque image ont trouvé au moins un correspondant sur une image. Le programme dispose alors d'une carte de profondeur par caméra.

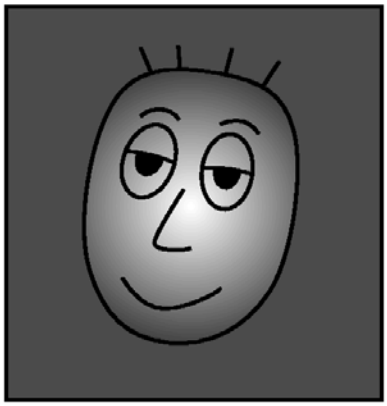


Figure 51 : carte de profondeur dense.

- **Construction d'un model 3d** (figure 52) : la dernière étape de la reconstruction consiste à générer un maillage 3d des points reconstruits. Cette étape commence en général par un algorithme de maillage comme l'algorithme de Delaunay [D34]. Il est ensuite possible d'alléger le maillage en supprimant quelques sommets de façon adéquate [H96, L01], en regroupant des sommets (*Vertex Clustering*) [RB93J, LT97] ou bien en remplaçant par exemple un ensemble de points coplanaires formant un mur par une seule grosse facette [PZB00]. Pour finir, il faut calculer une texture pour chaque primitive générée. A la fin de cette étape, le programme dispose d'un maillage texturé représentant la scène.

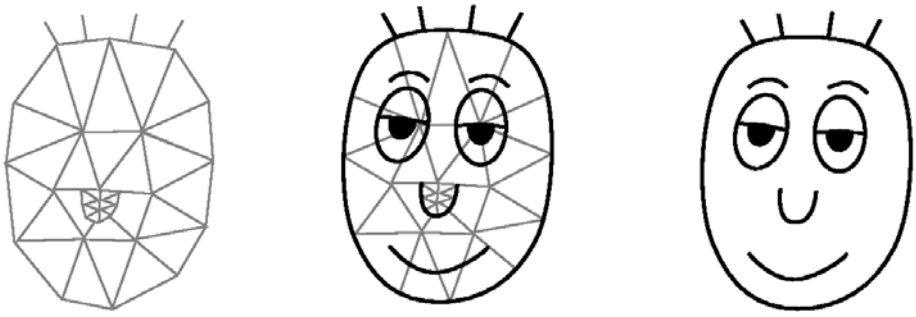


Figure 52 : maillage et placage de textures.

Le rendu s'effectue comme avec n'importe quel maillage, sous OpenGL par exemple. Le rendu peut aussi être réalisé à l'aide de *texture mapped rendering*. Debevec et al. [DTM96] utilisent des textures projetées sur le modèle 3d reconstruit. Les recouvrements de textures sont gérés polygone par polygone en mélangeant les textures de façon pondérée suivant l'emplacement de leur caméra respective. Pour un polygone donné, les coefficients de pondération dépendent de l'angle entre le rayon issu de la caméra virtuelle vers ce polygone et les rayons issus des caméras associées aux textures sélectionnées. Cette méthode de mélange est améliorée dans [DYB98]. D'autres méthodes de reconstruction spécialement dédiées à la reconstruction d'objets architecturaux sont développées dans [CR99] et [LCZ99].

Yu et al. [YDM99] utilisent une reconstruction d'une scène ainsi que les images en entrées pour retrouver les propriétés de réflectance de la surface de chaque objet composant la scène. Ils génèrent de nouvelles vues de la scène avec éventuellement une nouvelle illumination de la scène et de nouveaux objets en images de synthèse. Boivin et Gagalowicz [BG01] effectuent le même genre d'opérations à partir d'une seule image calibrée et d'un modèle 3d de la scène comportant la position des sources lumineuses.

Repko et Pollefeys [RP05] effectuent une reconstruction à partir d'images obtenues en filmant la scène « à la main ». Une sélection parmi toutes les images du film est effectuée afin d'alléger les calculs. Les images sélectionnées sont celles qui proposent le meilleur recouvrement pour un nombre minimal d'images. La reconstruction est renforcée par une correction de distorsion du modèle (*projective drift* en anglais).

Pour plus de détails sur les méthodes de reconstruction à base d'images, le lecteur peut se référer aux travaux de Pollefeys [PVV00, P00, PKV00,PGV04] ainsi qu'à [GSB94].

Pour plus de précision sur les méthodes de maillage à partir de nuages de points, le lecteur peut se référer à la méthode de Delaunay [D34] ainsi qu'aux travaux de Mücke [M95] pour une approche plus pratique.

2.2.8 Bilan

Les méthodes présentées dans cette section permettent de générer en temps réel une nouvelle vue d'une scène. Cependant elles nécessitent toutes un long prétraitement. Ce prétraitement est parfois manuel comme la plupart des méthodes de rendu à partir d'une seule image. Il peut aussi s'agir de rendu partiel comme pour les méthodes de la famille des imposteurs ou le rendu à base d'images n'est utilisé que pour une partie des éléments de la scène.

C'est aussi dans cette section qu'apparaissent les méthodes de rendu à base d'images à proprement parler. Ces méthodes, comme le *Plenoptic Modeling*, *QuickTime VR* ou bien le *Light Field Rendering* n'effectuent aucun calcul concernant la géométrie de la scène. Ils essaient uniquement de rendre compte du flux de rayons lumineux aux alentours de la scène. Plus ces méthodes tentent d'avoir une représentation fine de ce flux lumineux, plus les ressources en mémoire nécessaires pour y parvenir deviennent grandes. Il en va de même pour les techniques d'acquisition du flux lumineux qui nécessitent alors beaucoup de matériel. Cela explique en partie pourquoi ces méthodes sont difficiles à adapter au rendu en temps réel à partir de vidéos. Ces méthodes peuvent tout à fait traiter les scènes ayant des reflets spéculaires et le rendu visuel est souvent de très bonne qualité. Enfin, il faut savoir que ces méthodes, en particulier le *Light Field Rendering* ont inspiré un grand nombre de chercheurs.

Enfin, nous avons vu dans cette section le principe général des méthodes de reconstruction. Nous avons notamment constaté que cette phase de reconstruction requiert beaucoup de calculs, faisant de ces méthodes des techniques difficilement adaptables au rendu de scènes dynamiques. Cependant, bien que la phase de reconstruction soit longue, la phase de rendu peut être très rapide si le nombre de facettes n'est pas trop élevé.

2.3 Rendu à base de vidéos : play-back interactif

Les méthodes abordées dans ce chapitre effectuent non plus un prétraitement sur un ensemble d'images représentant une scène statique mais sur des séquences vidéo. Ces prétraitements sont stockés puis utilisés pendant le visionnage des vidéos pour recréer de nouvelles vues, permettant ainsi de choisir son point de vue de façon interactive. La majorité de ces techniques nécessitent toutefois l'utilisation de matériel puissant et souvent onéreux.

2.3.1 Virtualized Reality™

Virtualized Reality™ est une réalisation innovante introduite par Kanade et al. [KNR95] et développée dans [KRN97, RNK97]. Elle a pour but de reconstruire une scène dynamique sous forme de primitives et de permettre la navigation dans cette scène quel que soit le point de vue. Le dispositif utilisé se compose d'un dôme de 5 mètres de diamètre doté de 51 caméras monochromatiques synchronisées et calibrées avec la méthode de Tsai [T87]. Cette méthode comporte une phase de prétraitement qui consiste à calculer une carte de profondeur par image et d'y associer un model 3d puis une phase d'affichage et de navigation.

Grâce à la synchronisation des caméras, les 51 images obtenues par les caméras correspondent à la scène dynamique au même instant. La fréquence des caméras doit cependant être suffisante pour que l'on puisse considérer que l'acquisition des images de la scène a été effectuée au même instant. Chacun de ces ensembles d'images est traité comme s'il représentait une scène statique. Pour chacune des images, une carte de profondeur est calculée par une MBS (*Multi-baseline Stereo*).

La MBS [OK93] fonctionne sur des caméras partageant le même plan image et ayant la même focale F. Considérons deux caméras vérifiant ces contraintes. Soit d la disparité d'un point de la scène sur les deux images (i.e. la distance en pixels entre 2 points de correspondance), z la profondeur du point considéré et B la distance entre les centres des deux caméras.

On obtient alors la relation :

$$d = BF\frac{1}{z}$$

Avec *n* caméras, cette relation demeure pertinente si l'on la note :

$$\frac{d}{BF} = \frac{1}{z} = const$$

La MBS permet ainsi de traiter plusieurs caméras à la fois pour le calcul de la carte de profondeur d'une des images en bénéficiant de robustesse et de précision. En effet, dans le cas où seules deux caméras sont utilisées, soit elles sont proches l'une de l'autre et le résultat obtenu est robuste mais imprécis (figure 53), soit elles sont plutôt éloignées l'une de l'autre et le résultat est précis mais pas robuste (car les correspondances entre deux points de vue éloignés sont difficiles à calculer). Dans le cas de *n* caméras couvrant l'espace à intervalles réguliers, les résultats obtenus sont précis et robustes.

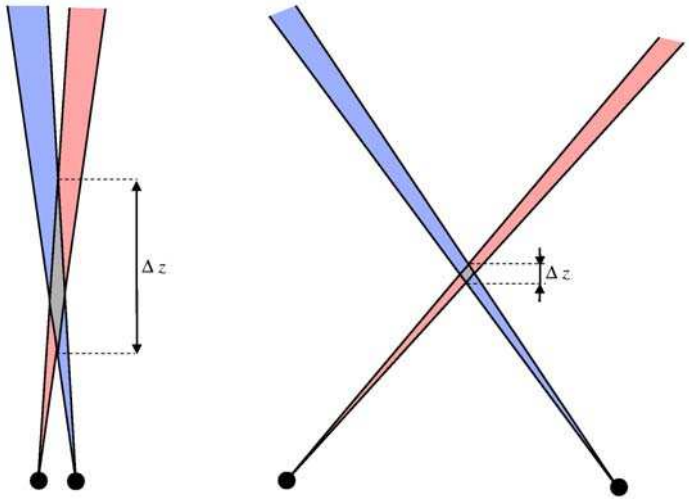


Figure 53 : précision de la triangularisation en fonction de la distance entre les caméras.

Chaque carte de profondeur permet de générer un maillage de triangles représentant la scène traitée. Il suffit pour cela de construire les triangles deux par deux à partir de quatre pixels adjacents et d'affecter à chaque sommet sa profondeur à l'aide de la carte de profondeur et une coordonnée de texture. Le nombre de triangles est ensuite réduit (avec la méthode de Lee et al. [LAM00] par exmple) pour obtenir une dizaine de milliers de triangles par scène reconstruite.

Cette méthode présente toutefois l'inconvénient de supposer que tous les éléments de la scène sont reliés de façon continue et ne gère donc pas les discontinuités (figure 54). Ces discontinuités sont donc détectées et les triangles excédentaires sont supprimés. La scène reconstruite comporte alors des trous qui correspondent aux éléments non visibles à partir de la caméra.

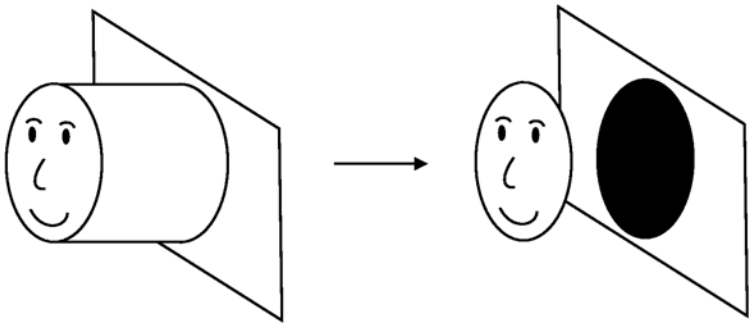


Figure 54 : Virtualized Reality, gestion des discontinuities spatiales.

Pour générer une nouvelle image à partir d'un point de vue donné, une reconstruction de référence est choisie. Cette reconstruction est issue de la carte de profondeur de la caméra la plus proche du point de vue traité. Ce choix permet de minimiser le nombre de trous dans l'image générée. Pour améliorer le rendu, les trous sont détectés sur l'image générée puis éventuellement comblés par des reconstructions issues de caméras voisines. Le rendu supplémentaire n'est évidemment effectué que sur les trous. Il se peut malgré tout que les trous ne soient pas totalement comblés après cette étape.

Virtualized Reality^{TM} est donc une méthode permettant de générer de nouvelles images d'une scène dynamique en différé. Cette méthode nécessite un nombre conséquent de caméras (une cinquantaine) et un dispositif particulier comme par exemple un dôme pour disposer les caméras de façon adéquate.

2.3.2 Layered Representation

Zitnick et al. [ZBU04] présentent une méthode interactive de rendu de scènes dynamiques en play-back en utilisant des *sprites*. Cette méthode, inspirée des travaux de Shade et al. [SGH98], consiste à calculer une carte de disparité et à générer des *sprites* pour représenter les différents objets de la scène. Les bords des *sprites* susceptibles de se trouver sur une discontinuité de la carte de disparité se voient attribuer deux disparités afin d'améliorer la qualité visuelle du rendu. Cette méthode marque un tournant dans l'évolution des méthodes de rendu à base d'images tant le rendu est fluide et de bonne qualité. Cette méthode est toutefois très gourmande en prétraitements et nécessite au minimum une dizaine de caméras.

Zitnick et al. effectuent l'acquisition des images à l'aide d'une dizaine de caméras haute résolution (1024x768) disposées en arc de cercle (d'environ 30°). Deux unités de concentration (*concentrator units* en anglais) se partagent les caméras afin de synchroniser les images et de les stocker sur des disques durs. Chaque caméra est alors calibrée avec la méthode de Zhang [Z00].

Une carte de disparité est calculée pour chaque image à l'aide d'une méthode de segmentation. Cette méthode découpe l'image en régions ayant les mêmes caractéristiques colorimétriques et donc sensées représenter une variation lente de la disparité (figure 57). Un masque de convolution de flou est préalablement appliqué sur les images afin d'obtenir une segmentation plus robuste. Les régions obtenues sont alors appareillées par une DSD (*Disparity Space Distribution*). Cette étape consiste à projeter itérativement sur une image I_2 les régions détectées sur une image I_1. Le rapport des histogrammes du projeté de I_1 sur I_2 et de I_2 est alors calculé. Deux régions sont appareillées si ce rapport vaut 1 pour la couleur de

la région traitée et si ce rapport est proche de 0 ou de l'infini pour les autres couleurs. La carte de disparité est alors optimisée en tenant compte pour chaque région de la disparité des régions voisines. Une disparité est alors attribuée à chaque pixel en acceptant une légère variation par rapport à la disparité de la région d'origine afin de mieux faire le lien entre régions (figure 58). Une des clés de la méthode consiste alors à attribuer deux disparités aux pixels proches d'une discontinuité dans la carte de disparité (cf. figure 55). Enfin, les images ainsi que leur carte de disparité sont compressées afin de pouvoir les stocker et surtout afin de pouvoir supporter les accès disques lors du rendu en temps réel.

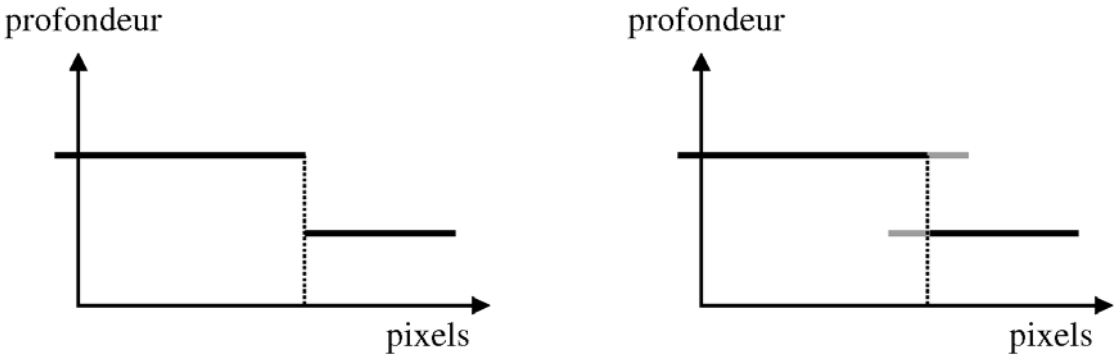


Figure 55 : Layered Representation, gestion des discontinuités spatiales.

Le rendu commence par le placement de la caméra virtuelle dans la scène. Les deux caméras cam_i et cam_{i+1} les plus proches de cette caméra virtuelle sont utilisées pour le rendu. Chacune des deux caméras effectue d'une part un rendu principal concernant tous le pixels de l'image puis un rendu des pixels se trouvant sur une discontinuité de la carte de disparité et ayant donc deux disparités. Comme illustré sur la figure 56, les quatre images obtenues sont alors mélangées à l'aide de *fragment shaders*.

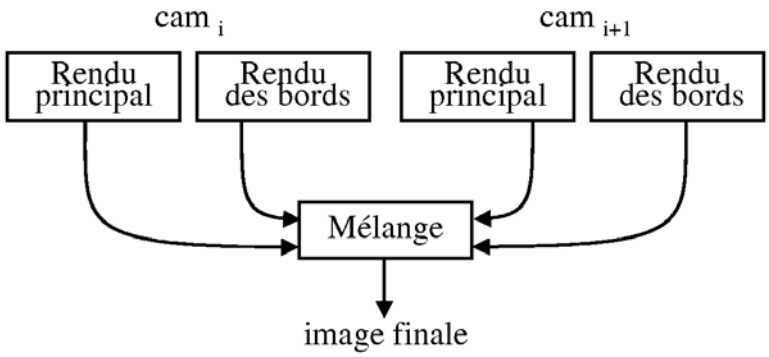
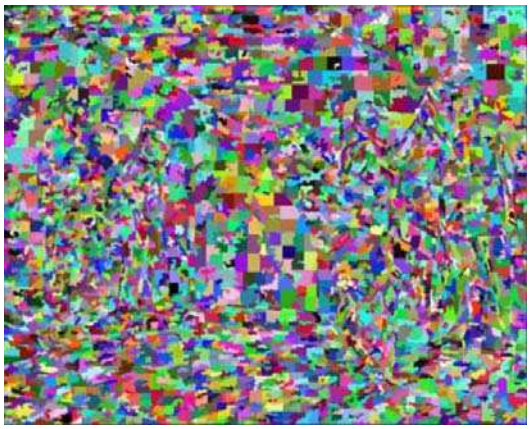


Figure 56 : Layered Representation, hiérarchie des traitements.

Cette méthode permet alors de se déplacer dans la scène dynamique de façon fluide avec un rendu d'une qualité visuelle remarquable. Les figure 57 et figure 58 montrent une carte de disparité construite avec la méthode de segmentation. La figure 59 montre un exemple de rendu avec cette méthode. Le rendu est plus saisissant sur une vidéo.



Avec l'aimable autorisation de Larry Zitnick [ZBU04]

Figure 57 : Layered Representation, segmentation des couleurs.

Avec l'aimable autorisation de Larry Zitnick [ZBU04]

Figure 58 : Layered Representation, carte de disparité générée.

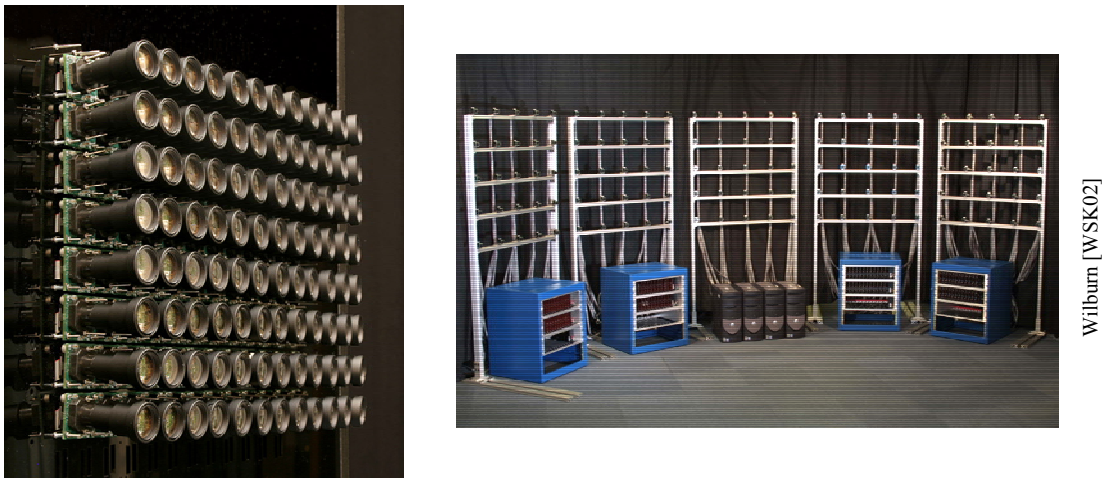
Avec l'aimable autorisation de Larry Zitnick [ZBU04]

Figure 59 : Layered Representation, image générée.

2.3.3 Murs de caméras

On désigne par murs de caméras (de l'anglais *camera array*) un dispositif constitué d'un grand nombre de caméras, souvent de premiers prix. L'idée de base consiste à dire que ce qui est fabriqué et vendu à grande échelle est souvent bon marché, donc plutôt que d'acheter quelques très bonnes caméras, il peut être préférable d'en acheter beaucoup plus mais de moins bonne qualité.

Le principal défi de ce genre de dispositif est la façon de gérer l'énorme flux de données transitant des caméras vers la ou les machines de traitement. Ceci explique pourquoi les premiers murs de caméras se limitaient à une cinquantaine de caméras.



Wilburn [WSK02]

Figure 60 : mur de caméras (Stanford Camera Array)

Le plus connu des murs de caméras est probablement le *Stanford Camera Array* (figure 60), il peut supporter plus d'une centaine de caméras. Les caméras utilisées et notamment leur focale varient selon que la scène soit intérieure ou extérieure, la résolution reste la même : 640×480. Chaque caméra est connectée à un FPGA (carte programmable) qui se charge de générer des fichiers RAW ou de la compression en MPEG2. Le FPGA sert aussi de routeur afin de transmettre les signaux de synchronisation aux caméras voisines. Les FPGA sont reliés entre eux sous forme d'arbres binaires et la caméra racine est connectée au PC qui les contrôle. Si le nombre de caméras devient trop élevé, il suffit d'ajouter un PC supplémentaire qui gérera son propre groupe de caméras. Ainsi, une centaine de caméras nécessitent 4 PC pouvant gérer un débit de 30 fps. Pour plus de détails techniques sur ce dispositif, le lecteur peut se référer à Wilburn et al. [WSK02].

Comme nous le montrent Wilburn et al. [WJ05], il existe une multitude d'applications aux murs de caméras. Pour la plupart de ces applications, il s'agit de simuler un système optique avec un seul centre de projection à partir de toutes les caméras en considérant que les objets étudiés ont des dimensions petites par rapport à la distance qui les sépare des caméras. Les murs de caméras permettent de générer des images haute résolution construites sous forme de mosaïques. Wilburn et al. obtiennent une image 3800×2000 à partir d'une centaine de caméras 640×480. Il est possible d'effectuer sur l'image obtenue une rectification dynamique des couleurs (de l'anglais *high dynamic range*) en paramétrant différemment les caméras adjacentes. Ce procédé nécessite l'emploi d'un calibrage des couleurs décrit dans [JW05] de Joshi et al..

Une autre possibilité proposée par les murs de caméras consiste à faire des vidéos à framerate très élevé. Il suffit pour cela de désynchroniser les caméras de façon adéquate. Cette méthode ne fonctionne que si les objets traités sont loin des caméras ce qui permet de rectifier les images de chaque caméra par une homographie tout en supposant négligeable la déformation perspective engendrée. A partir de 52 caméras, Wilburn et al. obtiennent un débit de 1560 images par seconde.

Les murs de caméras permettent aussi de simuler une caméra à ouverture variable. L'utilisateur choisit un plan de netteté sur lequel il projette les images voisines de la vue à générer, ce qui a pour effet de simuler une très grande ouverture de caméra (et par conséquent de diminuer la profondeur de champ). En pratique, la projection des images du mur de caméras sur un plan revient à les décaler les unes par rapport aux autres et à en faire

la moyenne. Il suffit ensuite de faire un décalage approprié pour modifier la position du plan focal. Dans le cas du mur de caméras de Stanford, le décalage est effectué par les FPGA ce qui permet de produire une simulation d'ouverture de caméras en temps réel sur des scènes dynamiques en attribuant 15 caméras par PC. Un diaphragme suffisamment ouvert a un effet de désocclusion. Ce phénomène est illustré par la figure 61 et la figure 62 sur lesquelles on peut voir une scène photographiée avec un diaphragme fermé et la même scène prise avec un diaphragme ouvert. Dans le second cas, la mise au point sur un objet en arrière plan peut flouter ses occluants du premier plan.

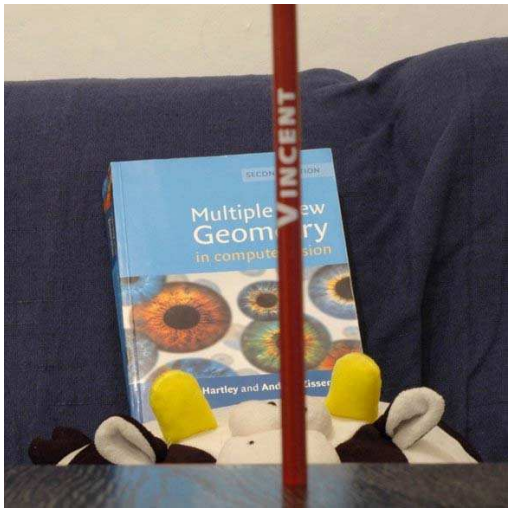


Figure 61 : image priseavec une ouverture du diaphragme très faible.

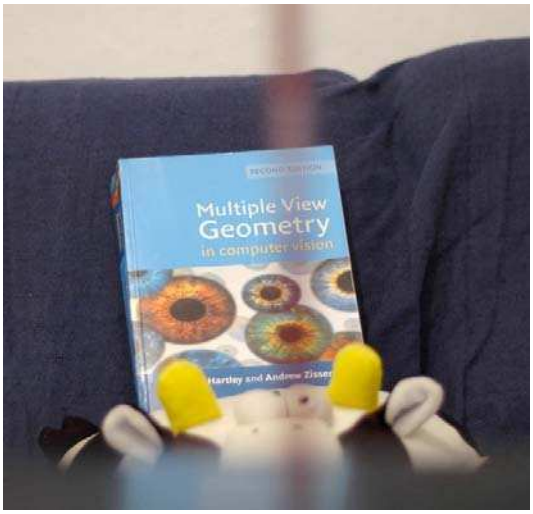


Figure 62 : image priseavec une ouverture du diaphragme très grande.

Vaish et al. [VG05] utilisent ce principe d'ouverture extrême pour effectuer de la « désocclusion » d'objet ce qui permet de voir un sujet caché par des occlusions. Wilburn et al. [WJ05] optimisent cette méthode en supprimant sur les images en entrée la contribution des pixels concernés par une occlusion. L'exemple proposé montre deux personnes en train de discuter derrière un buisson (figure 63), une seule image ne permet pas de les distinguer. En réglant le plan focal sur eux, les images obtenues montrent clairement leurs mouvements. Le résultat est saisissant.

Enfin, les murs de caméras permettent de générer des nouvelles vues d'une scène. Wilburn et al. [WJ05] utilisent une méthode à base de flux optique (cf. chapitre 2.1.1.4). Cette méthode nécessite la spécification des plans *near* et *far* délimitant la scène. Un taux d'échantillonnage temporel est calculé en fonction des caractéristiques de la scène et du positionnement relatif des caméras. La méthode de flux optique proposée par Black et Anandan [BA93] est alors appliquée sur les quatre images issues des caméras les plus proche de la caméra virtuelle. Une carte de flux optique est alors calculée à partir de ces quatre images puis appliquée sur ces mêmes images afin de les mélanger et de générer la nouvelle vue. Les caméras sont calibrées avec la méthode de Vaish et al. [VWJ04]. Les résultats montrent que les grands mouvements des objets de la scène masquent les mouvements d'amplitude plus faible. De plus, cette méthode ne gère pas les occlusions mais il faut noter que d'autres méthodes de rendu à base d'images peuvent être utilisées avec un mur de caméras de ce genre comme par exemple la méthode de Zitnick et al. [ZBU04].

Wilburn [WSK02]

Figure 63 : simulation d'ouverture du diaphragme.
L'image de droite représente une des images de départs, on ne distingue rien derrière le buisson. L'image de gauche correspond à l'image générée avec une mise au point sur les personnes derrière le buisson.

Yang et al. [YEB02] utilisent pour leur *distributed light field camera* un ensemble de 8x8 caméras de type webcam (voir chapitre 2.4.2). Les caméras utilisées ne permettent pas de synchronisation et il n'y a pas de calibrage des couleurs. Zhang et Chen [ZC04] ne synchronisent pas non plus leurs 48 caméras ce qui provoque des artéfacts dans les vues générées, notamment sur les contours des objets mobiles.

2.3.4 Light Fields et Lumigraph

Goldlücke et al. [GMW02] étendent le principe de Light Field aux scènes dynamiques. Le principe de base reste inchangé puisqu'il s'agit de reproduire à partir d'une centaine de caméras une représentation des rayons lumineux à travers un espace. Le dispositif proposé est constitué d'un mur de caméras calibrées. Dans un premier temps, l'utilisateur effectue l'acquisition des données et après un pré-traitement, il peut passer d'une vue à l'autre au fil de la vidéo.

Durant le pré-traitement et pour un instant donné, une carte de disparité est calculée pour chaque image source. Cette carte de disparité est calculée par une méthode du type *Sum of Absolute Difference*, mais optimisée. Une fois ces disparités calculées, chaque image source est recouverte d'un maillage donc chaque sommet se voit attribuer une profondeur correspondant à la disparité moyenne des pixels avoisinants. Il en résulte une carte de disparité moins précise mais plus légère en espace mémoire.

Soit m le nombre d'images sources, la création d'une nouvelle vue utilise une méthode en m passes en utilisant tour à tour chacune de ces images sources. Pour une image source I_k, chacun des triangles du maillage précédemment calculé est projeté sur l'image virtuelle et se voit pondéré par un score w_k associé à l'image. Ce score w_k est une gaussienne du type $w_k = e^{-c.d^2}$ où d est la distance entre la caméra k et la caméra virtuelle. De ce fait, plus une caméra est loin de la caméra virtuelle, moins elle a d'influence sur la création de la nouvelle vue. Pour un score $w_k < \varepsilon$, l'image k n'est pas prise en compte. La constante c peut varier selon les configurations de caméras. En effet, cette constante est définie de telle sorte que si la caméra virtuelle correspond à une caméra du dispositif, alors les caméras voisines auront toutes un score inférieur au seuil ε de telle sorte que l'image créée ne soit pas altérée par une contribution erronée provenant des autres images.

Une fois tous les scores calculés, le poids final de la contribution d'une image est donné par la formule :

$$W_k = \frac{w_k}{\sum_{i<m} w_i}$$

Cette méthode utilise l'accélération graphique et notamment les *shaders programs* pour les fonctions de mélange. Le rendu est temps réel mais ne permet pas une navigation trop éloignée des caméras, elles-mêmes assez proches les unes des autres. En outre, des artefacts peuvent apparaître sur les bords des objets, lors d'une discontinuité de la carte de disparité.

Wang et Yang [WY05] s'attaquent au problème de la synchronisation des caméras et proposent une synchronisation artificielle en interpolant au même instant des images non-synchronisées. Une fois l'acquisition des images effectuée, le flux optique temporel (entre deux images d'une même caméra) est calculé pour chaque caméra. Certaines erreurs concernant ce flux optique sont supprimées en calculant un flux optique spatial (au même instant, entre différentes caméras). Des correspondances de points sont alors établies entre deux images successives provenant d'une même caméra. Ces correspondances doivent vérifier les contraintes épipolaires entre les différentes caméras. A partir de ces correspondances, les bords des objets de la scène sont détectés. Les correspondances de bords ainsi établies permettent de générer des images intermédiaires par morphing en utilisant la méthode de Beier et Neely [BN92]. Les images intermédiaires sont calculées pour chaque caméra de telle sorte qu'elles soient synchronisées, ce qui permet ensuite d'effectuer un rendu en utilisant une méthode de *Light Field Rendering* classique (en l'occurrence la méthode de Buehler et al. [BBM01]). Après le prétraitement, cette méthode permet un rendu en temps interactif.

2.3.5 Bilan

Cette section décrit les méthodes de rendu à base de vidéos en différé. La totalité de ces méthodes utilisent soit un grand nombre de caméras, soit un nombre de caméras plus réduit mais à grande résolution. Un des problèmes majeur consiste alors à gérer le flux vidéo en direct. Ensuite, il se trouve que toutes ces méthodes nécessitent un prétraitement sur les images. Ce prétraitement est long et ne peut être effectué en temps réel. Notons aussi qu'un prétraitement sur des vidéos implique qu'il est nécessaire de les stocker ce qui requiert une quantité de mémoire non négligeable.

Parmi les méthodes détaillées dans cette section, la méthode de *Virtualized Reality* a été la première proposée et a eu un impact assez fort, même si la qualité visuelle des images générées n'était pas excellente.

Le mur de caméras de Stanford est lui aussi très impressionnant, tant du point de vue de la quantité de matériel mis en œuvre qu'au point de vue des résultats obtenus. L'avantage de cette méthode est d'être performante sur d'autres applications que le rendu à base de vidéos. C'est le cas en particulier pour la rectification dynamique des couleurs et pour la simulation de profondeur de champ.

Enfin, la méthode des *Layered representation* marque un tournant dans les méthodes de rendu à base de vidéos tant la qualité du rendu et la fluidité de la navigation sont bonnes. Il

faut toutefois noter que, comme les autres méthodes, elle nécessite beaucoup de matériel de haute technologie.

2.4 Rendu à base de vidéos en direct

Dans ce dernier chapitre de l'état de l'art, nous présentons les méthodes permettant de générer à la volée de nouvelles images d'une scène filmée par plusieurs caméras. La contrepartie de la vitesse d'exécution se voit en général sur la taille et sur la qualité des images générées. En effet, rares sont les méthodes de rendu à base de vidéos en direct capables de traiter en temps réel des images d'une résolution supérieure à celle d'un écran de télévision. La plupart de ces méthodes utilisent abondamment les possibilités des cartes graphiques en vue d'une part d'accélérer les traitements et d'autre part de libérer le CPU pour d'autres taches telles que le traitement des flux vidéo ou bien la gestion des paramètres de la caméra virtuelle.

Nous commencerons par le projet « 3D TV » constituant le premier dispositif de rendu stéréoscopique en direct. Cette méthode requiert une quantité impressionnante de matériel puissant. Nous continuerons ensuite avec des méthodes beaucoup moins onéreuses et souvent moins impressionnantes issues des techniques de *Light Fields* et de *Lumigraph*. Puis nous décrirons les différentes variantes de *Visual Hulls*, permettant de représenter l'objet principal d'une scène sous des points de vue très éloignés les uns des autres en utilisant un nombre restreint de caméras. Nous finirons par une introduction à la méthode des *Plane Sweep*.

2.4.1 « 3D TV »

Le système 3D TV [MP04] mis en œuvre par Matusik et Pfister est une sorte de *Light Field* grandeur nature. Ce procédé consiste à filmer une scène avec un grand nombre de caméras et à restituer à l'observateur les images provenant des caméras adéquates à l'aide d'un écran lenticulaire (voir annexe 2).

L'acquisition s'effectue à partir de 16 caméras haute définition (1300×1030) alignées sur un banc d'environ deux mètres. Compte tenu de la difficulté à aligner correctement les caméras, celles-ci sont calibrées [Z00] et alignées par rectification (une simple homographie sur les images). Les 16 caméras sont connectées via un port firewire à 8 PC Pentium4 3 GHz dotés de 2 GB de RAM. Ces 8 PC compressent les vidéos et les transmettent par broadcast à 8 autres PC situés dans la salle de rendu.

Le rendu s'effectue grâce à un réseau lenticulaire illuminé par 16 vidéo-projecteurs alignés au mieux de façon à représenter de façon optimale les 16 caméras. Lorsque l'observateur se déplace devant l'écran, chaque réseau lenticulaire se charge de répartir le rayon incident vers la bonne caméra. C'est de ce point de vue que ce système ressemble à un light field grandeur nature. Ainsi, les deux yeux de l'observateur ne sont pas traités de la même façon ce qui génère l'effet de rendu stéréoscopique. Pour plus de détails sur les réseaux lenticulaires, le lecteur peut se reporter à l'annexe 3.

Un dernier PC relié à une caméra s'occupe de calibrer les vidéo-projecteurs entre eux, tant au niveau du cadrage qu'au niveau de l'illumination. Ce calibrage s'effectue automatiquement grâce aux méthodes de Raskar et al. [RBY99] et de Li et al. [LCY00]. Les intensités lumineuses de chaque vidéo-projecteur sont uniformisées entre elles et chaque

vidéo-projecteur se voit attribué un masque d'intensité pour uniformiser l'intensité lumineuse sur l'ensemble de l'image.

Le maillon faible de cette chaîne se situe au niveau du transfert des images haute définition des caméras vers les PC via le port firewire ce qui limite le framerate à 12 images par seconde.

Dans leur système de rendu, Matusik et Pfister utilisent un écran de 1,80 m sur 1,20 m, le rendu est fluide mais le passage d'une caméra à l'autre se fait sentir par un saut sur l'image. De plus, les auteurs conseillent de ne pas s'approcher à moins de 3 mètres de l'écran pour éviter des problèmes d'aliasing. Dans ces conditions, le champs de vision est limité à 30° du fait de l'utilisation de réseaux lenticulaires.

2.4.2 Light Fields dynamiques, Lumigraphs dynamiques

Le premier pas vers la gestion de scènes dynamiques par Light Field a été introduit par Li et al. [LKH98]. Ils proposent de décrire une scène à l'aide de deux Light Fields, le premier permettant de traiter les composantes ambiantes de la scène et le second les composantes diffuses dépendantes des sources de lumière. Le rendu s'effectue en combinant ces deux Light Fields dont la compression est raisonnable compte tenu de leur similarité. Cette méthode nécessite toutefois l'utilisation de *Depth Field*, sorte de carte de profondeur, afin de traiter correctement les effets diffus. De plus, l'emploi de ces *Depth Field* contraint à n'utiliser que des objets rigides.

Yang et al. [YEB02] présentent un système utilisant 64 caméras disposées sous forme de quadrillage de 8x8 caméras. Afin de ne pas saturer la bande passante entre les caméras et la machine responsable du rendu, des PC servant d'intermédiaires ne transmettent que les portions d'images utiles à la création de la nouvelle vue traitée (voir figure 64). Les informations concernant les portions d'images à sélectionner sont transmises à chaque PC intermédiaire par broadcast ce qui permet de ne pas accaparer trop de bande passante supplémentaire. Un broadcast est suffisant car seules les informations concernant la vue à générer sont transmises. Le rendu est effectué par *projective texture mapping* sur un plan focal. De ce fait, tous les objets de la scène qui ne sont pas situés sur ce plan focal sont dédoublés ce qui produit un effet similaire à un flou de profondeur de champ. L'intérêt principal de cette méthode est de pouvoir gérer plusieurs utilisateurs simultanément. La bande passante utilisée entre les caméras et les diverses machines est proportionnelle au nombre d'utilisateurs et non pas au nombre de caméras.

Schirmacher et al. [SMS01] étendent les possibilités du Lumigraph au traitement en direct des scènes dynamiques. Le dispositif proposé est composé de 6 caméras reliées par deux à trois PC. Dans la même idée d'optimisation du nombre de données à traiter que Yang et al. [YEB02], les PC intermédiaires ne communiquent à la machine chargée du rendu que les données nécessaires au rendu de la nouvelle image. Cependant, dans ce cas, ce sont uniquement des parcelles d'images qui sont transmises. Les images, groupées par deux, sont rectifiées à partir de leur propriétés epipôlaires afin d'accélérer un processus de comparaison de blocs de pixels. Une carte de disparité est ainsi obtenue en temps réel mais au détriment d'une bonne qualité. La méthode de rendu est inspirée du Lumigraph : une fois la caméra virtuelle placée, comme dans un lancer de rayons, chaque rayon passant par le centre de la caméra et par le pixel traité est identifié. Les trois rayons les plus proches de ce rayon et provenant de trois caméras différentes parmi les six utilisées sont sélectionnés. Ces trois rayons définissent un triangle sur l'image finale. En combinant chacun des triangles de

l'image finale ainsi définis et les cartes de disparité, il est possible de déterminer les zones des images de départ concernées par chaque triangle. Seules ces zones sont transmises à la machine chargée du rendu afin d'alléger le flux de données sur le réseau. Finalement, pour calculer la couleur d'un pixel de l'image finale, il suffit de trouver dans quel triangle ce pixel est inscrit. Les trois images de départ associées à ce triangle vont contribuer au choix de la couleur de ce pixel. Il s'agit alors de pondérer la contribution de ces trois images selon leur proximité avec le rayon passant par le pixel et le centre de la caméra virtuelle. Cette méthode atteint un framerate interactif mais les images générées sont extrêmement bruitées.

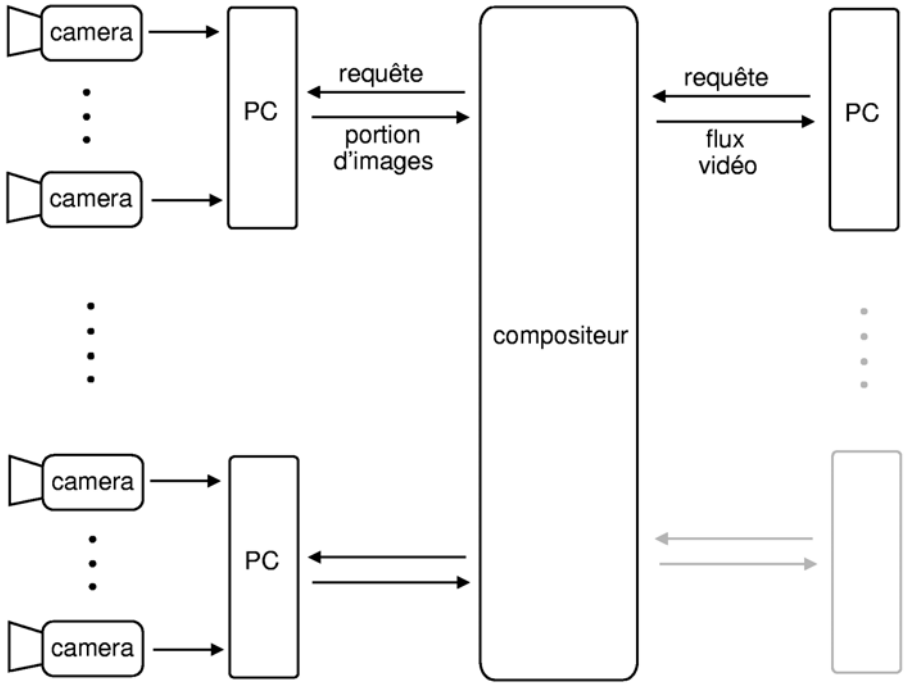


Figure 64 : Distributed Light Field, système de client/serveur.

2.4.3 Visual Hulls

Le principe général des *Visual Hulls* consiste à filmer un objet principal avec plusieurs caméras et d'isoler sur ces caméras la silhouette de cet objet, par extraction de l'arrière plan par exemple. On calcule à partir de chaque caméra le pseudo-cône délimité par la silhouette de l'image émanant du centre de la caméra. Ce cône réduit la proportion de l'espace où l'objet en question est susceptible de se trouver et spécifie plus particulièrement les endroits où il ne se trouve pas. Si cette même opération est effectuée sur toutes les caméras, il est alors possible d'approximer la forme de l'objet en calculant l'intersection de tous les cônes (figure 65). Il faut toutefois noter que même en faisant tendre le nombre de caméras utilisées vers l'infini, on ne retrouve pas nécessairement la forme de l'objet car cette méthode ne tient pas compte les zones concaves. On comprend facilement qu'il est possible et conseillé de disposer les caméras tout autours de la scène plutôt que de les regrouper ce qui distingue cette méthode parmi toutes les autres.

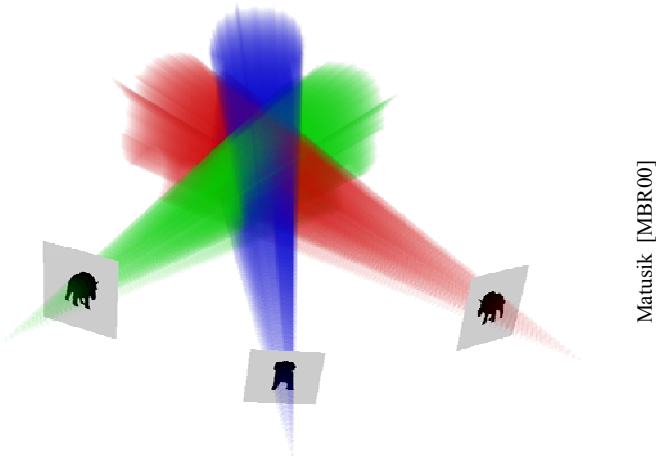


Figure 65 : Visual Hulls, principe général.

2.4.3.1 Volumetric Visual Hulls

Les premières méthodes basées sur le principe des *Visual Hulls* sont les méthodes dites de *Volumetric Visual Hulls*. La scène est décomposée en voxels et seuls les voxels dont le projeté sur chaque image de départ se situe dans une silhouette sont gardés. Un grand nombre d'articles ont étés publiés sur cette approche, notamment les travaux de Potmesil [P87], Kanade et al. [KSV98] et Borovikov et Davis [BD00].

La méthode de Theobalt et al. [TLM03] est particulièrement caractéristique des *Volumetric Visual Hulls*. Une série de PC gèrent les caméras deux à deux. Leur tâche consiste d'une part à extraire la silhouette de l'objet principal sur les deux images gérées et d'autre part à calculer un modèle grossier de voxels à partir des deux silhouettes obtenues. Tous ces PC sont reliés par réseau à un PC central chargé de calculer une reconstruction plus fine et du rendu final. La reconstruction se base sur les modèles grossiers précalculés ainsi que sur les textures des objets. Il est possible d'insérer d'autres PC « nœuds » chargés de combiner les reconstructions grossières de quelques caméras pour faciliter le travail du PC principal afin par exemple de gérer un nombre important de caméras, ingérable avec un seul ordinateur principal. Cette méthode ne génère qu'un ensemble de voxels dépourvus de couleur et de texture. Le passage de la représentation par voxels à la représentation par triangles n'étant pas accessible en temps réel, Wurmlin et al. [WLS02] proposent d'ajouter de la couleur en utilisant une représentation par points. Goldlücke et Magnor [GM03] s'inspirent des *Microfacet billboarding* de Yamazaki et al. [YSK02] en associant un *billboard* à chaque voxel. Pour mieux gérer le bord des objets, les *billboards* situés sur le contour de la silhouette sont à moitié transparents. Les couleurs des *billboards* sont calculées à partir des couleurs extraites de deux caméras choisies suivant l'angle formé par les rayons passant par le *billboard* et les centres de la caméra virtuelle et de la caméra de référence. Plus cet angle est petit et plus la contribution colorimétrique de la caméra est significative.

2.4.3.2 Polyhedral Visual Hulls

La méthode de *volumetric visual hulls* décrite dans la section précédente ne permet pas de générer de maillage en temps réel à partir de l'ensemble de voxels généré. Les méthodes dites de Polyhedral Visual Hulls permettent de générer directement des triangles sans passer par un calcul de voxels. Pour cela, il suffit de définir une silhouette comme un polygone.

Ensuite, les intersections des cônes des silhouettes peuvent être calculées avec un algorithme du type CSG (*Constructive Solid Geometry*) qui correspond à une combinaison booléenne d'éléments. Parmi les travaux majeurs effectués sur les *polyhedral visual hulls*, nous pouvons citer ceux de Lazebnik et al. [LBP01], ceux de Matusik et al. [MBM01] ainsi que ceux de Franco et Boyer [FB03].

Pour une implémentation facile, Li et al. [LMS03] effectuent l'extraction de silhouette ainsi que sa modélisation par polygones à l'aide de la bibliothèque OpenCV [I00]. Le calcul des intersections polyhédrales est réalisé à la volée par la bibliothèque BREP [netBREP05], nouvellement renommée *GNU Triangle Surface Library* (GTS). Li et al. [LMS03] insistent sur l'importance d'un calibrage de bonne qualité et montrent que dans leurs tests aboutissant à la construction d'un modèle constitué de 500 facettes, le temps utilisé pour l'extraction d'une silhouette, son approximation par un polygone et le calcul de l'intersection polyhédrale, est inférieur au temps nécessaire à l'acquisition synchronisée d'une image pour un ensemble de caméras.

Ces opérations aboutissent à la création d'un maillage qu'il faut texturer. La solution la plus utilisée consiste à choisir la caméra faisant le plus face à la facette traitée. Le mode d'application de texture est toutefois délicat. En effet, l'utilisation de texture projetée ne permet pas de gérer les occlusions. Debevec et al. [DBY98] effectuent un prétraitement de fenêtrage (*clipping*) sur les facettes avant d'effectuer une projection de texture. Sawhney et al. [SAK02] ainsi que Li et al. [LMS03] utilisent plutôt une méthode du type *shadow-mapping* (une passe de rendu par caméra).

Li et al. [LSM02] combinent une méthode de *polyhedral visual hulls* avec une méthode de calcul de correspondances stéréo. Ils composent des groupes de deux caméras positionnées assez près l'une de l'autre (sur un arc de cercle de 5° environ) et chaque groupe est positionné assez loin des autres groupes (sur un arc de cercle de 90° environ). A chaque groupe de deux caméras est associé un PC qui gère l'acquisition vidéo, l'extraction de la silhouette et la création du polygone la représentant. Ces informations sont envoyées au PC central qui renvoie aux PC associés aux caméras l'intersection polyhédrale calculée. En effet, ces derniers sont aussi chargés de calculer une carte de disparité en se servant de l'intersection polyhédrale pour accélérer les calculs et limiter les erreurs d'appariement. La carte de disparité est calculée avec une simple SAD (*Sum of Absolute Difference*) appliquée sur des blocs de pixels (voir chapitre 2.1.1.1). Pour chaque facette du modèle 3d de l'objet principal de la scène, le groupe de caméras considéré comme le plus « en face » de la facette est sélectionné. La facette est alors texturée par *warping* 3d entre les deux caméras du groupe sélectionné en accord avec la carte de disparité préalablement établie. L'utilisation d'une méthode de calcul de cartes de disparité produit néanmoins une grosse chute du framerate.

2.4.3.3 Image Based Visual Hulls

Comme son nom l'indique, la méthode *Image Based Visual Hulls* n'utilise ni voxel, ni polyhèdre. Le rendu s'effectue directement dans l'espace de l'image et bénéficie d'une optimisation due à une bonne exploitation de la géométrie épipolaire. L'algorithme de référence que nous allons détailler est proposé par Matusik et al. [MBR00].

Comme pour les méthodes précédentes, la première étape de cet algorithme consiste à extraire la silhouette de l'objet principal de la scène et de l'approximer avec un polygone. Ensuite, à la manière d'un lancer de rayons, chaque pixel de l'image finale est traité

séparément. Pour chaque pixel de l'image finale, on calcule le rayon R_f associé et on le projette sur les images de référence (figure 66). Ces rayons projetés correspondent aux droites épipolaires associées au point de l'image finale traitée. Sur chaque droite épipolaire, seuls sont gardés les segments constituant l'intersection de la droite avec la silhouette. Ces segments sont alors reprojetés sur le rayon R_f. Les portions de segment de R_f n'étant pas recouvertes par la totalité de ces segments projetés sont supprimées. Finalement, le point de R_f sélectionné est l'extrémité du segment le plus proche du centre de la caméra virtuelle.

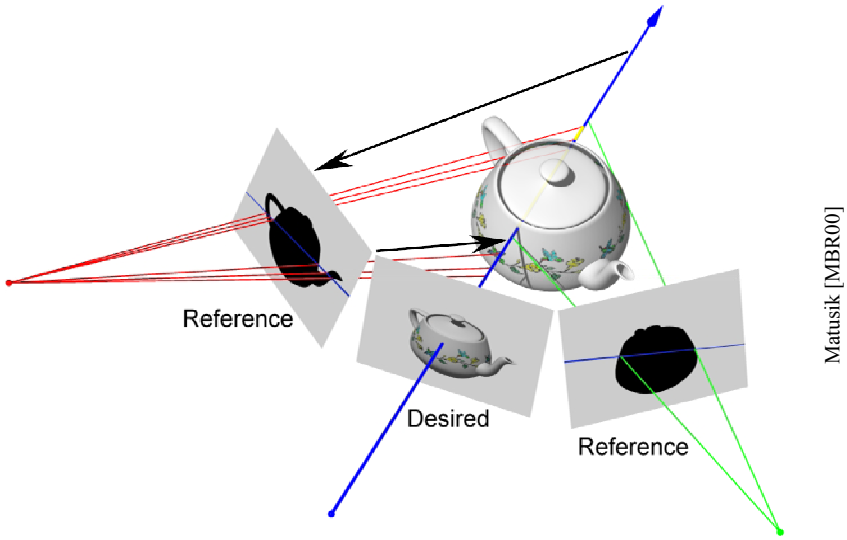


Figure 66 : Image Based Visual Hulls,
creation d'une nouvelle image.

Reste encore à attribuer une couleur à ce point. Pour cela, la caméra dont le rayon passant par ce point forme un angle minimal avec R_f est choisie. La couleur du projeté du point sur cette caméra est une couleur finale potentielle. Il faut toutefois vérifier qu'aucune occlusion n'intervienne sur ce pixel. Pour régler ce problème de visibilité, on parcourt la ligne du pixel traité sur l'image finale (figure 67). On effectue le parcours de droite à gauche ou de gauche à droite de telle sorte que ce parcours corresponde à un parcours du premier plan vers l'arrière plan sur l'image de référence traitée. Pendant la traversée, on traite chaque rayon associé au pixel rencontré et on accumule les intervalles de silhouette intersectés par ces rayons. Lors du traitement du point de R_f précédemment sélectionné, on vérifie que son projeté sur les segments accumulés ne recouvre pas un segment précédemment projeté. Dans le cas d'un recouvrement, ce point est occlus et il faut donc choisir une autre caméra de référence pour déterminer la couleur du pixel de l'image finale traitée.

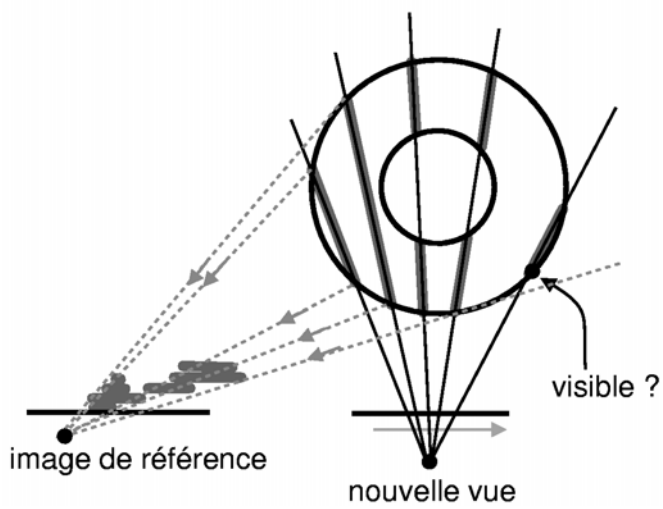


Figure 67 : Image Based Visual Hulls, problème de visibilité.

Pour accélérer le calcul d'intersection entre une silhouette et une droite épipolaire, les images de référence sont découpées en zones définies par les droites passant par les sommets des segments constituant la silhouette et par l'épipôle (figure 68). A chaque zone on associe les segments de silhouette présents dans la zone. Pour une droite épipolaire donnée, il suffit de trouver dans quelle zone elle se situe pour avoir accès à tous les segments qu'elle intersecte. Le tri de ces zones n'est pas très long, cependant il peut être évité en effectuant une rectification épipolaire (droites épipolaires horizontales) sur l'image de référence. Néanmoins lors de la rectification, il est courant de subir une perte de qualité de l'image. De plus, il existe des cas de dégénérescence lorsque l'épipôle d'une image de référence se trouve dans le champ de vision de l'image finale. L'utilisation de ces zones prend toute sa signification lorsque l'on considère le nombre d'intersections qu'il est nécessaire de calculer entre des droites épipolaires et les segments constituant la silhouette.

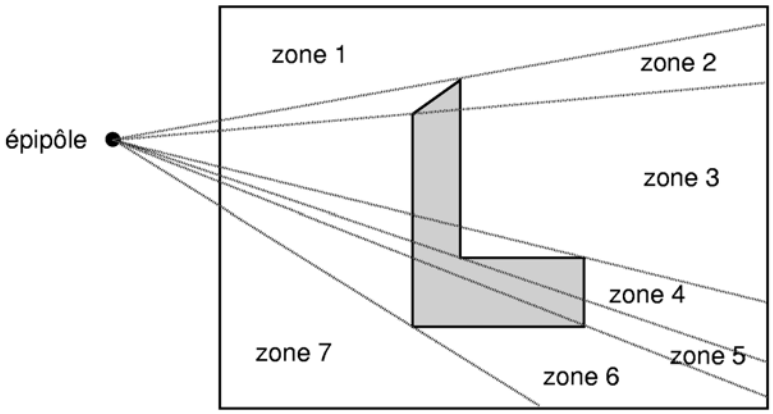


Figure 68 : Image Based Visual Hulls, decoupage de l'image en plusieurs zones.

En utilisant un PC par caméra ainsi qu'un PC pour le rendu, cette méthode génère des images en temps réel pour des scènes dynamiques en direct. Le principal inconvénient de cette méthode est qu'elle n'utilise qu'une image pour calculer la couleur finale d'un pixel. Les auteurs insistent sur l'importance de la qualité de l'extraction de silhouette sur les images de référence.

2.4.3.4 Visual Hulls et GPU

Il existe plusieurs adaptions des *Visual Hulls* capable d'utiliser l'accélartion matérielle de la carte graphique. Le processeur de la carte graphique, GPU (*Graphic Processing Unit*), est alors utilisé comme alternative au processeur de l'ordianteur, le CPU (*Central Processing Unit*). Ces adaptions des *Visual Hulls* utilisant le CPU ne sont pas nécessairement plus rapides mais libèrent systématiquement le CPU pour d'autres tâches.

Li et al. [LMS03b] proposent une méthode de rendu utilisant des textures projetées. Comme pour les méthodes précédentes, les étapes de segmentation, d'extraction de silhouette et d'approximation polygonale de cette silhouette sont réalisées en software. Le rendu est plus direct puisqu'il n'est pas nécessaire de calculer d'intersection polyhédrale. Pour chaque image de référence, on parcourt les segments de la silhouette un par un. Pour chacun de ces segments, on crée un triangle passant par le centre de la caméra associée à l'image de référence et par les deux extrémités du segment (figure 69). Chacune des autres caméras de référence projette la silhouette qui lui est associée sur ce triangle en veillant à activer l'*alpha-blending* (gestion de la transparence). La valeur de transparence (alpha) d'un pixel vaut 1 s'il est situé à l'intérieur d'une silhouette et 0 sinon. Ainsi, seules les parties « intéressantes » du triangle sont texturées. Cette méthode a plusieurs inconvénients car d'une part, elle ne gère pas les occlusions et d'autre part, des problèmes d'*aliasing* (crénelage) peuvent apparaître dans le cas de caméras mal placées par rapport à une facette.

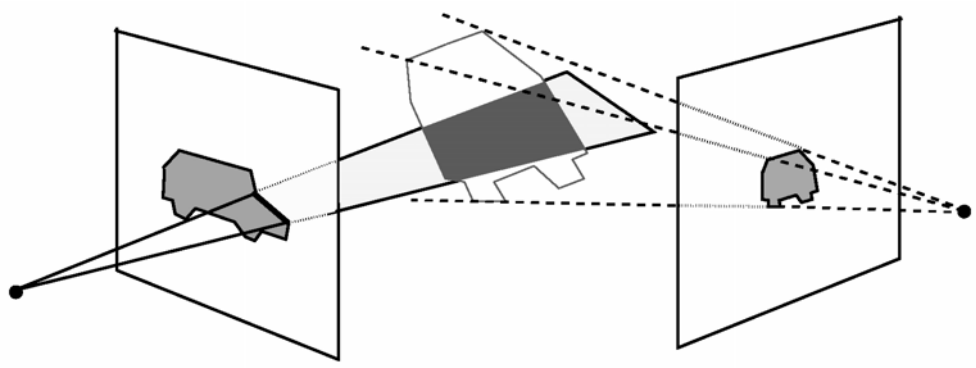


Figure 69 : Visual Hulls et textures projetées.

Li et al. [LMS04] étendent ensuite cette méthode avec l'emploi du *stencil buffer* pour calculer les CSG. Notons que cette nouvelle méthode limite le nombre d'images de référence utilisables pour une nouvelle vue au nombre d'unités de texture. Cela dit, les cartes graphiques du moment proposent en général 16 unités de texture ce qui est largement suffisant pour les méthodes *Visual Hulls* qui utilisent en général moins d'une dizaine de caméras.

2.4.4 Plane Sweep

La méthode des « *Plane Sweep* » (balayage par plans en français) consiste à balayer une scène 3d à l'aide d'un plan virtuel sur lequel sont projetées les images filmées par les caméras. En utilisant des méthodes *multitexturing*, il est possible de détecter des correspondances de points et donc de générer une nouvelle vue de la scène. Ce procédé fonctionne en temps réel en exploitant au mieux les possibilités des cartes graphiques. La principale contribution de nos recherches concerne cette méthode, nous la décrivons donc en détail dans la partie qui suit.

2.4.5 Bilan

Cette dernière section est consacrée aux méthodes de rendu à base de vidéos temps réel. Compte tenu des contraintes fortes qui leur sont imposées, ces méthodes sont peu nombreuses.

La première méthode proposée, « 3D TV », consiste à recréer un *Light Field* « réel ». Pour cela, il nécessite un arsenal d'ordinateurs et de vidéo-projecteurs et évidemment de caméras. Les résultats de cette méthode sont très intéressants mais il faut reconnaître que son efficacité est totalement tributaire d'une grande quantité de matériel de haute technologie. Cette méthode n'est donc pas à la portée de tout le monde.

La méthode des *Light Field* dynamiques est une adaptation des *Light Fields* exposés dans les sections précédentes. L'idée principale permettant à cette méthode d'effectuer du rendu à base de vidéos en direct repose sur le principe de requêtes et de partage des traitements à effectuer. Le principe de requêtes permet d'éviter d'avoir à transmettre la totalité des informations disponibles alors que seule une partie de ces informations est nécessaire. Le partage du travail consiste à séparer les tâches et à les distribuer à d'autres ordinateurs. Il faut remarquer que toutes les méthodes de rendu ne peuvent pas prétendre à cette séparation des tâches et qu'il s'agit donc d'un avantage du *Light Field* par rapport aux autres méthodes.

Enfin, la dernière méthode décrite est la méthode des *Visual Hulls*. Il s'agit de la principale méthode concurrente des *Plane Sweep* détaillés dans le chapitre suivant. Comme toutes les méthodes se passant de corrélations entre images, cette méthode peut gérer les reflets spéculaires. De plus, elle tolère toutes les dispositions de caméras, y compris lorsque les caméras sont disposées tout autour de la scène. Il est alors possible d'en faire virtuellement le tour. Elle a cependant deux défauts majeurs : d'une part elle ne gère que les scènes composées d'un objet principal et ne restitue pas l'arrière plan. Ce dernier doit d'ailleurs être statique pour ne pas perturber le calcul de la silhouette. D'autre part, l'objet filmé doit être convexe sans quoi il subit des distorsions sur l'image générée. Tout comme les *Plane Sweep*, les *Visual Hulls* sont particulièrement prédisposés à une adaptation sur GPU leur permettant de bénéficier au mieux des performances de la carte graphique.

3 « Plane Sweep » : Notre contribution

Ce chapitre constitue notre contribution principale à la méthode des *Plane Sweep*. Nous y développerons d'une part le principe de fonctionnement et d'autre part un état de l'art complet. Nous présenterons aussi le cheminement de pensée qui nous a mené jusqu'à cette méthode. Ceci permettra au lecteur de faire lien entre les *Plane Sweep* et d'autres techniques de rendu à base de vidéos.

Ensuite, nous exposerons notre nouvelle approche et les avantages qu'elle présente. Nous détaillerons les possibilités qui en découlent, notamment au niveau de la gestion des occlusions. Enfin, nous proposerons un bilan concernant la méthode des *Plane Sweep* en analysant ses points forts et ses faiblesses.

3.1 Principe général

Il existe plusieurs variantes de l'algorithme des *Plane Sweep*. Ce chapitre est consacré à une explication générale de cette méthode et n'est donc spécifique à aucune variante en particulier. Cette description est faite en deux temps, elle commence par une explication détournée ayant pour but de montrer notre cheminement de pensée et de relier cette méthode aux autres techniques existantes. La seconde partie est alors consacrée à une description détaillée de la méthode des *Plane Sweep* à proprement parler.

3.1.1 Première approche

Plutôt que de décrire directement le fonctionnement des *Plane Sweep*, nous allons commencer par décrire le cheminement que nous avons eu pour y parvenir. Cette approche permet de bien faire le lien avec d'autres méthodes concurrentes.

Pour commencer, nous nous sommes orientés vers une approche inspirée du lancé de rayons en nous fixant dès le départ l'objectif d'obtenir un rendu en temps réel. La méthode des *image-based priors* de Fitzgibon et al. [FWZ03] est celle qui se rapproche le plus de notre point de départ.

Etant donné un ensemble de caméras calibrées Cam_i, nous voulons générer une image correspondant au point de vue d'une caméra virtuelle Cam_x définie par l'utilisateur. L'idée générale consiste à traiter les pixels de Cam_x indépendamment les uns des autres, comme pour un lancé de rayons. Pour un pixel p de Cam_x traité, le but est d'estimer la profondeur p_z de p. Pour simplifier le problème, il est préférable de borner l'ensemble des possibilités de profondeur sur une portion de droite correspondant au volume supposé de la scène. En déplaçant un point virtuel p_x sur cette portion de droite, il suffit de comparer les projetés de p_x sur chacune des caméras Cam_i et de les comparer. Si leurs couleurs sont suffisamment cohérentes, le point p_x traité est un candidat potentiel comme valeur finale pour p_z. Comme le montre la figure 70, cette méthode revient donc à comparer des portions de droites épipolaires sur chacune des images de référence.

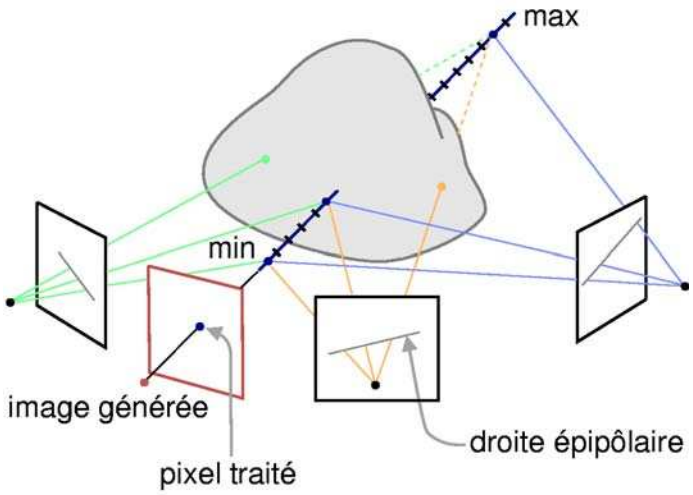


Figure 70 : Image-Based Priors.

Les *Image-Based Priors* de Fitzgibon et al. [FWZ03] utilisent ce principe sur un grand nombre d'images de référence en utilisant les propriétés statistiques des priors. Le pseudo-code de cette méthode peut s'écrire de la manière suivante :

1	Pour chaque pixel p de Cam_x
2	• Déterminer le rayon R passant par le centre de Cam_x et par p
3	• Pour chaque p_x parcourant R de *min* vers *max* par pas Δp
4	- Projeter p_x sur chaque image de référence
5	- **Si** les couleurs des projetés sont cohérentes
6	**Alors** affecter un bon score à p_x
7	• Affecter à p la couleur du p_x de score maximal

Pour l'instant, admettons qu'il existe des méthodes pour estimer si des couleurs sont cohérentes. Que devient cet algorithme si l'on permute les boucles des lignes 1 et 3 ? Tous les pixels de Cam_x sont traités simultanément. Le pseudo-code devient alors :

1	Pour chaque profondeur p_z de *min* à *max* par pas Δp
2	• Déterminer le plan P parallèle au plan image de Cam_x et passant par p_z
3	• Pour chaque pixel p_x de P
4	- Projeter p_x sur chaque image de référence
5	- **Si** les couleurs des projetés sont cohérentes
6	**Alors** affecter un bon score à p_x
7	• Affecter à chaque pixel p de Cam_x la couleur du pixel p_x correspondant de score maximal

Cet algorithme est illustré par la figure 71.

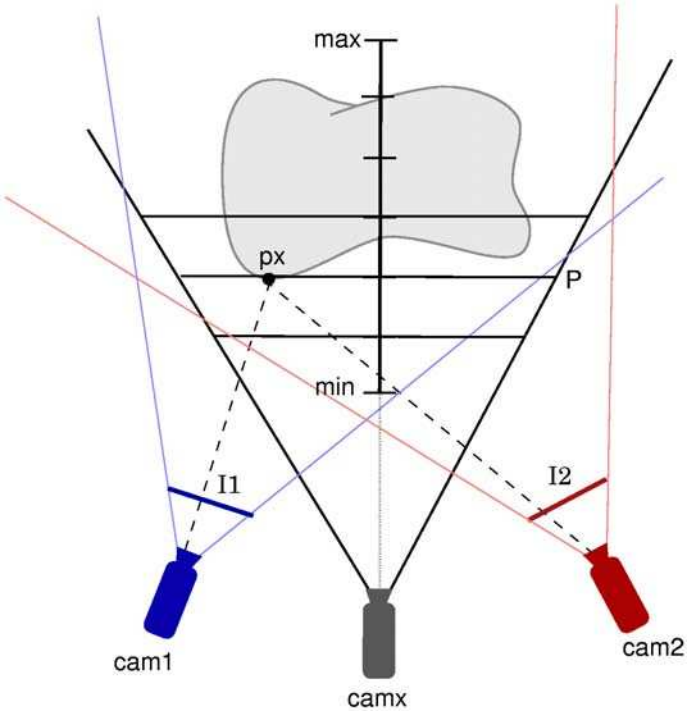


Figure 71 : Plane Sweep, première approche.

Il apparaît que cette méthode s'adapte mieux que la précédente au traitement du pipeline sur une carte graphique. En effet, tous les points p_x de P peuvent être traités dans la même passe en utilisant des textures. Nous reviendrons en détail sur ce point dans les chapitres suivants.

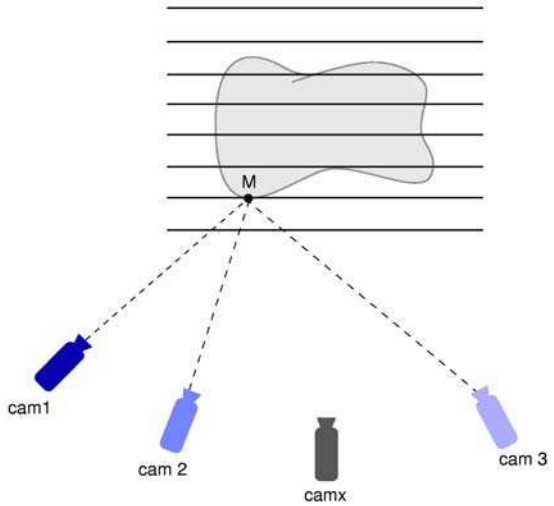


Figure 72 : Plane Sweep, le point M appartient à la surface d'un objet.

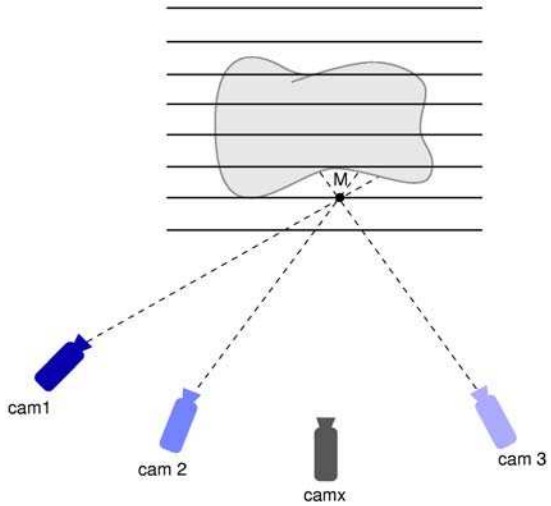


Figure 73 : Plane Sweep, le point M n'appartient pas à la surface d'un objet.

3.1.2 Principe de fonctionnement

Une fois la caméra virtuelle placée, cette méthode commence donc par définir un plan *near* et un plan *far* encadrant la scène. L'espace ainsi défini est discrétisé par des plans parallèles. Imaginons un point *M* situé à la fois sur un plan et sur la surface d'un objet de la scène (figure 72), alors ce point sera vu par les caméras de référence avec la même couleur. Ce n'est en général pas le cas d'un point *M* situé sur un plan mais hors de la surface d'un objet de la scène (figure 73). Un point situé sur un plan qui est vu par toutes les caméras de référence avec la même couleur est donc un point appartenant potentiellement à la surface d'un objet de la scène.

Durant la phase de rendu, chaque plan est donc traité de façon indépendante. Sur chacun de ces plans sont projetées les images de référence. Pour chaque point du plan correspondant à un pixel de la caméra virtuelle, un score est calculé en fonction de la cohérence entre les couleurs provenant de chaque image projetée. Une fois tous les points du plan traités, on les projette sur la caméra virtuelle (figure 74). Si le point projeté sur un pixel p_x de la caméra virtuelle a un score meilleur que le score courant, alors on procède à une mise à jour de la couleur et du score.

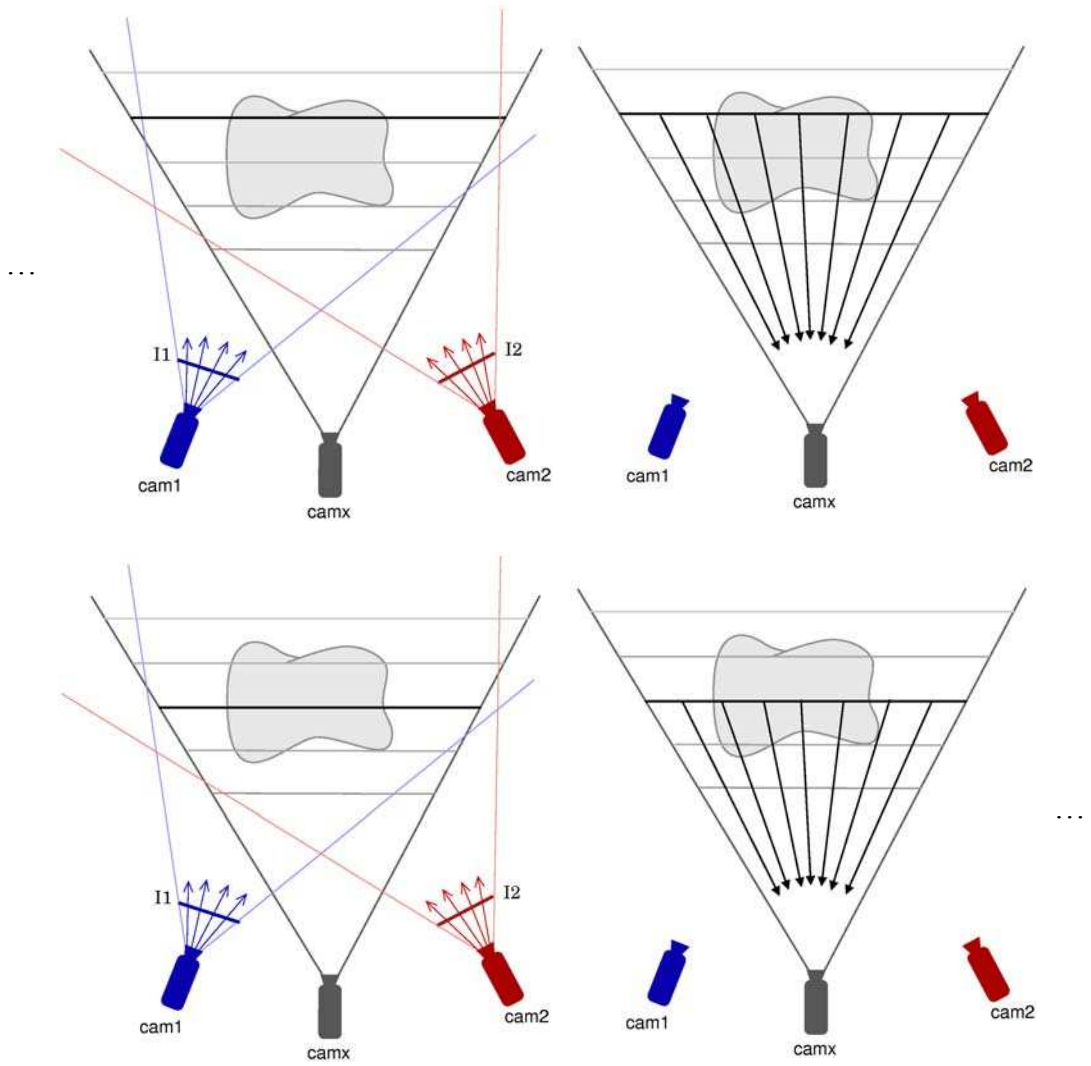


Figure 74 : Plane sweep, détail d'une itération.

La figure 75 illustre quatre plans sur lesquels les quatre images initiales ont été projetées. Les yeux du « bonhomme » se superposent plutôt bien sur la seconde image ce qui indique qu'ils se situent probablement sur le second plan.

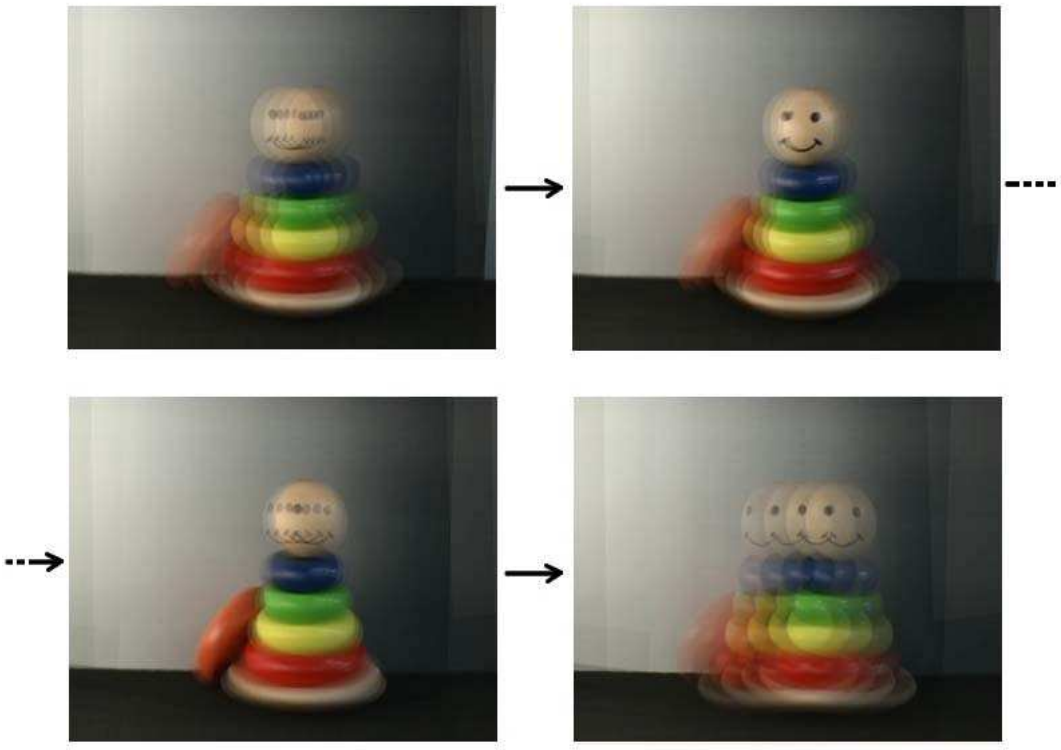


Figure 75 : Plane Sweep, projetés des images de departs sur quatres plans situés à des profondeurs différentes.

3.2 Etat de l'art

Les travaux effectués sur les *Plane Sweep* sont suffisamment rares pour qu'il soit possible de les détailler un par un.

3.2.1 Méthode de Collins

C'est Collins [C96] qui en 1996 introduit la méthode des *Plane Sweep*. Il utilise des vues aériennes de bâtiments sur lesquelles il effectue une détection de contours. Il applique ensuite la méthode des *Plane Sweep* sur les images binaires obtenue afin de reconstruire la géométrie des bâtiments (figure 76).

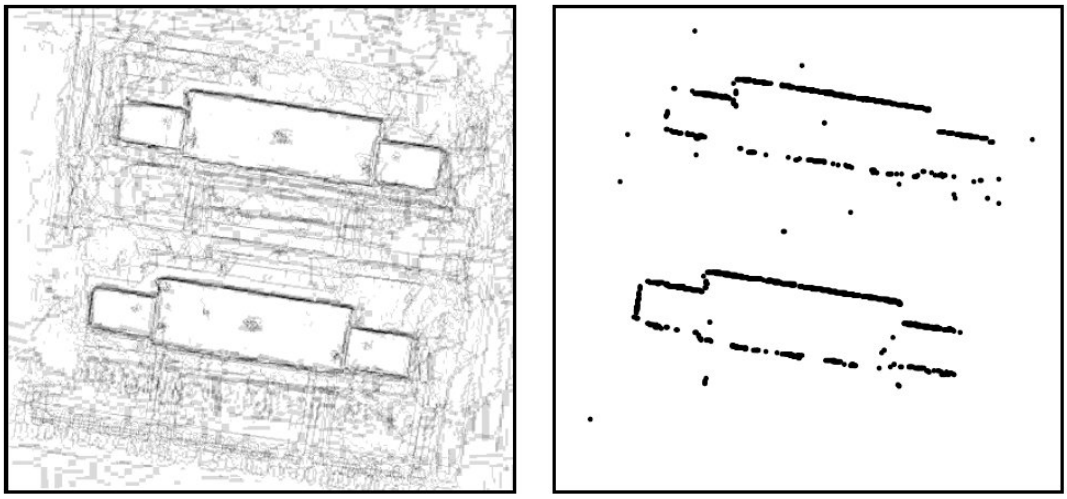


Figure 76 : méthode de Collins sur les images en niveaux de gris.

3.2.2 Méthode de Yang

Yang et al. [YWB02] sont les premiers à proposer une implémentation des *Plane Sweep* utilisant non pas des images binaires mais des images couleur. Ils génèrent une nouvelle vue de la scène (figure 77) et proposent une implémentation de cette méthode en utilisant les possibilités d'accélération de la carte graphique.

Comme pour toutes les méthodes, la première chose à faire est de définir la position et les paramètres de la caméra virtuelle. Il faut ensuite définir un plan *min* et un plan *max* encadrant la scène. Cette scène est ensuite découpée en plans. Sur chacun de ces plans sont projetés les images provenant des caméras. Les auteurs signalent qu'il s'agit d'une homographie et qu'il est possible de l'appliquer directement sur la carte graphique à l'aide de *register combiners*.

Pour chaque « point » de chaque plan, il faut alors calculer un score représentatif de la cohérence des couleurs provenant des images initiales. Par « point » du plan, nous désignons un futur pixel sur la caméra virtuelle. Pour le calcul des scores, une caméra de référence est choisie en fonction de sa position. La caméra de référence est la caméra dont le centre est le plus proche de celui de la caméra virtuelle. Concrètement, pour un plan *D*, chaque image initiale est projetée. Pour chaque point *p* de *D*, le calcul du score s'effectue en sommant les SSD (*Sum of the Square Difference*) entre chaque image projetée et l'image de référence (elle aussi projetée). Finalement, le point *p* se voit attribuer une couleur et un score. La couleur choisie provient de l'image de référence et le score précédemment calculé est stocké dans la composante alpha de la couleur. Tous les points de *D* sont alors projetés dans le *frame-buffer* et seul les pixels dont le score est supérieur au nouveau score proposé sont mis à jour. Une fois tous les plans traités, le *frame-buffer* contient l'image finale.

En choisissant comme couleur une valeur proportionnelle à la profondeur du plan *D* plutôt que la couleur de l'image de référence, il est possible de générer une carte de profondeur. Cette carte de profondeur est toutefois très bruitée c'est pourquoi les auteurs proposent de la flouter à l'aide d'une convolution.

Enfin, Yang et al. soulignent qu'il est possible d'améliorer le rendu en effectuant une première passe de *Plane Sweep* sur les images initiales utilisées avec une résolution

inférieure. Pour cela, il suffit d'utiliser les outils de *mipmapping* proposant ces images avec différents degrés de détails.

Le calcul des scores, l'application des homographies sur les plans discrétisant la scène ainsi que la projection de ces scores sur le *frame-buffer* sont effectués sur la carte graphique à l'aide de *register combiners*. Ceci permet à cette méthode d'effectuer le rendu d'une nouvelle vue en temps réel en utilisant 4 pc dont 3 s'occupant de l'acquisition de vidéos provenant de 5 caméras.

Etant donné le nombre restreint d'instructions, cette méthode est plutôt rapide cependant elle présente deux inconvénients majeurs liés tous deux à l'emploi d'une caméra de référence. Le premier problème concerne la position de la caméra virtuelle qui doit s'orienter exactement dans la même direction que la caméra de référence sans quoi il n'est plus possible de calculer de score. Le second problème concerne le changement de caméra de référence lors du déplacement de la caméra virtuelle. Ce changement peut provoquer « un saut » sur les images générées.

Yang et al. [YWB02]

Figure 77 : méthodes de Yang.

3.2.3 Méthode de Wotzel

Woetzel et al. [WJK04] calculent uniquement une carte de profondeur de la scène et ne génèrent donc pas de nouvelle vue. Lors le calcul du score, plutôt que de choisir une caméra de référence, les auteurs calculent les SSD entre toutes les paires d'images initiales. Parmi les n SSD calculées, les m ($<n$) plus mauvais scores sont supprimés ainsi que leur contribution pour le calcul de la couleur finale. Il s'agit d'un premier pas vers la gestion des occlusions.

Ensuite, plutôt que de stocker le score dans la composante alpha de la couleur, elle est stockée dans la composante de profondeur du fragment ce qui permet de laisser le z-test comparer et mettre à jour les bonnes couleurs. Enfin Woetzel et al. effectuent un filtre médian sur la carte de profondeur obtenue pour supprimer le bruit inhérent aux *Plane Sweep*.

Cette méthode calcule des cartes de profondeur en temps réel en utilisant les ressources de la carte graphique mais ne génère pas de nouvelle vue ce qui la rend difficilement comparable

avec les autres méthodes. De plus, le fait de calculer une SSD sur tous les couples d'images peu vite devenir fastidieux si le nombre de caméras augmente.

La figure 78 montre une des images utilisées pour le calcul de la carte de profondeur illustrée sur la figure 79.

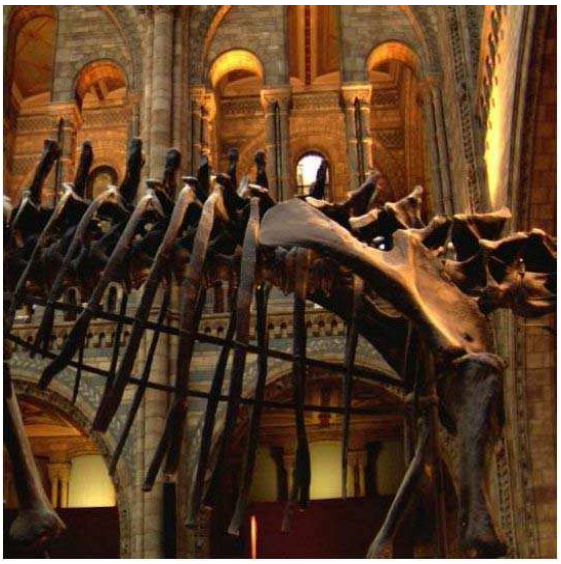


Figure 78 : méthode de Woetzel, une des images de départ.

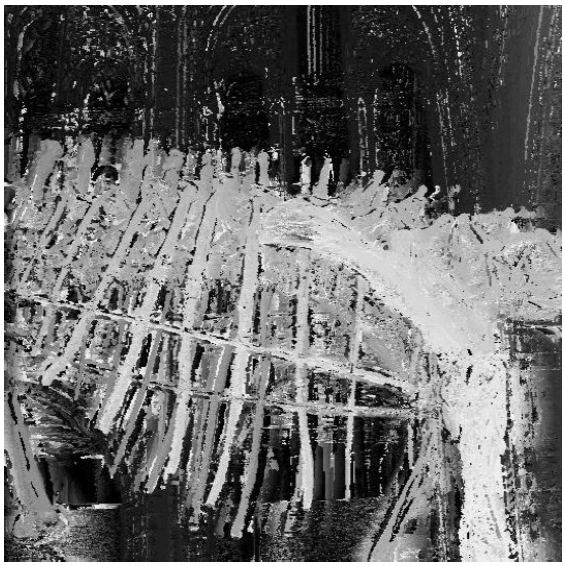


Figure 79 : méthode de Woetzel, carte de profondeur générée.

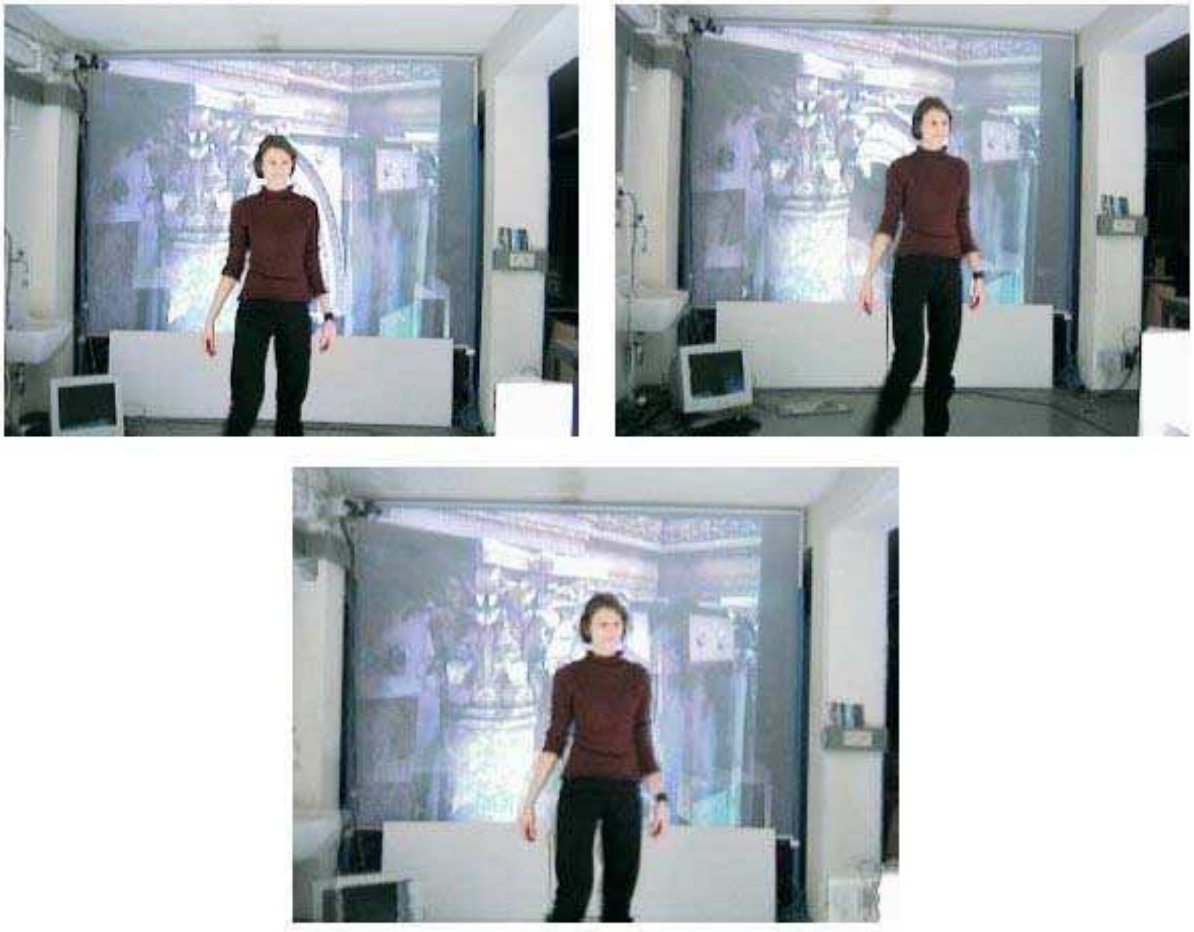
Woetzel et al. [WJK04]

3.2.4 Méthode de Geys

Les méthodes de *Plane Sweep* présentées dans les chapitres précédents utilisent largement les possibilités des cartes graphiques et libèrent ainsi le CPU. Geys et al. [GKV04] proposent d'utiliser ce CPU disponible pour améliorer le rendu. Cette méthode est composée de trois étapes. La première est juste un *Plane Sweep* effectué par le GPU à partir duquel est extrait une carte de profondeur grossière. La seconde étape effectuée par le CPU consiste à optimiser la carte de profondeur avec un algorithme de flot maximal. Enfin, le rendu s'effectue avec une méthode de *texture mapping*.

Cette méthode ne fonctionne qu'avec deux caméras et commence par une phase d'initialisation durant laquelle il faut procéder à une extraction robuste de l'arrière plan. Il faut générer une carte de profondeur précise à partir de laquelle on construit un histogramme du *z-buffer*. Cet histogramme permet de fixer automatiquement les deux plans *min* et *max* encadrant la scène. Cette dernière étape sera d'ailleurs répétée toutes les 50 images afin de mettre à jour ces 2 plans.

Une fois la phase d'initialisation terminée, le calcul de la carte de profondeur en temps réel peut commencer. Ce calcul est en deux passes. Durant la première passe, une carte de profondeur grossière est calculée par *Plane Sweep* à l'aide de la carte graphique. Cette carte de profondeur n'est établie que sur le premier plan de l'image et non sur l'arrière plan redéfini à chaque itération de la méthode. Pour chaque point de chaque plan traité, un score est calculé en utilisant la SAD (*Sum of Asolute Distance*) entre les deux images de référence. Le résultat obtenu pour chaque plan est sauvegardé sous forme de texture en stockant le score dans les composantes de couleurs et le numéro du plan dans la composante alpha.



Geys et al. [GKV04]

Figure 80 : méthode Geys.
Les deux images du haut correpondent aux images
de départ. L'image du bas est une image générée.

La seconde passe de cet algorithme est faite en parallèle par le CPU. Il s'agit d'améliorer la carte de profondeur préalablement calculée. Pour cela, les auteurs proposent une méthode de minimisation d'énergie basée sur une méthode de flot maximal sur des graphes. La variable d'énergie à minimiser contient un terme de correspondance préalablement calculé durant la première passe. Un terme de continuité temporelle qui correspond à une pénalité dans le cas de changement brusque de la couleur d'un pixel. Notons que ce terme se base sur un changement temporel continu des couleurs valable uniquement pour un framerate conséquent. Le terme suivant est un terme d'occlusion entre les pixels du premier plan et les pixels de l'arrière plan. Finalement, le dernier terme concerne la continuité spatiale des couleurs. Les auteurs partent du principe qu'une discontinuité géométrique sur une image implique en général une discontinuité des couleurs. Ce terme de continuité spatial est calculé sur le 4-voisinage des pixels traités. La minimisation de cette fonction d'énergie s'effectue à l'aide d'outils sur les graphes. Le graphe utilisé est constitué de chaînes représentant chacune un pixel de l'image finale. Une chaîne contient autant de nœuds qu'il y a de plans possibles pour un pixel. Chaque chaîne relie un sommet de départ à un sommet d'arrivée. D'autre part, les chaînes sont reliées entre elles : chaque nœud est en relation avec les 4 nœuds de même profondeur situés sur les chaînes voisines. Le coût pour passer d'un nœud à un autre dans une même chaîne correspond au changement d'énergie attribué à chaque nœud. Le passage d'une chaîne à l'autre est défini par le terme de continuité spatiale de la fonction d'énergie. La solution optimale est déterminée par un algorithme de flot maximal.

Cet algorithme consiste à couper une et une seule fois chaque chaîne à une profondeur déterminée. Cela définit la carte de profondeur finale. Finalement, le rendu s'effectue par *texture mapping* sur la carte de profondeur calculée.

Cette méthode est un bon exemple de gestion optimale des ressources d'un ordinateur. Elle permet de passer d'une caméra de référence à l'autre en temps réel pour des scènes dynamiques. Cette méthode a toutefois ses limites puisqu'elle ne traite que deux caméras et se base sur une représentation de la scène sous forme de premier plan dynamique et d'arrière plan statique. D'ailleurs seules les occlusions entre ces deux plans sont gérées. Pour finir, le passage à trois caméras introduit dans [GV05] n'est pas trivial.

La figure 80 montre les deux vues de référence ainsi que la nouvelle image obtenue.

3.2.5 Méthode de Photo Hulls

Li et al. [LMS04] proposent les *Photo Hulls* (figure 81) combinant la méthode des *Visual Hulls* avec la méthode de *Plane Sweep*. Cette dernière ne s'applique alors que sur l'objet principal de la scène. Le temps de calcul supplémentaire n'est donc pas conséquent. Notons aussi que l'ajout des *Plane Sweep* aux *Visual Hulls* permet d'envisager de traiter les parties convexes des objets représentés. Cependant la disposition des caméras tout autour de la scène caractérisant la méthode des *Visual Hulls* est en partie contradictoire avec la disposition des caméras pas trop distantes les unes des autres de la méthode des *Plane Sweep*.

Figure 81 : méthode des Photo Hulls.

3.2.6 Bilan

Les méthodes présentées dans cette section présentées après 1996 constituent une évolution de la méthode des *Plane Sweep* proposée par Collins [C96]. Toutes ces méthodes utilisent pleinement les ressources du processeur de la carte graphique ce qui leur permet de générer des nouvelles vues de la scène en temps réel.

Yang et al. [YWB02] sont les premiers à proposer une implémentation en GPU. Leur méthode se base sur l'emploi d'une caméra de référence à partir de laquelle des scores sont calculés. Ils proposent en outre des optimisations utilisant le mipmapping.

La méthode proposée par Woetzel et al. [WJK04] optimise légèrement la méthode de Yang et al. en rejetant systématiquement les *n* moins bonnes contributions de couleurs. Il s'agit là d'un premier pas vers la gestion des occlusions. Cependant cette méthode est elle aussi basée sur une utilisation de caméras de référence. De plus, Woetzel et al. ne font que calculer une carte de profondeur sans générer de nouvelles vues de la scène, ce qui rend difficile la comparaison avec les autres méthodes.

Geys et al. [GKV04] proposent une adaptation des *Plane Sweep* basée sur une répartition de la scène en premier plan et arrière plan. Cette méthode permet notamment de gérer les occlusions entre un objet du premier plan et un objet de l'arrière plan. De plus, Geys et al. proposent une amélioration de la carte de profondeur par une méthode de minimisation d'énergie. Le rendu en temps réel est toutefois conservé grâce à une répartition des tâches entre le CPU et le GPU. Cependant cette méthode n'est valable que pour un système pourvu de 2 caméras ce qui constitue un inconvénient majeur. De plus, compte tenu de la décomposition de l'image en premier plan et arrière plan, cette méthode nécessite un arrière plan statique.

Enfin, la méthode des *Photo Hulls* proposée par Li et al. [LMS04] est un mélange des *Plane Sweep* et des *Visual Hulls*. Li et al. sont parvenus à tirer parti des deux méthodes et à en garder les avantages majeurs à l'exception d'un : tout comme les *Visual Hulls*, les *Photo Hulls* ne gèrent pas l'arrière plan alors que la méthode des *Plane Sweep* le permet. Le fait de ne pas gérer l'arrière plan simplifie considérablement le problème puisqu'il diminue le nombre de données à traiter et accélère ainsi le processus de rendu. D'autre part, la suppression de arrière plan limite sérieusement les risques d'occlusions partielles.

Finalement, les méthodes utilisant une caméra de référence souffrent de « sauts » sur les images générées lors d'un changement de caméra de référence. De plus, la gestion de l'arrière plan implique une gestion des occlusions partielles. C'est sur ces deux points que s'orientent nos travaux.

3.3 Nouvelle approche des Plane Sweep

Lors de la création d'une nouvelle vue, les méthodes décrites précédemment utilisent toutes soit une caméra de référence la plus proche de la caméra virtuelle, soit uniquement 2 caméras en entrée ce qui revient à choisir l'une d'elle comme caméra de référence. L'utilisation d'une caméra de référence engendre des sauts sur l'image générée, notamment lors du changement de caméra de référence. De plus, la caméra virtuelle doit s'orienter exactement dans la direction de sa caméra de référence sans quoi celle-ci perd son rôle de « guide » pour les autres caméras. Nous proposons dans le chapitre suivant une solution pour s'affranchir de l'emploi d'une telle caméra de référence.

3.3.1 Calcul de scores

Le placement de la caméra virtuelle et les paramètres de la scène permettent de définir l'ensemble des plans servant à discrétiser la scène. Pour un plan donné, plutôt que de projeter une image de référence et de la comparer tour à tour avec le projeté des images

provenant des autres caméras, nous proposons de tirer parti de toutes les caméras en même temps.

Etant donné un plan D et un point p de ce plan (figure 82). Chaque caméra cam_i (i = 1…nombre de caméras) va projeter sur p un rayon de couleur C_i. Le but de l'opération est de trouver un score représentatif de la cohérence entre toutes les couleurs C_i de telle sorte que si les couleurs sont très différentes, le score obtenu doit être mauvais et si les couleurs sont semblables, le score doit être bon. Quel que soit le score, il faut ensuite fournir une couleur représentative de l'ensemble des couleurs. Pour le moment, nous nous contenterons de la couleur moyenne.

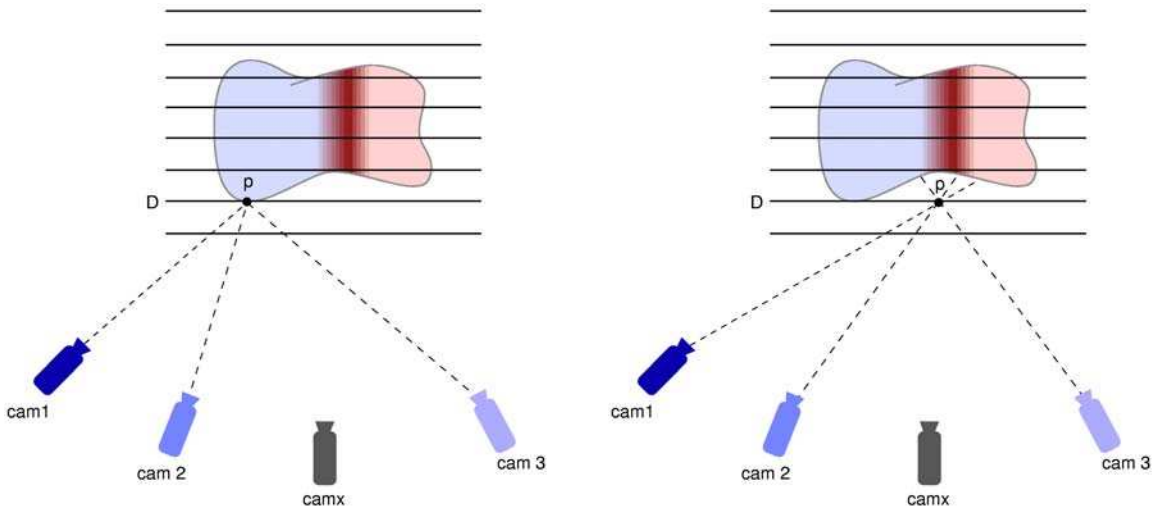


Figure 82 : Plane Sweep, calcul de score.

En considérant les couleurs C_i comme des points de $\mathbb{R}^3$, nous proposons de choisir comme score la variance de ces couleurs. Celle-ci varie comme l'écart type de ces couleurs, représentatif de la distance moyenne entre les couleurs C_i et la couleur moyenne. Ainsi, lorsque les couleurs C_i seront similaires, la variance sera faible ce qui correspond à un bon score et la couleur moyenne sera tout à fait représentative de l'ensemble des couleurs. Dans le cas où les couleurs se sont pas similaires, la variance sera élevée ce qui correspond à un mauvais score. Peu importe alors la pertinence de la couleur moyenne puisqu'elle ne sera probablement pas utilisée compte tenu du mauvais score. Finalement, la couleur finale C_{final} et le score se calculent donc avec :

$$C_{final} = \begin{pmatrix} R_{final} \\ V_{final} \\ B_{final} \end{pmatrix} = \frac{1}{card(\{C_i\})} \sum_{k=1}^{card(\{C_i\})} \begin{pmatrix} R_k \\ V_k \\ B_k \end{pmatrix}$$

et

$$score = \frac{1}{card(\{C_i\})} \sum_{k=1}^{card(\{C_i\})} \left[\left(R_k - R_{final}\right)^2 + \left(V_k - V_{final}\right)^2 + \left(B_k - B_{final}\right)^2 \right]$$

où (R,V,B) sont les composantes rouge, verte et bleue d'une couleur.

Cette méthode permet de s'affranchir définitivement d'une caméra de référence et des contraintes sous-jacentes puisque toutes les caméras sont traitées simultanément. De plus, cette méthode conserve l'aspect de « guide » attribué à la caméra la plus proche et dirigée dans le même sens que la camera virtuelle. En effet, considérons une caméra virtuelle placée et orientée de façon similaire à l'une des caméras Cam_n, quel que soit le plan traité, un point du plan associé à un pixel particulier de la caméra virtuelle se verra attribuer à chaque fois la même couleur de la part de Cam_n. Cette couleur C_n sera donc privilégiée et le score optimal sera certainement obtenu pour un ensemble de couleurs proches de C_n. Si la caméra virtuelle s'éloigne de Cam_n pour se rapprocher d'une autre caméra, il n'y aura aucun saut sur la séquence d'images générée puisqu'il n'y a pas de caméra de référence et donc pas de changement de caméra de référence. Cette méthode bénéficie donc de tous les avantages des méthodes précédentes sans en subir les inconvénients.

Nous avons utilisé la couleur moyenne des C_i pour représenter l'ensemble de ces couleurs. Il paraît judicieux de pondérer la contribution de chaque couleur C_i en fonction de sa distance à la couleur moyenne. Dans le cas d'un ensemble de couleurs comportant un intrus, la couleur finale ne sera pas pénalisée puisque la couleur intruse aura un coefficient de pondération très faible. Dans ce cas, le score et la couleur finale se calculent de la façon suivante :

$$C_{moy} = \begin{pmatrix} R_{moy} \\ V_{moy} \\ B_{moy} \end{pmatrix} = \frac{1}{card(\{C_i\})} \sum_{k=1}^{card(\{C_i\})} \begin{pmatrix} R_k \\ V_k \\ B_k \end{pmatrix}$$

Pour chaque couleur C_i : $dist_i = \left(R_i - R_{moy}\right)^2 + \left(V_i - V_{moy}\right)^2 + \left(B_i - B_{moy}\right)^2$

$$score = \sum_{k=1}^{card(\{C_i\})} dist_k$$

$$C_{final} = \begin{pmatrix} R_{final} \\ V_{final} \\ B_{final} \end{pmatrix} = \frac{1}{score.(card(\{C_i\})-1)} \sum_{k=1}^{card(\{C_i\})} (score - dist_k) \begin{pmatrix} R_k \\ V_k \\ B_k \end{pmatrix}$$

Le score ne correspond pas directement à la variance mais varie de la même façon et préserve donc le classement.

En pratique la différence entre la couleur finale calculée avec cette méthode et la couleur moyenne n'est pas visible sur l'image générée. En effet, la couleur finale sélectionnée pour la nouvelle vue correspond à la couleur de meilleur score où les couleurs C_i sont relativement homogènes (sinon, elle ne serait probablement pas la couleur de meilleur score). Dans le cas de couleurs C_i homogènes, il est tout à fait inutile de pondérer les couleurs puisqu'elles sont toutes très semblables. Cette méthode a donc été rejetée car elle ralentit l'application et n'améliore pas la qualité du rendu.

3.3.2 Mipmapping

Yang et al. [YWB02] suggèrent d'utiliser différents niveaux de détails (Level Of Details) lors du calcul des scores. Il s'agit d'un moyen détourné de prendre en compte les couleurs du voisinage d'un pixel plutôt que de faire des comparaisons de pixels un à un. Le *mipmapping* est un moyen d'utiliser des images avec des niveaux de détails variables. A chaque augmentation du niveau de *mipmapping*, on réduit de moitié la largeur et la hauteur de l'image ce qui diminue la résolution de l'image au quart de sa résolution précédente. Cependant avec cette méthode, le voisinage d'un pixel n'est pas nécessairement centré autour de lui ce qui réduit sa pertinence. Yang et al. utilisent une pyramide de niveau de détails qui contribuent chacun au calcul du score final.

Nous avons constaté que l'ajout de *mipmapping* sur notre méthode améliorait de façon significative la qualité visuelle du rendu. Cependant, nos tests ont montré que contrairement à la méthode de Yang et al., un niveau de détails supplémentaire était suffisant et qu'un troisième n'apportait rien de plus. Par ailleurs, l'usage de *mipmapping* ajoute des opérations supplémentaires pour chaque pixel de chaque plan, ce qui ralentit l'application.

3.3.3 Implémentation

Cette méthode a été implémentée en C++ sous Linux en utilisant les bibliothèques OpenGL 2.0, GLSL et Glut. Les caméras sont préalablement calibrées et les plans *Dmin* et *Dmax* encadrant la scène sont déterminés. Les performances temps réel de ce programme sont principalement dues à l'utilisation massive de la carte graphique et à sa maniabilité, notamment en ce qui concerne les *vertex shaders* et le *fragment shaders* (cf. annexe 3). Voici les étapes principales d'un cycle du programme générant une nouvelle image :

1 Placer la caméra virtuelle
2 Charger les images provenant des caméras synchronisées
3 Initialiser le z-buffer et le color-buffer
4 Pour chaque plan *D* de *Dmin* à *Dmax* interpolé linéairement
5 • Dessiner un rectangle sur *D* de telle sorte qu'il occupe tout l'écran
6 • **Vertex shader** : Stocker la position dans une variable `pos` des 6 sommets de ce rectangle (composé de 2 triangles).
7 • **Fragment shader** : Pour tous les fragments (futurs pixels de l'image finale) provenant des triangles
8 - Récupérer les couleurs de chaque caméra par texture projetée en utilisant la variable `pos` automatiquement interpolée
9 - Calculer un score et une couleur finale
10 - couleur du fragment ← couleur finale
11 - profondeur z du fragment ← score
12 • **z-test** : autorise ou non la mise à jour du score (profondeur) et de la couleur dans le frame-buffer
13 Afficher l'image finale

La phase 5 consistant à dessiner un rectangle inscrit dans un des plans *D* est réalisée en projetant les quatre coins de l'image virtuelle sur *D*. Les quatre points obtenus forment le rectangle sur lequel vont être projetées les textures associées aux caméras.

Le *vertex program* utilisé à la ligne 6 est spécialement simple :

```
uniform sampler2D texture0;
uniform sampler2D texture1;
uniform sampler2D texture2;
uniform sampler2D texture3;
varying vec4 pos;
void main(void)
{
  pos = vec4(gl_Vertex);
  gl_Position = ftransform();
}
```

En revanche, le fragment program de la ligne 7 est un peu plus volumineux. C'est dans ce programme qu'un fragment accède aux coordonnées de textures des caméras en utilisant des textures projetées. C'est aussi dans ce programme que sont calculés le score et la couleur finale d'un fragment.

```
uniform sampler2D texture0;
uniform sampler2D texture1;
uniform sampler2D texture2;
uniform sampler2D texture3;

varying vec4 pos;
void main(void)
{
  vec3 colors[4];
  vec4 coord0 = gl_TextureMatrix[0]*pos;
  colors[0] = vec3(texture2DProj(texture0,coord0));
  vec4 coord1 = gl_TextureMatrix[1]*pos;
  colors[1] = vec3(texture2DProj(texture1,coord1));
  vec4 coord2 = gl_TextureMatrix[2]*pos;
  colors[2] = vec3(texture2DProj(texture2,coord2));
  vec4 coord3 = gl_TextureMatrix[3]*pos;
  colors[3] = vec3(texture2DProj(texture3,coord3));

  vec3 average = colors[0] + colors[1] + colors[2] + colors[3];
  average = average / 4.0;

  float score = abs(distance(colors[0],average));
  score = score + abs(distance(colors[1],average));
  score = score + abs(distance(colors[2],average));
  score = score + abs(distance(colors[3],average));

  gl_FragDepth = score;
  gl_FragColor = vec4(average,1.0);
}
```

L'utilisation de textures projetées nécessite simplement de spécifier la matrice de texture comme étant la matrice de projection d'une caméra à laquelle il faut ajouter une ligne nulle correspondant à la composante z. Notons toutefois que le calibrage de la caméra en question doit être effectué sur une image normée (i.e. dont les coordonnées maximum et minimum sont 1 et 0 pour la hauteur ainsi que pour la largeur).

Dans le cas où l'on désire améliorer le rendu avec une passe supplémentaire sur chaque plan en utilisant les images avec un niveau de détails inférieur, le *fragment program* devient :

```
uniform sampler2D texture0;
uniform sampler2D texture1;
uniform sampler2D texture2;
uniform sampler2D texture3;
varying vec4 pos;

void main(void)
{
  float bias = 0.0;
  vec3 colors[4];
  float score = 0.0;
  vec3 average;

  // coordonnees de texture
  vec4 coord1 = gl_TextureMatrix[1]*pos;
  vec4 coord0 = gl_TextureMatrix[0]*pos;
  vec4 coord2 = gl_TextureMatrix[2]*pos;
  vec4 coord3 = gl_TextureMatrix[3]*pos;

  // niveau de detail grossier
  bias = 3.0;
  colors[0] = vec3(texture2DProj(texture0,coord0,bias));
  colors[1] = vec3(texture2DProj(texture1,coord1,bias));
  colors[2] = vec3(texture2DProj(texture2,coord2,bias));
  colors[3] = vec3(texture2DProj(texture3,coord3,bias));

  average = colors[0] + colors[1] + colors[2] + colors[3];
  average = average / 4.0;

  score += abs(distance(colors[0],average));
  score += abs(distance(colors[1],average));
  score += abs(distance(colors[2],average));
  score += abs(distance(colors[3],average));
  if(score > 0.8) discard;

  // niveau de detail fin
  bias = 0.0;
  colors[0] = vec3(texture2DProj(texture0,coord0,bias));
  colors[1] = vec3(texture2DProj(texture1,coord1,bias));
  colors[2] = vec3(texture2DProj(texture2,coord2,bias));
  colors[3] = vec3(texture2DProj(texture3,coord3,bias));

  average = colors[0] + colors[1] + colors[2] + colors[3];
  average = average / 4.0;

  score += abs(distance(colors[0],average));
  score += abs(distance(colors[1],average));
  score += abs(distance(colors[2],average));
  score += abs(distance(colors[3],average));

  score = min(1.0, score*0.05);
  gl_FragDepth = score;
  gl_FragColor = vec4(average,1.0);
}
```

Enfin, pour générer un carte de profondeur plutôt qu'une nouvelle image, il suffit de remplacer l'instruction

```
gl_FragColor = vec4(average,1.0);
```

par

```
gl_FragColor = vec4(vec3(gl_FragCoord.z),1.0);
```

Pour finir, notons que le nombre de caméras utilisées dans cette méthode est limité par le nombre d'unités de texture disponibles sur la carte graphique, soit 8 ou 16 pour les cartes graphique actuelles. Dans le cas d'un dispositif comprenant trop de caméras, il suffit alors de se contenter des caméras les plus représentatives de la caméra virtuelle.

3.3.4 Mode opératoire

Les paramétrages préliminaires concernent le calibrage des caméras ainsi que le calcul des plans *min* et *max* délimitant la scène. Durant nos tests, nous avons calibré nos caméras avec le *Gold Standard Algorithm* décrit dans le chapitre 1.2.4. Nous utilisons pour cela une mire constituée de deux « damiers » non coplanaires illustrés sur la figure 83. Les points de cette mire sont d'abord localisés à l'aide d'un bras d'acquisition (microscribe 3d) avec une précision inférieure au millimètre.

Figure 83 : les points caractéristiques de la mire sont localisés à l'aide d'un bras articulé.

Nos tests ont été effectués avec des caméras Tri-CCD ce qui garanti une balance des blancs correcte. Cette balance des blanc est effectuée manuellement à l'aide d'une feuille blanche. Après avoir disposé les caméras et les éclairages, nous disposons la mire sur la scène à l'endroit où figureront les objets filmés. Nous prenons alors un cliché de cette mire avec chacune des caméras. Celles-ci sont munies de télécommandes (la même télécommande contrôle toutes les caméras car elles sont de même modèle) ce qui évite d'avoir à y toucher. Les images capturées serviront au calibrage des caméras.

La mire est ensuite retirée et remplacée par les éléments de la scène. Avant de commencer à filmer la scène, il faut définir les plans *min* et *max*. Nous utilisons pour cela un objet plan tel qu'une pochette de CD que l'on dispose devant l'objet puis derrière l'objet (figure 84). Les deux plans seront définis en localisant 3 coins de la pochette du CD par triangularisation.

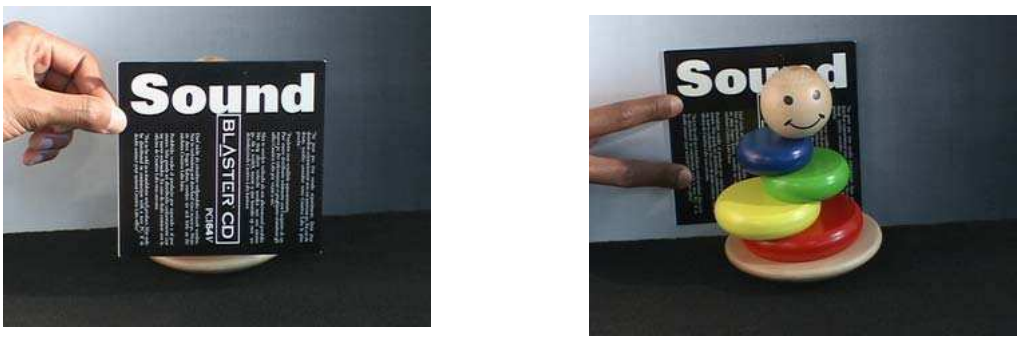


Figure 84 : choix du plan *min* et du plan *max*.

Le tournage peut alors commencer et il faudra alors veiller à ne pas placer (ni déplacer) d'objet hors des limites définies (figure 85).

Figure 85 : Plane Sweep, dispositif d'acquisition.

Une fois l'acquisition effectuée, il faut calibrer les caméras. Pour cela, nous disposons d'un détecteur de mire automatique fait maison qui calibre toutes les caméras simultanément. Une fois les caméras calibrées, il est possible de déterminer les plans *min* et *max* par triangularisation. Ensuite, il ne reste plus qu'à utiliser l'application principale. Il en existe deux versions, la première reçoit un script correspondant à un scénario décrivant les mouvements de la caméra virtuelle. La seconde version permet de naviguer dans la scène à l'aide du clavier et de la souris ou encore à l'aide de capteurs de position dans la salle de réalité virtuelle de l'université de Marne-la-Vallée.

3.3.5 Résultats

Nos tests ont été effectués en utilisant un Athlon 1 GHz et une Nvidia GeForce 6800 GT. L'acquisition des images a été effectuée à l'aide de quatre caméras Tri-CCD Sony DCR-PC 1000E. La figure 86 montre un exemple de quatre images de la même scène prise à partir des quatre caméras.

Figure 86 : quatres images de départ.

La série d'image suivante (figure 87) a été générée en utilisant les images de la figure 86 réduites en 320x240. Ces images ont été obtenues en subdivisant la scène avec 20 plans et sans utiliser de *mipmapping*.

Figure 87 : images générées.

3.3.5.1 Mipmapping

Les effets du *mipmapping* sont visibles sur la figure 88. L'image de gauche a été générée sans *mipmapping* et l'image de droite avec *mipmapping* tout en conservant les autres paramètres. Le *mipmapping* améliore incontestablement le rendu visuel et notamment la restitution des détails, cependant il ajoute aussi une impression de flou sur l'image.

Figure 88 : Plane Sweep et mipmapping.
L'image de gauche est générée sans utiliser de mipmapping. L'image de droite est générée à l'aide de mipmapping lors du calcul des scores.

3.3.5.2 Nombre de plans

Le nombre de plans nécessaires pour discrétiser l'espace dépend en général la scène. La figure 89 montre 5 images générées du même point de vue en utilisant un nombre différent de plans pour subdiviser l'espace. Les résultats montrent que sur cette scène, en dessous de 10 plans, l'image est très bruitée. A partir d'une vingtaine de plans, les résultats peuvent devenir acceptables et au delà de 50 plans, il n'y a plus d'amélioration du résultat visuel. Le temps d'exécution des *Plane Sweep* est directement linéaire en fonction du nombre de plans donc il est nécessaire de trouver l'équilibre idéal entre framerate et qualité du résultat. Enfin, il faut retenir que les résultats présentés sont à moduler en fonction de la scène et notamment de la position des deux plans *min* et *max* encadrant les éléments de cette scène.

5 plans 10 plans 30 plans

50 plans 100 plans

Figure 89 : Plane Sweep, influence du nombre de plans sur la qualité visuelle des images générées.

3.3.5.3 Carte de profondeur

Comme le montre la figure 90, les cartes de profondeurs générées par la méthode des *Plane Sweep* sont en générale d'assez mauvaise qualité. Ce sont principalement les zones de couleur homogène qui posent problème. En effet, sur un aplat de couleur, les corrélations ne sont pas précises. Cependant, sur une telle zone, même si la profondeur d'un pixel est fausse, sa couleur est généralement bonne et ce problème n'est donc absolument pas gênant pour la création de nouvelles vues de la scène. Il le deviendrait si le but de notre application était par exemple d'effectuer une reconstruction géométrique de la scène.

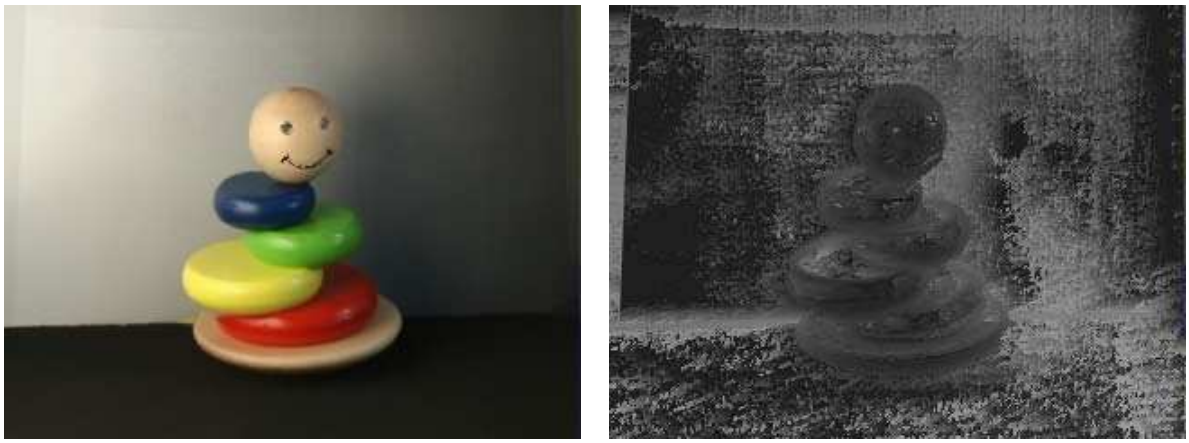


Figure 90 : création d'une carte de profondeur.

3.3.5.4 Performances

Les tests suivants ont été réalisés sur un Athlon 1 GHz et une carte graphique Nvidia GeForce 6800 GT. Les images traitées et les images générées sont en 320x240. Le tableau suivant illustre les *framerate* de notre méthode sans *mipmapping* et avec un niveau de *mipmapping*.

Nombre de plans	Images par seconde sans *mipmaping*	Images par seconde avec *mipmapping*
10	90	30
30	43	11
50	30	7
100	15	3.5

Les tests montrent qu'avec une trentaine de plans, avec lesquels les résultats visuels sont satisfaisants, notre méthode atteint 40 images par seconde, ce qui est suffisant pour être considéré comme une application visuelle temps réel. L'ajout de *mipmapping* augmente considérablement le nombre d'instructions à effectuer pour chaque pixel ce qui explique la baisse de framerate : une application avec *mipmapping* est environ quatre fois plus lente qu'une application sans. Les images traitées sont au format 320x240 ce qui correspond au format de certaines webcam. Leur usage n'est d'ailleurs pas exclu.

Enfin, il est intéressant de noter que notre méthode fonctionne correctement même avec un processeur peu performant comme c'est le cas dans nos tests. L'essentiel est d'avoir une bonne carte graphique.

3.3.5.5 Vidéos

Par manque de matériel, nous n'avons encore pas pu tester notre méthode sur un flux vidéo. Nous avons toutefois simulé un flux vidéo en rechargeant pour chaque caméra une nouvelle image après chaque nouvelle vue générée. Bien sûr, ce protocole n'est pas équivalent à un vrai test utilisant des vidéos. Cependant les caméras vidéos pressenties pour nos tests sont des caméras USB 2.0, les données sont transmises tel quel et ne sont pas compressées. Dans notre protocole, nous remplaçons le transfert d'une image via un port série (USB) par une lecture sur le disque dur probablement plus lente. En pratique, nos tests correspondent donc à du rendu à base de vidéos en différé mais les résultats obtenus ne sont probablement pas aberrants par rapport à du rendu à base de vidéo en direct. Le tableau suivant illustre les framerates obtenus avec une scène statique (rendu à base d'images sans *mipmapping*) ainsi que ceux obtenus avec une scène dynamique (rendu à base de vidéos en différé). Ces tests ont été effectués sur un Pentium 4 à 2,8 GHz accompagné d'une carte graphique Nvidia GeForce 6800 GT. Les images proviennent de 4 caméras et sont au format 320x240.

Nombre de plans	Images par seconde Rendu à base d'images sans *mipmaping*	Images par seconde Rendu à base de vidéos
10	92	22
30	45	21
50	30	20
100	15	16

Les résultats montrent que le framerate est limité à une vingtaine d'images par seconde. Cette limite est fixée par le chargement des 4 nouvelles images à chaque nouvelle vue générée. Ce framerate correspond tout à fait au framerate espéré par notre prochain dispositif de caméras USB. Notons aussi que 20 images par secondes sont suffisantes pour obtenir une sensation de fluidité lors de la navigation dans la scène virtuelle.

3.3.5.6 Bilan

La méthode que nous proposons permet de s'affranchir de l'emploi de caméras de référence. La caméra la plus proche tient naturellement ce rôle et le passage entre deux caméras se fait « en douceur ». Cette méthode bénéficie donc des avantages de l'emploi d'une caméra de référence sans avoir à subir ses inconvénients.

Les images générées sont assez réalistes et la navigation est fluide pour des images dont la résolution n'est pas trop élevée. Les contraintes sur la scène sont peu nombreuses puisqu'il suffit de calibrer les caméras et de définir deux plans encadrant la scène. Notons toutefois que le nombre de caméras utilisables est limité par le nombre d'unités de textures disponibles sur la carte graphique. Cette méthode ne gère pas les reflets spéculaires mais en pratique, certains reflets spéculaires sont supportés correctement (figure 87). C'est notamment le cas lorsque les taches spéculaires sont suffisamment grandes pour couvrir à peu près la même zone quelle que soit la caméra filmant la scène. Enfin, il est préférable de minimiser la distance entre les plans *near* et *far* afin de réduire au minimum le nombre de subdivisions nécessaires pour un rendu correct.

3.4 Gestion des occlusions

Nous exposons dans ce chapitre une extension de notre méthode de calcul de score permettant de gérer les occlusions partielles de la scène. Nous proposons deux approches faciles à insérer dans la méthode de base.

3.4.1 Problématique

Le traitement des occlusions est une des problématiques majeures dans les domaines de reconstruction. En effet, les occlusions sont à la source d'une grande partie d'erreurs d'appariement entre pixels d'images différentes. Certaines méthodes permettent de les détecter mais peu d'entres elles les gèrent correctement.

On parle d'occlusion lorsqu'un point de la scène est visible par une partie des caméras et pas par l'autre. Il existe plusieurs degrés d'occlusion que l'on peut traiter plus ou moins bien.

Lors d'une occlusion totale, la caméra virtuelle est placée face à un objet de la scène qu'aucune caméras n'est en mesure de voir (figure 91). Il est alors absolument impossible d'estimer la couleur et la profondeur de cette région de la scène.

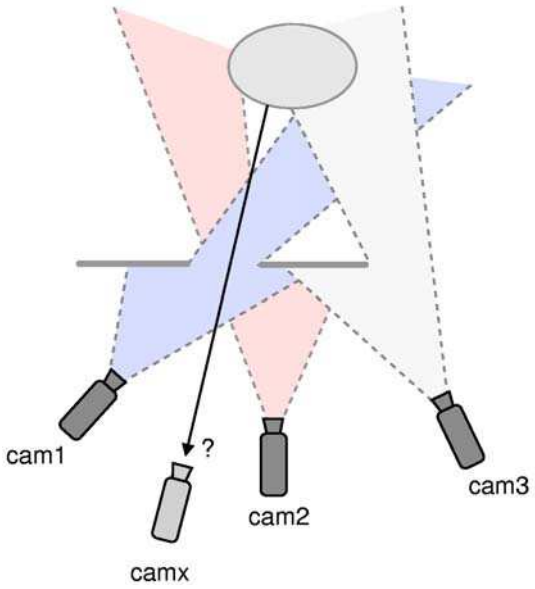


Figure 91 : occlusion totale.

Une occlusion partielle survient lorsqu'une zone de la scène n'est visible que par une partie des caméras (figure 92). Il y a plusieurs degrés d'occlusion partielle dont on évalue la gravité suivant le nombre de caméras dont on dispose et la proportion d'entre elles n'étant pas en mesure de voir la zone en question. Plus on dispose de caméras, plus il est possible de détecter et de régler le problème. Pour les systèmes pourvus d'un nombre limité de caméras, le critère d'évaluation de la gravité du problème devient la proportion de caméras voyant la zone en question.

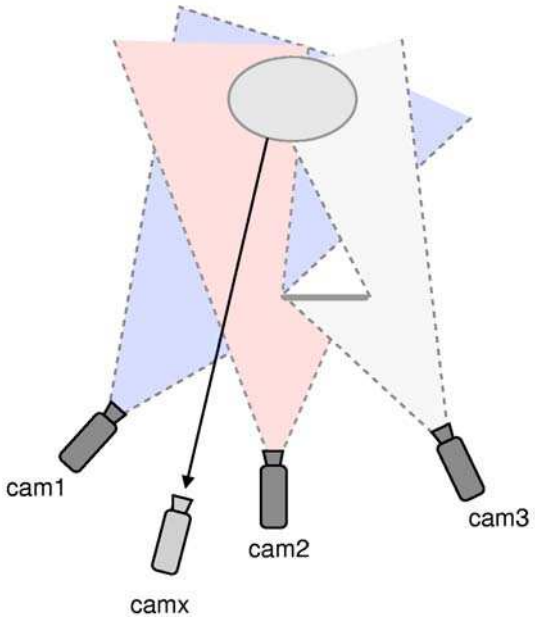


Figure 92 : occlusion partielle.

Les auto-occlusions (*self-occlusions* en anglais) sont des cas particuliers très répandus d'occlusions partielles. Elles surviennent sur les bords des objets vus par une caméra. Au delà de ces bords, la caméra en question ne voit plus l'objet alors qu'une autre caméra excentrée peut éventuellement voir une partie du prolongement de ces bords (figure 93 et figure 94).

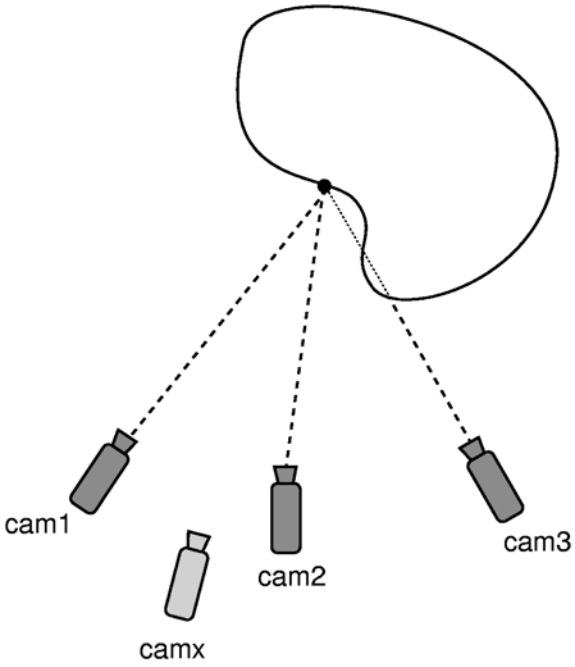


Figure 93 : auto-occlusion classique.

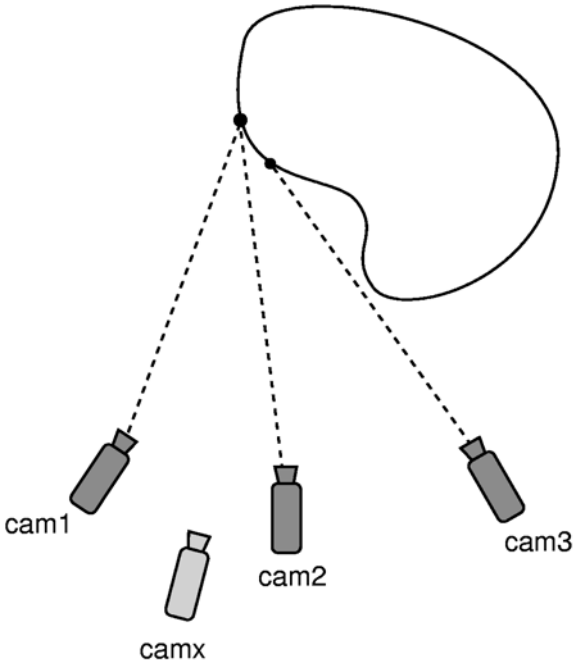


Figure 94 : auto-occlusion sur le bord d'un objet.

Les méthodes que nous proposons sont dédiées à la gestion d'occlusions partielles et d'auto-occlusions.

3.4.2 Sigma-Clipping

Etant donnée un point *M* d'un plan situé sur la surface d'un objet de la scène, dans le cas d'une occlusion partielle de *M* (figure 95), les caméras risquent de projeter des couleurs différentes sur *M* réparties en deux groupes, les couleurs qui correspondent réellement au point *M* et les autres.

Soit *S* l'ensemble des couleurs C_i projetées sur *M* par les caméras. Nous proposons un algorithme itératif utilisant une méthode de *sigma-clippling* permettant d'extraire de *S* les couleurs les moins représentatives de *S*.

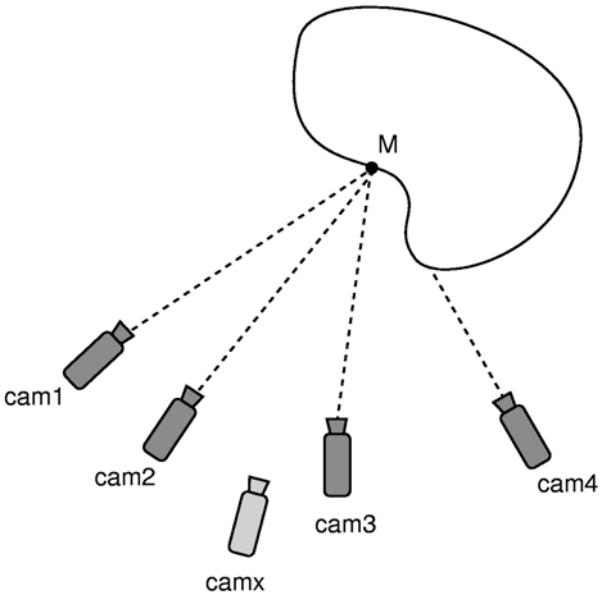


Figure 95 : occlusion partielle et Sigma Clipping.

A chaque itération, la couleur moyenne C_{moy} de $S=\{C_i\}_{i=1...n}$ est calculée. La couleur C_{max} de S la plus éloignée de C_{moy} est alors identifiée. Si C_{max} est trop loin de C_{moy}, alors elle est supprimée de S. Ces calculs sont répétés en boucle jusqu'à stabilisation ou jusqu'à ce qu'il ne reste plus que deux couleurs dans S. Un score représentatif de S est alors calculé en utilisant par exemple une méthode utilisant la variance proposée dans les chapitres précédents. Cet algorithme peut se résumer de la façon suivante :

---

1 **Bool** *stable* = **faux**

2 $S=\{C_i\}_{i=1...n}$

3 C_{moy} = moyenne(S)

4 **Répéter**

5 • Trouver la couleur C_{max} de S la plus éloignée de C_{moy}

6 • **Si** distance(C_{max}, C_{moy}) > d **alors**

7 $S = S - C_{max}$

8 C_{moy} = moyenne(S)

9 **Sinon** *stable* = **vrai**

10 **Tant que** Card(S)>2 **et** *stable* = = **faux**

11 *score* = variance(S, C_{moy}) / Card(S)

Le choix de la distance critique d dépend de la scène filmée et de la disposition des caméras. Le score est calculé en utilisant la variance de S puis il est pondéré par le nombre de couleurs compris dans S afin de favoriser les situations où un maximum de couleurs concordent.

Cette méthode est facilement implémentable dans le *fragment program* de la méthode standard.

3.4.3 Algorithme dynamique

Nous proposons une seconde méthode elle aussi itérative permettant de répondre au même problème. Elle consiste à calculer itérativement le score qu'aurait un ensemble de couleurs sans « la plus mauvaise » d'entre elles. Si ce score est meilleur que le score courant alors c'est qu'il est préférable de supprimer cette couleur de l'ensemble des couleurs.

Cette méthode commence donc par un calcul de la couleur moyenne C_{moy} et d'un score associé à S, par exemple sa variance. La couleur C_{max} de S la plus éloignée de C_{moy} est alors identifiée. Un nouveau calcul du score est effectué sans tenir compte de C_{max}. Si ce nouveau score est meilleur que le précédent, C_{max} est alors supprimé de S. Ces calculs sont répétés jusqu'à stabilisation où jusqu'à ce qu'il n'y ait plus que 2 couleurs dans S. Cet algorithme peut se résumer de la façon suivante :

Bool *stable* = **faux**

$S=\{C_i\}_{i=1...n}$

C_{moy} = moyenne(S)

score = variance(S, C_{moy}) / Card(S)

Répéter

- Trouver la couleur C_{max} de S la plus éloignée de C_{moy}
- S* = S - C_{max}
- C^*_{moy} = moyenne(S*)
- *score** = variance(S*, C^*_{moy}) / Card(S*)
- **Si** *score** est meilleur que *score* **alors**

    $S = S^*$

    $C_{moy} = C^*_{moy}$

    score = *score**

    Sinon *stable* = **vrai**

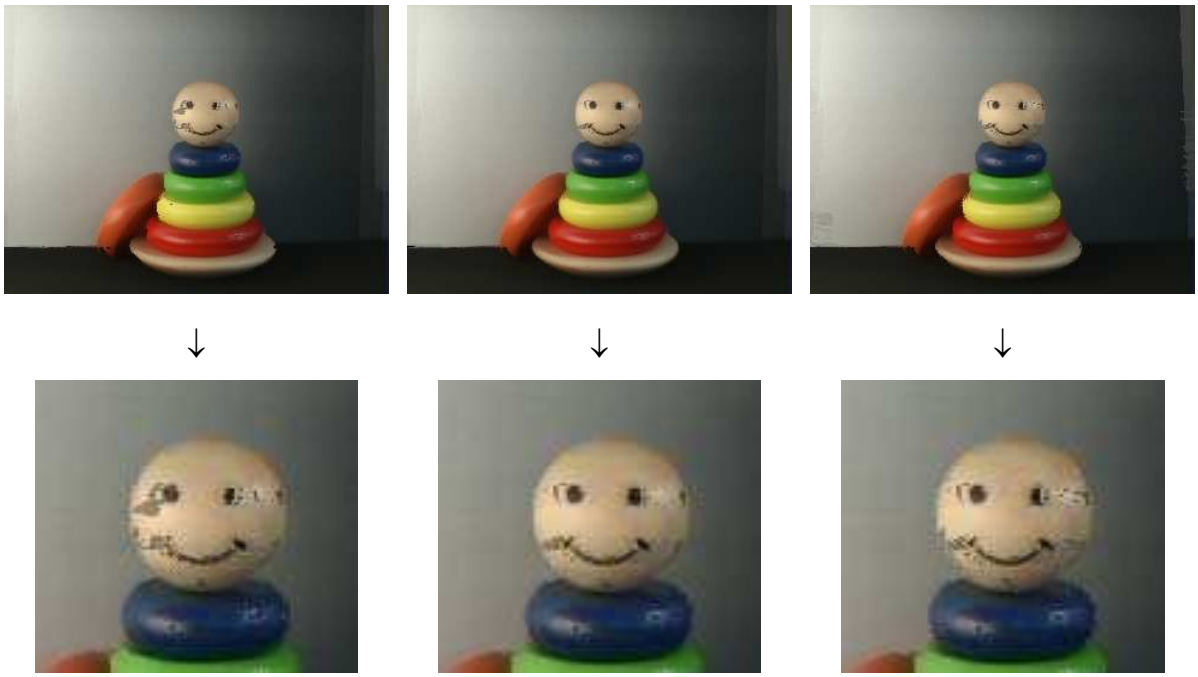
Tant que Card(S)>2 **et** *stable* = = **faux**

Cette méthode est elle aussi facilement implémentable dans le *fragment program* de la méthode standard.

3.4.4 Résultats

Ces deux méthodes ont été testées dans les mêmes conditions que les méthodes précédentes, c'est à dire avec un Athlon 1 GHz et une Nvidia GeForce 6800 GT. L'acquisition des images a été effectuée à l'aide de quatre caméras Tri-CCD Sony DCR-PC 1000E.

Les images de la figure 96 montrent un comparatif des deux méthodes traitant les occlusions avec la méthode standard (sans *mipmapping*). Comme on peut le constater sur ces images, la méthode de *sigma clipping* améliore sensiblement le rendu visuel, notamment sur les discontinuités de texture des objets et sur leurs contours. La méthode dynamique améliore parfois le résultat visuel mais cette amélioration est peu stable et moins significative qu'avec la méthode de *sigma clipping*.



Méthode standard (sans mipmapping) | Sigma Clipping | Méthode dynamique

Figure 96 : comparatif des méthodes standard, de Sigma Clipping et dynamique.

3.4.4.1 Effets locaux

Les points rouges visibles sur la figure 97 représentent les pixels où le meilleur score n'a pas été trouvé durant la première passe de *sigma clipping* mais durant la seconde ou la troisième. On remarque que ces points rouges se situent majoritairement sur les bords des objets c'est-à-dire sur les discontinuités de profondeur. On les retrouve aussi sur les discontinuités de textures comme par exemple autours des yeux ou de la bouche du jouet. Enfin, certains points rouges se trouvent sur les reflets spéculaires particulièrement prononcés, en particulier sur ceux situés sur des surfaces sombres. Cela montre bien le caractère sélectif de cette méthode.

Figure 97 : zones où le sigma clipping prend effet.

La méthode dynamique n'est pas aussi performante dans la sélection des zones où elle intervient. En effet, comme nous le montre la figure 98, les points rouges représentant les pixels où la méthode dynamique est intervenue, sont répartis sur toute l'image, notamment sur le sol de couleur tout à fait homogène. On retrouve toutefois quelques points rouges sur les bords des objets et des textures.

Figure 98 : zones où la méthode dynamique prend effet.

Il apparaît donc clairement que la méthode de *sigma clipping* intervient dans des zones clés de l'image et permet ainsi d'améliorer le rendu. Ce n'est pas tout à fait le cas de la méthode dynamique.

3.4.4.2 Performances

Les tests suivants ont été réalisés sur un Athlon 1 GHz et une carte graphique Nvidia GeForce 6800 GT. Les images traitées et les images générées sont en 320x240. Le tableau suivant illustre les framerates des méthodes traitant les occlusions ainsi que les framerate de la méthode standard.

Nombre de plans	Images par seconde Méthode standard	Images par seconde Sigma-Clipping	Images par seconde Méthode dynamique
10	90	57	60
30	43	28	30
50	30	17	18
100	15	9	9

Compte tenu des opérations supplémentaires à effectuer, les méthodes gérant les occlusions sont moins rapides. Pour une trentaines de plans, les framerates obtenus correspondent toutefois à ceux d'une application visuelle temps réel. Il apparaît que la méthode dynamique est légèrement plus rapide que celle inspirée du *sigma clipping*. La différence est cependant négligeable dans le cas d'une discrétisation de la scène avec une trentaine de plans.

3.4.4.3 Bilan

Les méthodes que nous proposons permettent de gérer les occlusions partielles et notamment les auto-occlusions dans le cas où une majorité de caméras sont cohérentes entre elles. Il s'avère que la méthode de *sigma clipping* obtient de meilleurs résultats visuels que la méthode dynamique. Cela est en partie dû au fait qu'elle sélectionne plus précisément les zones de l'image (et de la scène) susceptibles de correspondre à une occlusion ou à un changement brusque de couleur.

La détection d'occlusions nécessite un traitement supplémentaire au niveau des *fragment programs* (cf. annexe 3) qui correspondent à la partie du pipeline de rendu la plus sollicité dans notre méthode. La baisse de framerate engendrée ne permet pas de combiner ces méthodes avec du *mipmapping* tout en gardant un nombre de plans suffisant pour obtenir un rendu visuel satisfaisant en temps réel. Il faut donc choisir entre performances maximales sans *mipmapping*, performances réduites avec une amélioration du rendu (*mipmapping*) ou performances réduites avec gestion des occlusions.

3.5 Plane Sweep : points forts et faiblesses

Ce chapitre tente de détailler les points forts et les points faibles de la méthode des *Plane Sweep*. Nous nous baserons essentiellement sur les méthodes que nous proposons bien que la plupart de ces remarques soient valables pour toutes les variantes de *Plane Sweep.*

3.5.1 Contraintes sur la scène

La méthode des *Plane Sweep* n'impose aucune contrainte sur la géométrie des objets constituant la scène dès l'instant que l'ensemble de ces objets peut être borné par deux plans. Il est d'ailleurs préférable que ces deux plans *min* et *max* soit suffisamment proches pour minimiser le nombre de subdivisions nécessaires à un rendu correct.

En revanche, les objets de la scène sont soumis à une contrainte colorimétrique puisque la méthode des *Plane Sweep* ne gère pas les reflets spéculaires. Cette contrainte peut paraître très restrictive sur les scènes envisageables cependant un grand nombre de méthodes de reconstruction ou de rendu à base d'images ne gèrent pas ces reflets spéculaires. De plus, les

tests montrent que même si notre méthode ne traite pas explicitement ce genre de reflets, les erreurs engendrées sur les images générées ne sont pas toujours très visibles.

La disposition des objets entre eux n'est pas non plus soumise à des contraintes particulières. Il faut cependant éviter les cas où une zone de la scène n'est visible que par une ou deux des caméras d'acquisition. Pour les occlusions ne concernant qu'un nombre limité de caméras, la méthode que nous proposons convient tout à fait.

Enfin, il se peut que des erreurs apparaissent dans le cas d'un arrière plan uniforme. Dans ce cas, un plan situé entre l'objet supposé correct et la caméra virtuelle fournissent de meilleurs scores que le plan situé sur la surface de l'objet en question. Ce phénomène est illustré sur la figure 99.

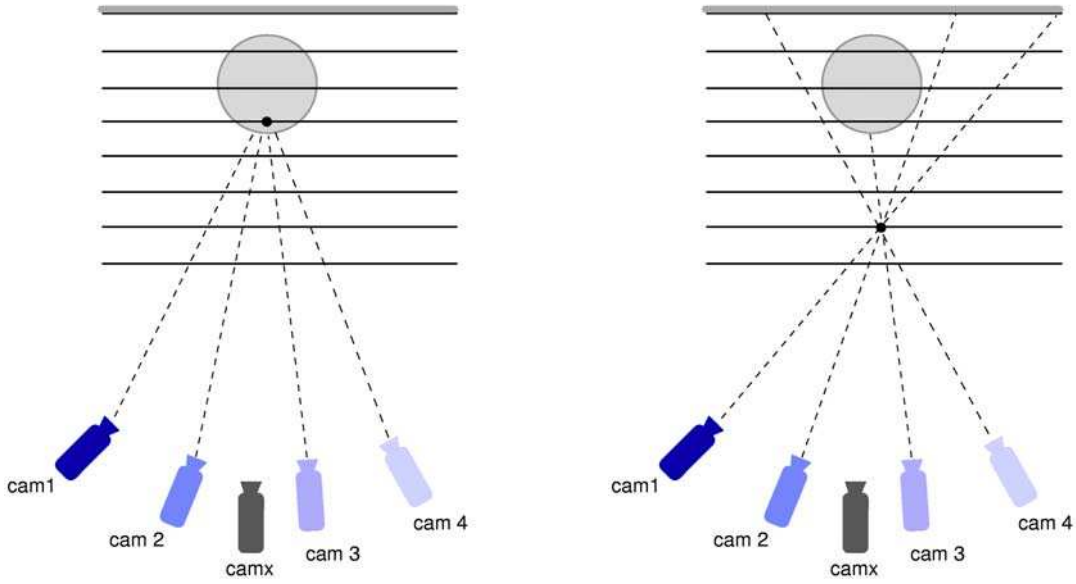


Figure 99 : arrière plan monochrome, source d'erreurs.

Deux solutions sont envisageables, la première consiste à utiliser des outils d'extraction de silhouette comme le font Li et al. [LMS04] avec leur Photo Hulls, les passes de *Plane Sweep* ne sont alors effectuées qu'à l'intérieur des silhouettes. La seconde possibilité est de veiller à ce que le fond ne soit pas uniforme. Un dégradé convient particulièrement bien.

3.5.2 Manipulation et usage

L'avantage majeur de cette méthode est la possibilité qu'elle offre de traiter en direct des images provenant de vidéos et de générer de nouvelles images en temps réel. Les méthodes permettant une telle réactivité sont peu nombreuses. De plus la manipulation de la caméra virtuelle est peu contraignante puisque sa position n'est pas limité aux alentours directs des caméras servant pour l'acquisition.

Comme la grande majorité des autres méthodes, les *Plane Sweep* nécessitent un très bon calibrage et le déplacement malencontreux d'une des caméras pendant le tournage implique le recalibrage immédiat de tout le dispositif.

3.5.3 Résultat visuel

La restitution des détails d'une scène dépend fortement du calibrage des caméra ainsi que de la qualité de leurs optiques. Cependant, même dans des conditions très favorables, le calcul des correspondances n'est pas aussi robuste qu'avec la plupart des méthodes plus lentes. Ce problème se pose moins dans les zones plutôt homogènes où une erreur de calibrage n'a pas trop d'impact.

Ceci dit, compte tenu du temps d'exécution de cette méthode et des résultats obtenus avec les méthodes concurrentes, les résultats visuels obtenus sont tout à fait satisfaisants.

3.5.4 Matériel

La méthode que nous présentons nécessite un dispositif d'acquisition relié à un ordinateur pourvu d'une bonne carte graphique. Ce sont les trois seules composantes importantes pour la faire fonctionner. La quasi-totalité des méthodes de rendu à base d'images ou de reconstruction traitant de la vidéo nécessitent plusieurs ordinateurs ce qui fait de notre méthode l'une des moins onéreuses à faire fonctionner. La carte graphique doit tout de même être de bonne qualité, une carte graphique dédiée aux jeux sur ordinateur convient parfaitement.

De plus, notre méthode ne fonctionne en temps réel qu'avec des images de faible résolution, ce qui correspond parfaitement aux images que l'on peut obtenir avec des webcams. Celles-ci commencent d'ailleurs à fournir des images de bonne qualité (ce qui n'était pas le cas il y a encore quelques années). De ce fait, notre méthode est très facilement utilisable par le grand public compte tenu d'une part du prix du matériel nécessaire et d'autre part de sa simplicité de mise en oeuvre.

Notons que le fait de n'utiliser qu'un ordinateur pour l'acquisition des vidéos (en USB 2.0 ou en firewire) peut poser un problème de synchronisation des caméras qui n'apparaît pas nécessairement si l'acquisition est effectuée par plusieurs ordinateurs qui transmettent les images par le réseau. En effet, lors d'un transfert par réseau, toutes les caméras (à peu près synchronisées) peuvent commencer à transmettre en se partageant la bande passante. Pour des liaisons série, la $n^{ième}$ caméra doit attendre la fin de la réception de la $(n\text{-}1)^{ième}$ caméra avant de pouvoir commencer à transmettre. Il se peut donc qu'entre la transmission de la première et de la dernière caméra, il se soit produit suffisamment de temps pour qu'une nouvelle acquisition ait eu lieu. Les images reçues sont alors désynchronisées. Dans le cas d'un nombre restreint de caméras ayant par ailleurs des images de faible résolution, il se peut que ce problème ne soit pas trop gênant.

Seconde partie
Rendu stéréoscopique à base de Vidéos et Réalité Virtuelle

4 Plane Sweep et Rendu Stéréoscopique

Certaines applications graphiques utilisent des images stéréoscopiques (i.e. permettant de voir en relief). C'est le cas de la plupart des dispositifs de réalité virtuelle ainsi que d'autres applications plus spécifiques. Nous proposons dans ce chapitre une adaptation de la méthode des *Plane Sweep* pour un rendu stéréoscopique rapide. Avant de détailler cette amélioration des *Plane Sweep*, nous rappelons les principes de bases concernant la création d'une paire d'images stéréoscopiques.

4.1 Perception du relief

Contrairement à une idée répandue, il existe de nombreuses façons de percevoir le relief, notamment avec la vision monoculaire. Nous allons néanmoins particulièrement détailler la vision stéréoscopique qui est le moyen de percevoir le relief le plus puissant.

4.1.1 Vision monoculaire

Il existe de nombreux indices permettant de percevoir le relief en vision monoculaire pour une scène statique. Les plus évidents sont les indices de perspective que l'on peut extraire de lignes de fuite. Les lumières et les ombres sur chaque élément d'une scène fournissent également de nombreux indices de profondeur, notamment en révélant les lignes de fuite. Les gradients de textures peuvent aussi donner ce genre de renseignements.

Il existe aussi des indices permettant d'ordonner la profondeur de plusieurs objets. En effet, si un objet en occulte un autre, c'est qu'il est devant lui. Cette notion s'étend même aux milieux participants comme la brume ou le brouillard. Si ce dernier est relativement homogène, il dissimulera d'avantage un objet situé loin de l'observateur. Il est aussi possible de placer l'un par rapport à l'autre deux objets usuels dont la taille est connue par l'observateur.

L'accommodation de l'œil sur un objet est aussi un indice précieux concernant sa position dans l'espace. En effet, même si la mise au point sur cet objet s'effectue inconsciemment, nous sommes capables d'en extraire de l'information. L'accommodation permet en outre de distinguer la profondeur de deux objets situés sur deux plans différents. Si l'un des objets est net, l'autre sera flou.

Enfin, la parallaxe de mouvement induite par le mouvement de l'observateur lui fournit plusieurs points de vue de la même scène. Cette perception du relief est donc fortement reliée à la perception stéréoscopique.

Pour plus d'informations, le lecteur peut se référer au Traité de la Réalité Virtuelle de Fuchs et al. [FMP01].

4.1.2 Vision stéréoscopique

La vision stéréoscopique est basée sur un principe simple, les deux yeux reçoivent chacun des images semblables, mais pas identiques. Le cerveau se charge de fusionner ces deux images et la « reconstruction » opérée aboutit à des renseignements sur la géométrie et la disposition des objets observés.

Dans le cas d'une image stéréoscopique, le procédé est le même. L'image stéréoscopique nécessite un support comme un écran de cinéma, d'ordinateur ou une simple feuille de papier. Sur ce support vont figurer deux images, une pour l'œil droit et une pour l'œil gauche. Il existe alors de nombreux dispositifs pour associer la bonne image à chacun des deux yeux (cf. annexe 2).

Etant donné un point *M* de la scène, ce point va figurer sur les deux images et comme le montre la figure 100, sa position sur ces deux images est en général différente. La distance entre le projeté de *M* sur l'image de droite et le projeté de *M* sur l'image de gauche est appelée parallaxe.

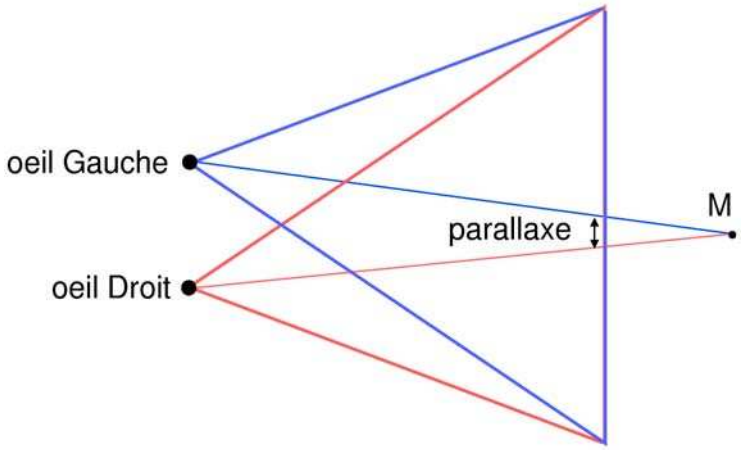


Figure 100 : parallaxe.

On parle parfois d'angle de parallaxe pour décrire l'angle formé par les deux projetés du point *M* avec le milieu du segment séparant les deux yeux (figure 101).

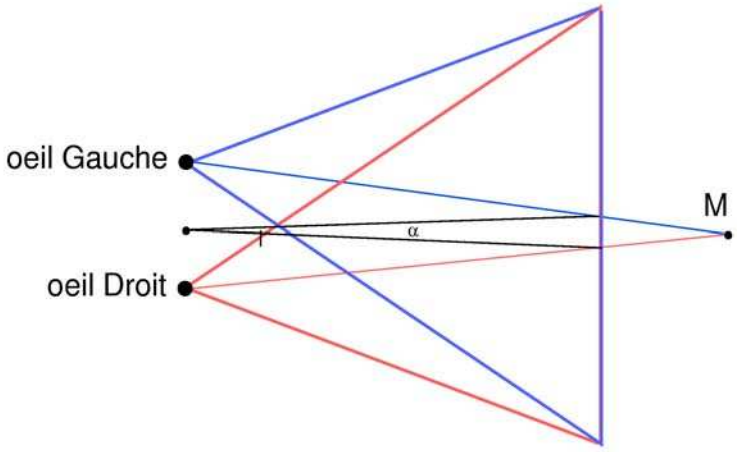


Figure 101 : angle de parallaxe.

4.1.2.1 Parallaxe horizontale

En position normale, les yeux des êtres humains se situent (sauf cas particuliers) sur un plan horizontal. La restitution stéréoscopique d'une scène en images de synthèse doit donc tenir compte de cette information en générant les deux images avec des caméras placées

judicieusement. La position d'une caméra est définie par la position de son centre de projection, l'orientation de son axe optique et l'orientation du vecteur définissant la direction de la hauteur. Dans le cas d'images stéréoscopiques, il faut donc que les vecteurs définissant la hauteur soit perpendiculaires à la droite passant par les centres des caméras. Notons que l'orientation de l'axe optique est directement définie par la normale du plan image de la caméra.

La distance séparant les deux caméras (ou séparant nos deux yeux) s'appelle « distance interoculaire », nous la noterons *dio* par la suite. Elle est estimée à 65 mm en moyenne. Notons aussi que la parallaxe horizontale correspond à la composante horizontale de la parallaxe (cette notation suppose que l'observateur se tient en position verticale).

On parle de parallaxe positive (non croisée) quand l'objet virtuel se situe derrière l'écran. Comme nous le montre la figure 102, le projeté d'un tel objet en direction de l'œil droit se trouve donc à droite de son projeté en direction de l'œil gauche.

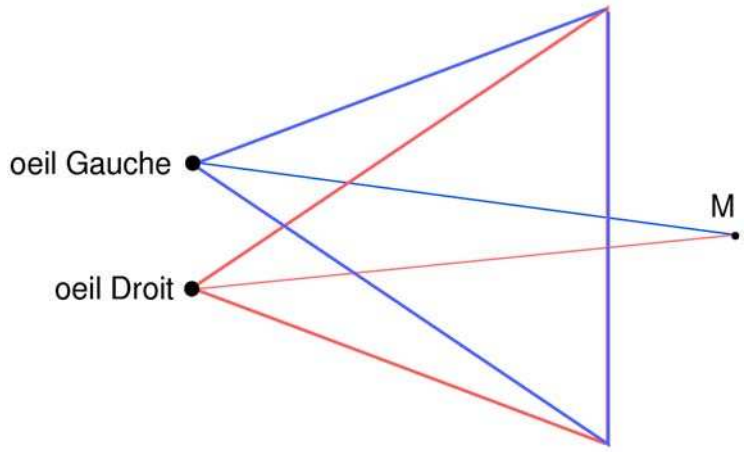


Figure 102 : parallaxe positive.

De même, on parle de parallaxe négative (croisée) quand l'objet virtuel se situe devant l'écran. Les points homologues gauche et droit sont affichés respectivement à droite et à gauche (figure 103).

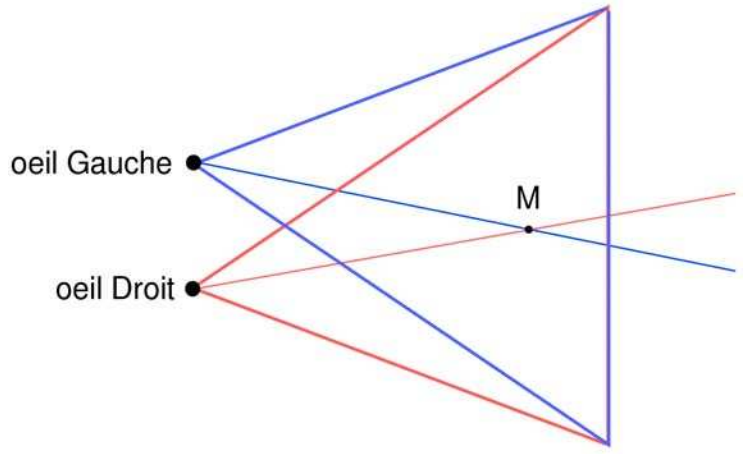


Figure 103 : parallaxe négative.

Enfin, on parle de parallaxe nulle quand l'objet virtuel se trouve sur le plan de l'écran. Les composantes vues par l'œil droit et l'œil gauche se superposent sur l'écran (figure 104).

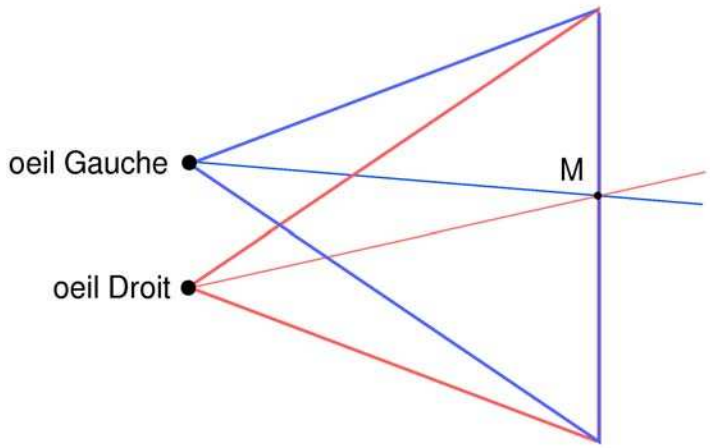


Figure 104 : parallaxe nulle.

4.1.2.2 Parallaxe verticale

Les paragraphes précédents montrent bien qu'il est naturel d'avoir affaire à de la parallaxe horizontale et que la parallaxe verticale est préjudiciable. Lors de la création d'images de synthèses stéréoscopiques, il faut savoir qu'un mauvais paramétrage des caméras peut engendrer de la parallaxe verticale. En effet, il parait plutôt intuitif de faire converger les axes optiques des deux caméras vers le centre de la scène (figure 105), cela semble apporter un supplément d'informations, notamment pour les scènes proches de l'observateur. En réalité, il est préférable de paramétrer les caméras de telle sorte que leur axe optique soient parallèles (figure 106). Si ce n'est pas le cas, il y a alors apparition de parallaxe verticale.

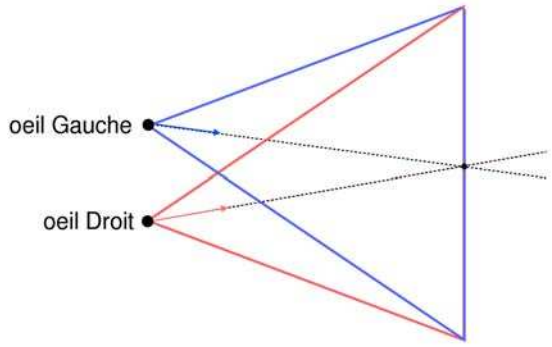


Figure 105 : axes optiques convergents.

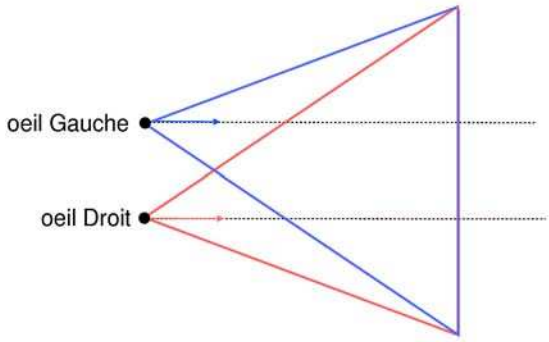


Figure 106 : axes optiques parallèles.

Prenons l'exemple d'un rectangle inscrit dans un plan vertical face aux deux caméras. En considérant que les axes optiques des caméras convergent tous les deux vers le centre du rectangle, il apparaît alors que le rectangle n'est pas parallèle aux plans images des caméras et que par conséquent, sur une carte de profondeur, le coin gauche du rectangle apparaît plus proche que le coin droit pour la caméra de gauche (et réciproquement pour l'œil droit). Par effet de perspective, le rectangle est donc projeté sous forme de trapèzes sur les deux images. Pour la caméra de gauche, le coté gauche du rectangle apparaît alors plus gros que le coté droit. Ce phénomène est symétrique par rapport aux deux caméras donc lors de la

superposition des deux images, il apparaît clairement qu'en plus de la parallaxe horizontale s'est introduite de la parallaxe verticale (figure 107).

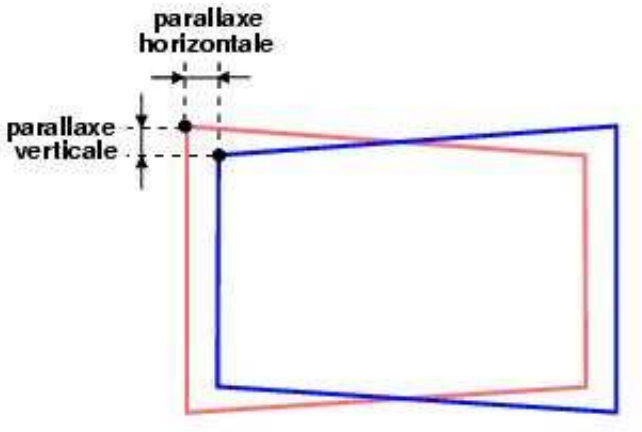


Figure 107 : apparition de parallaxe verticale.

Le cerveau a du mal à fusionner ce genre d'images stéréoscopiques comportant de la parallaxe verticale c'est pourquoi il est absolument nécessaire de positionner les caméras de telle sorte que leurs axes optiques soient parallèles.

4.2 Rendu stéréoscopique en synthèse d'images

Comme nous venons de le voir dans les paragraphes précédents, les axes optiques des caméras doivent être parallèles. Une première méthode de rendu consiste donc à générer une vue de la scène puis à décaler la caméra afin de générer une seconde vue. Les deux vues sont alors fusionnées en les décalant de la même distance que celle qui sépare les deux caméras. Dans ce cas là, les bords de l'image finale où ne figure qu'une des deux images sont perdus. Une autre approche consiste à utiliser une pyramide de projection non symétrique. La figure 108 illustre les variables nécessaires pour paramétrer un système de deux caméras.

Un des paramètres déterminant de ce système est la position du plan image. Ce plan image défini l'ensemble des points de la scène qui, lors de la restitution, seront perçus comme étant situés sur le support de l'image stéréoscopique. En effet, les points situés sur ce plan auront la même position (en pixel) sur l'image finale, qu'ils soient destinés à l'œil droit ou à l'œil gauche. Ces points auront donc une parallaxe nulle.

La variable *W* correspond à la largeur de l'écran (ou du support des images stéréoscopiques en général) dans la scène. La variable α représente le champ de vision du système. Les paramètres *near* et *far* servent à définir une caméra dans certaines bibliothèques graphiques comme OpenGL ou DirectX. Il en est de même pour les paramètres *min* et *max*.

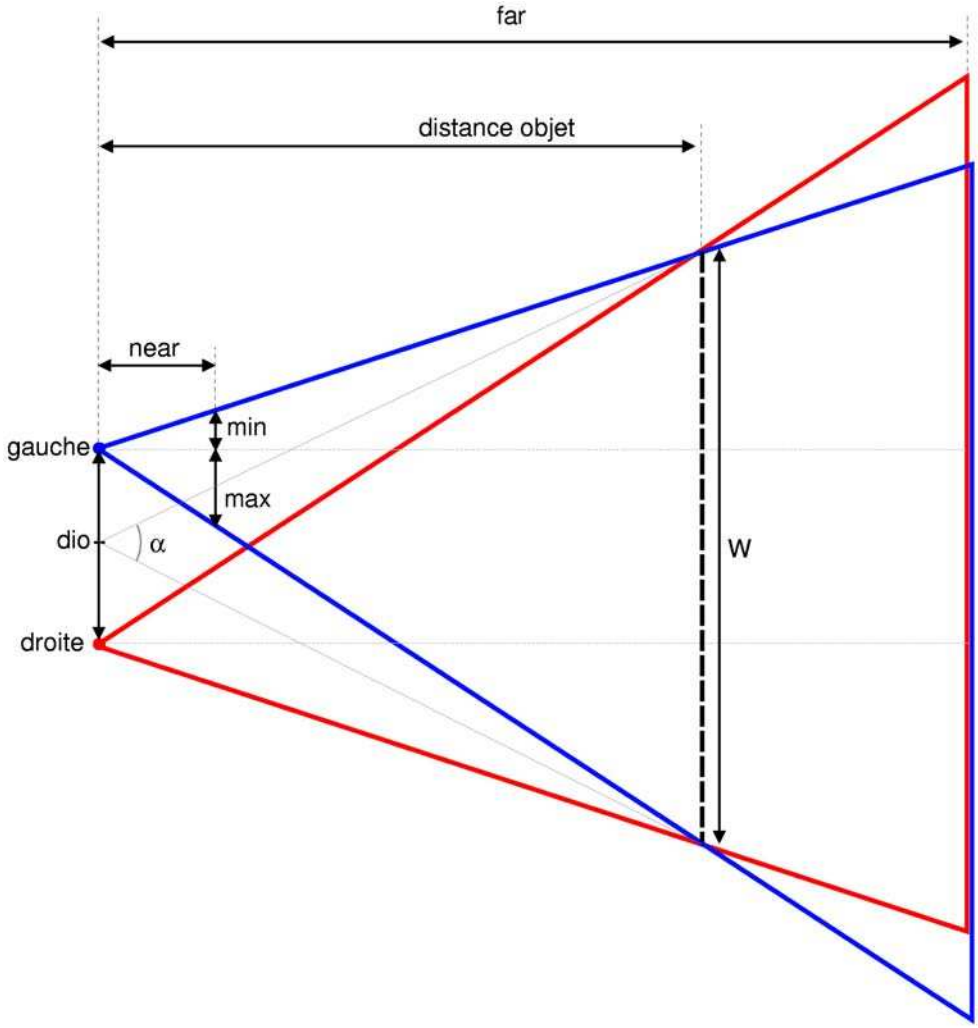


Figure 108 : système de deux caméras stéréoscopiques.

La distance interoculaire, notée *dio*, correspond à la distance entre les deux caméras. Si l'effet stéréoscopique veut rendre compte d'une scène vue par un humain, il faut alors espacer les deux caméras de 6,5 cm, à un facteur d'échelle près associé à la scène. Ces 6,5 cm correspondent à la distance interoculaire moyenne. En revanche, si l'effet stéréoscopique souhaité est de simuler la vue d'une fourmi ou d'un géant (figure 109 et figure 110), alors il faut paramétrer la variable *dio* en conséquence. Fuchs et al. [FMP01] conseillent de respecter au mieux le rapport $f = 20dio$.

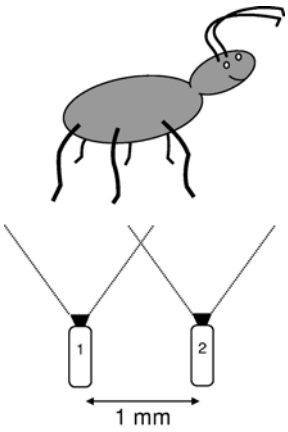


Figure 109 : caméras virtuelles proches.

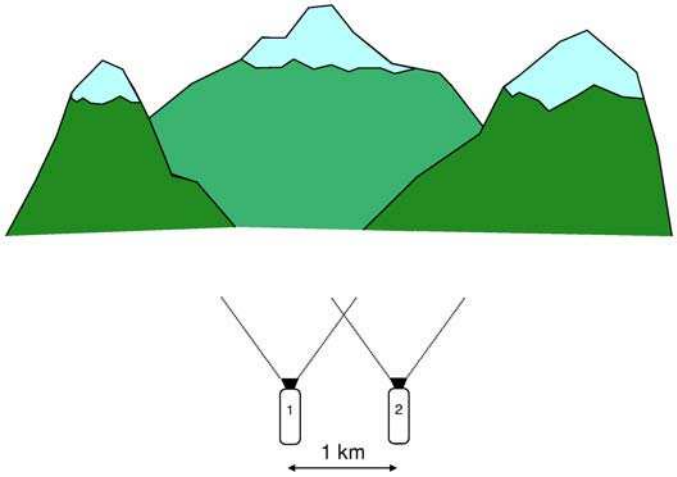


Figure 110 : caméras virtuelle éloignées.

D'autres paramètres propres au système de caméras n'apparaissent pas sur ce schéma, tels que la position du centre des caméras, l'orientation des axes optiques, le vecteur indiquant la verticale, la largeur et la hauteur de l'écran en pixels ... Voici les relations que l'on peut déduire de la figure 108:

$$\min = near\frac{W - dio}{2f}$$

$$\max = near\frac{W + dio}{2f}$$

$$f = \frac{W}{2\tan(\frac{\alpha}{2})} \Leftrightarrow W = 2f.\tan(\frac{\alpha}{2}) \Leftrightarrow \alpha = 2.\arctan\left(\frac{W}{2f}\right)$$

D'après ces relations, il est alors possible de déduire les procédures à suivre pour mettre à jour tous les paramètres lorsque l'un d'entre eux est modifié. Ces relations sont directement applicables sur les bibliothèques graphiques comme OpenGL, notamment avec la fonction `glFrustum` (figure 111). Les paramètres `left` et `right` correspondent aux variables *min* et *max* (à permuter selon la caméra). Les paramètres `top` et `bottom` se calculent à partir de `left` et `right` en conservant le rapport hauteur/largeur de l'image.

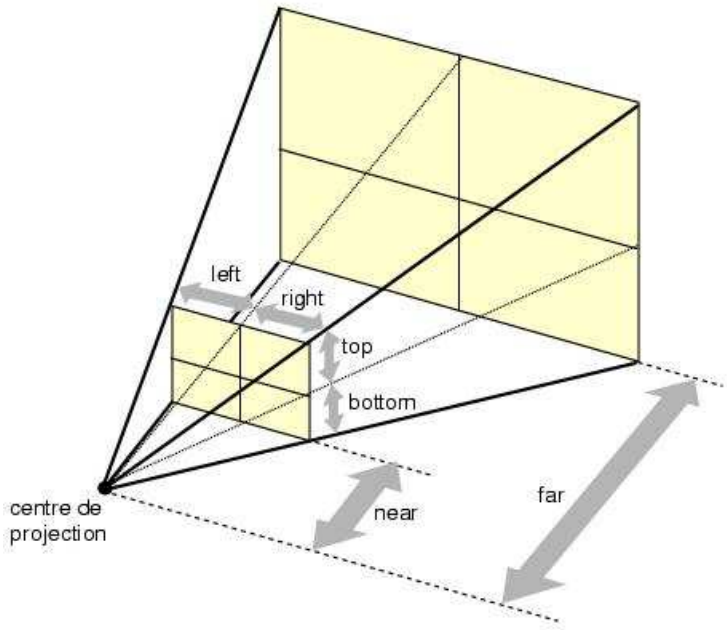


Figure 111 : pyramide de projection (glFrustum).

4.3 Plane Sweep et rendu stéréoscopique

Dans le cas de rendu stéréoscopique, la plupart des méthodes doivent effectuer deux rendus, un pour chaque œil. Ce double rendu n'est pas trop contraignant pour les méthodes ne gérant que les scènes statiques dont le rendu est déjà temps réel. Par contre elle l'est davantage pour les méthodes traitant de la vidéo et en particulier pour les méthodes en direct, qui utilisent déjà la totalité des ressources disponibles.

Il est intéressant de noter qu'une grande partie des informations concernant la scène peuvent être partagées pour les deux vues, comme par exemple un maillage ou l'illumination diffuse et ambiante. Les méthodes de reconstruction ainsi que certaines méthodes de la famille des *Visual Hulls* peuvent facilement partager ces informations une fois la reconstruction effectuée. Ce n'est pas le cas pour toutes les méthodes de rendu à base d'images ou pour les méthodes utilisant des cartes de profondeur dépendant souvent du point de vue. Ces méthodes doivent alors calculer les deux images stéréoscopiques de façon indépendante.

Il se trouve que la méthode des *Plane Sweep* est spécialement bien adaptée au rendu stéréoscopique puisque les calculs des scores et des couleurs sont faits localement. Ils peuvent ainsi être partagés par plusieurs caméras virtuelles.

4.3.1 Modification de l'algorithme classique

Comme illustré sur la figure 112, les informations des scores et des couleurs calculées sur un plan *D* donné sont des informations locales à la scène et donc communes aux deux caméras. C'est la projection de *D* sur les deux caméras qui fait que le rendu sera différent pour chacune d'entre elles. Sachant que le calcul du score et des couleurs est la tâche principale

de la méthode des *Plane Sweep*, le gain de temps en partageant les données devient alors évident.

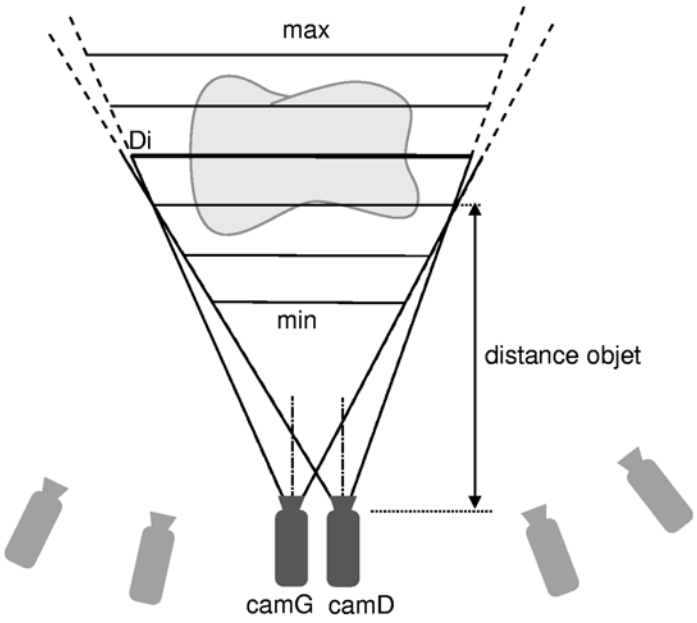


Figure 112 : Plane Sweep et stéréoscopie.

Soit Cam_D et Cam_G les deux caméras virtuelles et I_D et I_G leur image respective. L'algorithme des *Plane Sweep* devient alors pour chaque plan un algorithme en deux passes. Durant la première passe, les scores et les couleurs calculées sur un plan *D* sont stockées dans deux textures. Durant la seconde passe, ces deux textures sont projetées sur les deux caméras virtuelles. L'algorithme de rendu devient donc :

1 Initialiser les scores et les couleurs de I_D et I_G

2 Pour chaque plan *D* entre *Dmin* et *Dmax*

3 • Projeter les images de référence sur *D*

4 • Faire une passe de rendu de *D* sur deux textures *texScore* et *texColor* stockant respectivement les scores et les couleurs calculées sur le plan *D*.

5 • Copier *texScore* et *texColor* sur *D*

6 • Projeter *D* multitexturé sur Cam_D puis sur Cam_G

7 • Mettre à jour les score et les couleurs sur I_D et I_G

8 Afficher I_D et I_G

Comme pour les systèmes de caméras décrits dans les chapitres précédents, les deux caméras utilisent une pyramide de projection asymétrique. Il est donc nécessaire de déterminer une distance objet définissant l'ensemble des points de la scène de parallaxe nulle situés sur le plan image.

4.3.2 Implémentation

Cette méthode a été implémentée en C++ sous Linux en utilisant les bibliothèques OpenGL et GLSL. La première passe de rendu, durant laquelle les scores et les couleurs sont calculés, doit être effectuée *off-screen*, c'est-à-dire que le résultat ne doit pas apparaître à l'écran. En effet, il n'est pas possible d'effectuer le rendu directement sur l'une des deux images puisque cette projection engendrerait une mise à jour des scores et des couleurs. On ne pourrait plus alors reprojeter ces données sur la seconde caméra.

Une première méthode consiste à effectuer ce rendu *off-screen* à l'aide de *p-buffers* ce qui nécessite un transfert de données du *frame-buffer* vers les textures *texScore* et *texColor*. L'emploi de *Frame Buffer Object* (FBO) et de *Multiple Render Target* (MRT) est beaucoup plus approprié puisqu'il évite ce genre de transfert coûteux en temps. En pratique la seconde méthode est nettement plus rapide sauf si la carte graphique utilisée ne gère pas en hardware les FBO et les MRT.

Cette implémentation permet donc d'éviter des transferts entre le *frame-buffer* et les textures ainsi que tout transfert entre la carte graphique et la mémoire principale de l'ordinateur.

4.3.3 Résultats

Cette adaptation des *Plane Sweep* pour la création d'images stéréoscopiques est particulièrement efficace. Voici quelques images pour illustrer nos résultats suivis des performances obtenues.

4.3.3.1 Résultats en images

L'acquisition des images a été effectuée à l'aide de quatre caméras Tri-CCD Sony DCR-PC 1000E. La figure 113 illustre une paire d'images stéréoscopiques obtenue avec notre méthode. Ces images ont été calculées en subdivisant la scène en 30 plans. Il est possible de voir une image en relief par vision croisée (en louchant).

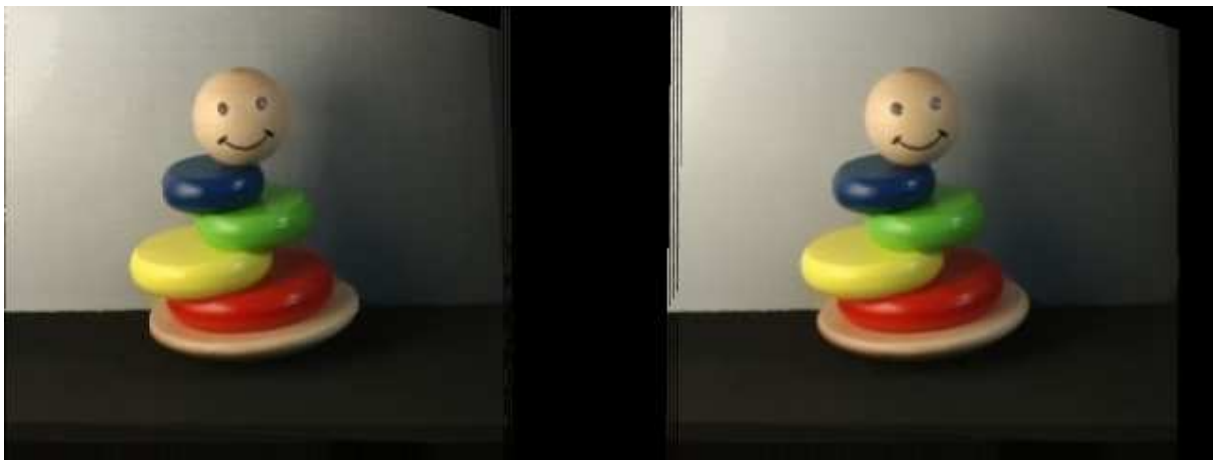


Figure 113 : images stéréoscopiques générées.

La figure 114 représente une séquence d'images stéréoscopiques. Comme pour les images de la figure 113, les images en entrées ainsi que les images générées sont au format 320x240.

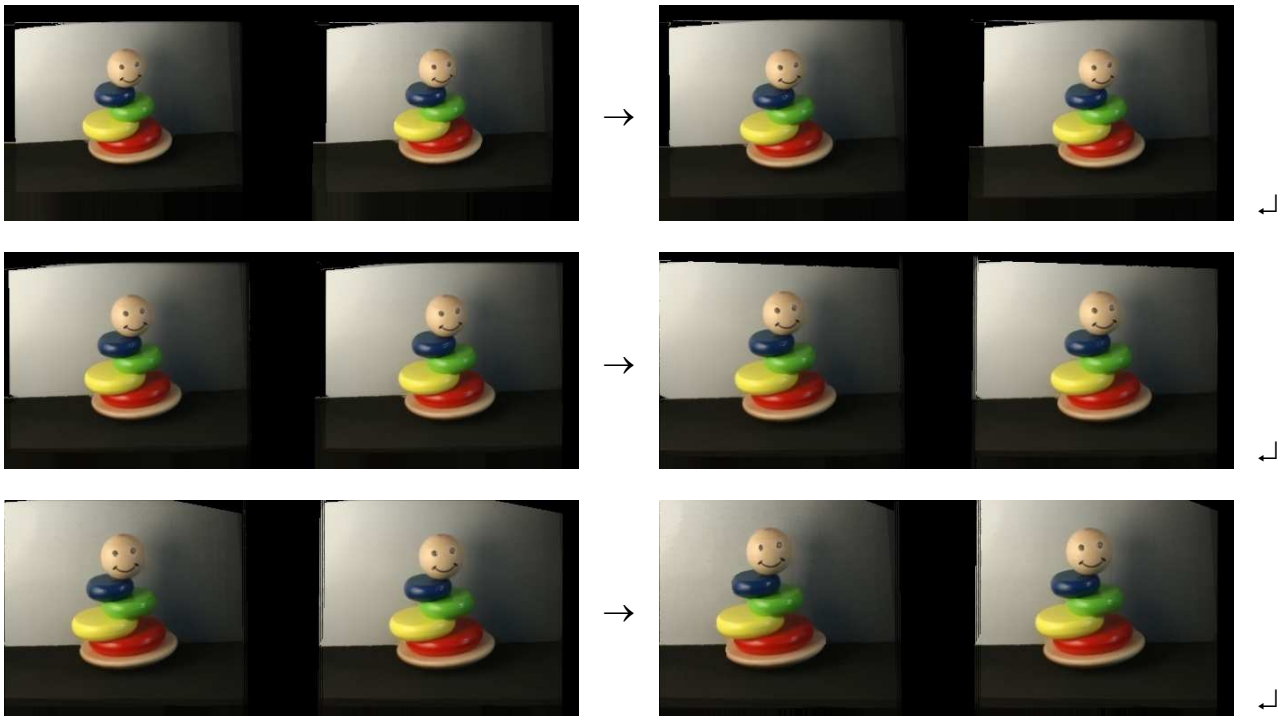


Figure 114 : séquence d'images stéréoscopiques.

4.3.3.2 Performances

Les tests suivants ont été réalisés sur un Athlon 1 GHz et une carte graphique Nvidia GeForce 6800 GT. Les images traitées et les images générées sont en 320x240 ce qui correspond à une image 640x240 lorsque les images stéréoscopiques sont mises côte à côte. Le tableau suivant illustre les framerate de notre méthode utilisée d'une part en mode standard en ne générant qu'une seule image et d'autre part en mode stéréoscopique et donc en générant une paire d'image stéréoscopique. Plusieurs degrés de discrétisation sont utilisés pour illustrer cette comparaison.

Nombre de plans	Images par seconde méthode mono	Images par seconde méthode stéréo
10	90	75
30	43	38
50	30	25
100	15	13

Les résultats montrent que la création d'une paire d'images stéréoscopiques n'entraîne qu'une chute du framerate de 15% au lieu des 50% attendus en effectuant deux fois le rendu. Autrement dit, la création de la seconde vue est peu coûteuse. Compte tenu du mode de création de cette seconde vue, il est clair que la création d'une troisième ou d'une quatrième vue n'ajouterait pas beaucoup de calculs supplémentaires, permettant ainsi un traitement multiutilisateur.

4.3.3.3 Bilan

L'adaptation de la méthode des *Plane Sweep* pour la création d'images stéréoscopiques bénéficie de tous les avantages du rendu mono tout en profitant d'une accélération due au partage d'informations sur chaque plan. Cette accélération est conséquente, ne nécessite aucun matériel supplémentaire et est facilement implémentable pour une personne qui disposerait de la méthode standard (mono).

Comme pour la méthode standard, notre méthode fonctionne correctement même avec un processeur peu performant comme c'est le cas dans nos tests. L'essentiel est d'avoir une bonne carte graphique.

Enfin, il est important de noter que les imperfections visuelles ne se situent pas nécessairement aux mêmes endroits sur les deux images. Ainsi, la fusion des deux images par le cerveau réduit l'impact des imperfections, les images en relief paraissent moins bruitées que les deux images prises une par une.

5 Réalité Virtuelle et Mouvements Pseudoscopiques

Burdea et Coiffet [BC93] définissent un système de réalité virtuelle comme « *une interface qui implique de la simulation temps réel et des interactions via de multiples canaux sensoriels. Ces canaux sensoriels sont ceux de l'homme* ». Cette définition souligne l'aspect d'immersion et d'interactivité. Pour offrir une sensation d'immersion maximale, la réalité virtuelle se doit de stimuler tous les sens de l'être humain et notamment la vue, domaine que nous allons traiter plus particulièrement. Le caractère interactif d'un système de réalité virtuelle peut lui aussi être en relation avec certains de ses sens et là encore, en particulier avec la vue. Le système peut par exemple s'adapter à l'utilisateur en captant la direction de son regard ou bien en adaptant les images générées au point de vue de l'observateur. C'est ce cas que nous allons traiter en l'appliquant à la vision en relief.

Comme nous l'avons défini dans le chapitre 4.1.2, une image stéréoscopique est composée de deux images, une pour chaque œil. En utilisant un appareillage adéquat (lunettes obturatrices, polarisées, etc.), un observateur peut fusionner les deux images et percevoir du relief. De telles images sont réalisables sur de nombreux supports tels que des écrans d'ordinateur comme ceux décris dans Pape et al. [PAB99], des bureaux immersifs comme ceux présentés par Czernuszenko et al. [CPS97] ou bien les écrans auto-stéréoscopiques comme dans Dodgson et al. [DML96]. Il existe d'autres supports disposant de surfaces plus grandes, éclairées par des vidéo-projecteurs comme les visiocubes tels que le CAVE [CSD93] de Carolina et al. Quelque soit la taille de ce genre de dispositif, l'utilisateur est susceptible de se mouvoir devant ces supports. Si l'image affichée n'est pas modifiée, l'observateur verra les objets de la scène se déplacer de façon anormale et sa perception du relief s'en trouvera perturbée. Ce type de déformations est connu sous le nom de mouvements pseudoscopiques et est largement décrit dans Fuchs et al. [FMP01]. Si la position de l'observateur est suivie, il est possible de corriger les images.

Nous proposons dans cette partie une méthode permettant de corriger ces mouvements pseudoscopiques. Cette méthode est applicable d'une part sur des méthodes classiques de rendu d'images de synthèse et d'autre part sur la méthode des *Plane Sweep*. Les intérêts principaux de cette méthode sont sa simplicité et son efficacité.

5.1 Mouvements pseudoscopiques en réalité virtuelle

Cette section propose une description des phénomènes observables face à une image stéréoscopique, notamment les mouvements pseudoscopiques ainsi qu'une de leurs conséquences, la position orthostéréoscopique.

5.1.1 Mouvements pseudoscopiques

Une image stéréoscopique est composée de deux images décrivant la même scène, mais d'un point de vue légèrement différent. Un point de la scène visible par les deux yeux de l'observateur sera présent sur les deux images, mais pas au même endroit. A l'aide d'un dispositif (lunettes obturatrices, polarisées, etc.), l'utilisateur va associer une image à chaque œil et percevoir du relief en fusionnant les deux images. Comme nous l'avons remarqué précédemment, lorsque l'observateur se déplace par rapport à l'écran, il constate que la scène se modifie d'une façon qui n'est pas naturelle. La figure 115 illustre ce phénomène en

décrivant un déplacement de l'observateur en direction de l'écran. En même temps que l'observateur avance, il voit l'objet se rapprocher vers lui. Normalement, il devrait juste avoir l'impression de s'être rapproché d'un objet immobile.

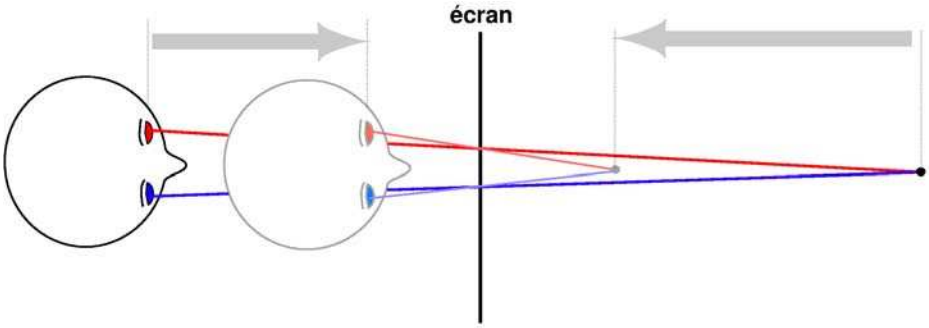


Figure 115 : mouvements pseudoscopiques vers l'avant.

De la même manière, lorsqu'il effectue un déplacement latéral, l'objet lui semble se déplacer latéralement comme le montre la figure 116. Là encore, il n'a pas la sensation d'avoir à faire à un objet immobile.

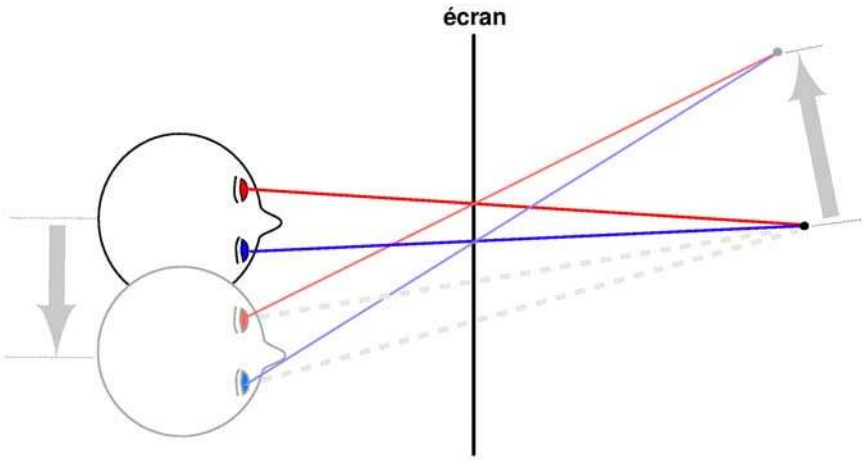


Figure 116 : mouvements pseudoscopiques latéraux.

Ces mouvements des objets virtuels sensés être immobiles sont appelés mouvements pseudoscopiques. Ils ont une influence néfaste sur la sensation de relief perçue par l'observateur, c'est la raison pour laquelle il est nécessaire de les supprimer. En pratique, on constate qu'un observateur peut se permettre de bouger la tête sans subir trop de déformations lorsque l'amplitude du mouvement de la tête est petite par rapport à la taille de l'écran. Ainsi, dans le cas de plusieurs observateurs assis devant un grand écran, il n'est pas nécessaire de corriger l'image. Par contre, cela devient impératif pour un observateur bougeant la tête devant son écran d'ordinateur ou pour un observateur se déplaçant dans un visiocube ou devant un écran de grande taille.

5.1.2 Position orthostéréoscopique

Il s'avère que lors du déplacement de l'observateur, les mouvements pseudoscopiques pour deux points distincts de la scène ne sont pas identiques et sont d'autant plus différents qu'ils

sont éloignés l'un de l'autre. La figure 117 illustre un segment perpendiculaire à l'écran qui est dessiné vu de près puis vu de loin. On constate que lorsque l'observateur se rapproche de l'écran, non seulement le segment s'en rapproche lui aussi mais on constate également que la longueur du segment diminue. Les mouvements pseudoscopiques provoquent donc des distorsions des distances sur les objets de la scène virtuelle. La figure 118 nous montre un segment positionné parallèlement à l'écran. Dans ce cas, pour le même déplacement de l'observateur que sur la figure 117, la taille du segment ne diminue pas, elle augmente plutôt. Les perceptions de la largeur et de la profondeur d'un objet virtuel ne sont donc pas affectées de la même manière par le déplacement de l'observateur.

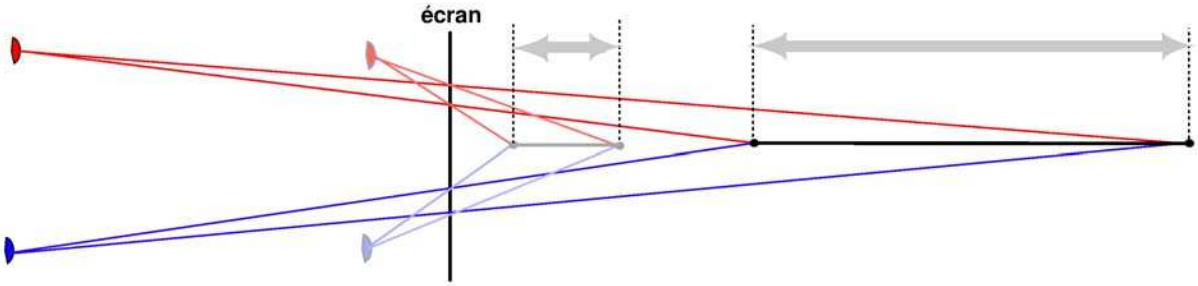


Figure 117 : distorsion de la profondeur.

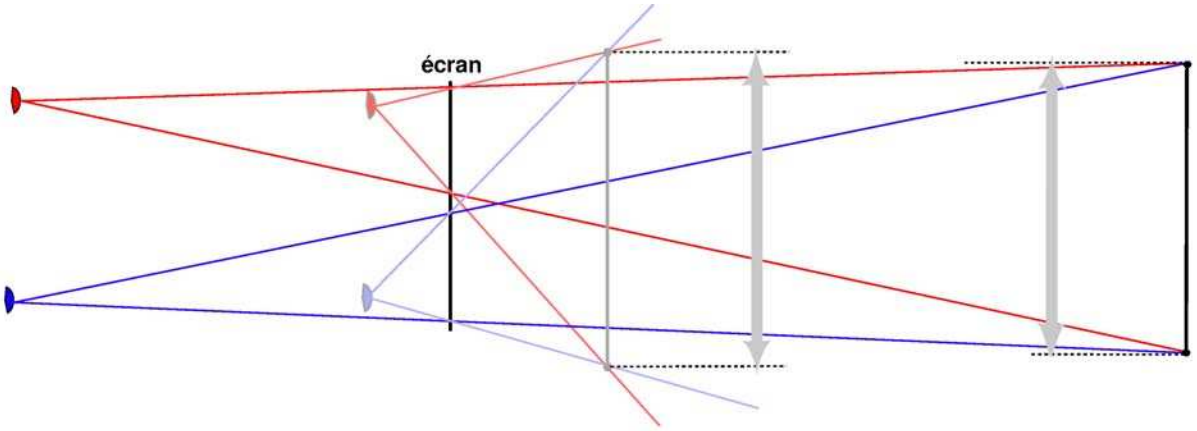


Figure 118 : distorsion de la largeur (et de la hateur).

Ainsi les mouvements pseudoscopiques sont à l'origine de déformations des objets virtuels : les proportions des objets virtuels ne sont plus toujours perçues par l'observateur de façon conforme à celles définies dans la scène virtuelle. En fait, il n'existe qu'une position permettant de restituer la scène dans les proportions désirées, il s'agit de la position orthostéréoscopique.

On peut calculer cette position de plusieurs façons mais la manière la plus intuitive de la définir est de considérer que le système de deux caméras chargé de l'acquisition ou de la synthèse des images stéréoscopiques (dans la scène virtuelle) doit se comporter de la même façon que les deux yeux de l'observateur (dans la scène réelle). Nous reviendrons sur ce point dans les chapitres suivants.

Finalement, nous constatons que non seulement les objets se mettent à bouger quand l'observateur bouge (mouvements pseudoscopiques), mais qu'en plus ils se déforment et ne gardent pas forcément leurs proportions originales. Il a donc deux rectifications à effectuer.

5.2 Correction des mouvements pseudoscopiques : méthodes existantes

Le problème des mouvements pseudoscopiques a été abordé plusieurs fois et résolu de manières différentes. Cruz-Neira et al. [CSD93] corrigent les images pour un visiocube et proposent une matrice de projection associant un point de la scène à une position sur l'écran. Cette matrice est modifiée en fonction de la position de l'observateur. Le point faible de cette méthode est qu'elle ne permet pas un accès facile aux paramètres stéréoscopiques comme l'angle d'ouverture du champ de vision ou bien la distance objet.

Deux autres méthodes proposées par Fuchs et al. [FMP01] permettent de générer des images stéréoscopiques et de les ajuster à une vision orthostéréoscopique. Elles ne sont cependant valables que pour un observateur centré devant son écran, éventuellement trop près ou trop loin. La première méthode procède par décalage des images générées. L'amplitude de ce décalage est calculée pour restituer une position orthostéréoscopique mais ce faisant, on perd les pixels sur les bords de l'écran qui ne sont recouverts que par une seule image. Cette méthode est très répandue pour les films stéréoscopiques tournés avec des objectifs standard (sans objectif à bascule ou à décentrement) mais l'est moins pour les animations en images de synthèse. En effet, dans ce cas, il est possible d'éviter la perte des pixels situés près des bords de l'écran en utilisant des pyramides de projection non-symétriques. La seconde méthode consiste à faire converger les axes optiques vers « le centre de la scène virtuelle » et s'épargne ainsi un décalage des images. L'inconvénient de cette méthode est qu'elle génère de la parallaxe verticale et donc des images plus difficiles à fusionner. De plus, ces deux méthodes ne corrigent pas les mouvements pseudoscopiques pour des déplacements latéraux.

Enfin, Raskar [R00] propose une méthode de correction d'images projetées par vidéo-projecteurs décentrés. En plus de corriger les mouvements pseudoscopiques, cette méthode corrige la déformation de l'image projetée par un vidéo-projecteur qui ne serait pas positionné correctement face à l'écran mais un peu de travers. Cette méthode procède en deux étapes. Dans un premier temps, elle calcule l'image en modifiant les paramètres de projection de la caméra virtuelle en fonction des caractéristiques de l'écran (dimensions et position) et de la position de l'observateur. La seconde étape nécessite le calcul d'une homographie pour passer du point de vue de l'observateur à celui du vidéo-projecteur. Il suffit alors d'intégrer cette matrice dans le pipeline de construction de l'image d'OpenGL pour obtenir une image corrigée. Cette méthode est efficace cependant elle ne traite pas spécifiquement des images stéréoscopiques et nécessite le calcul et l'insertion d'une homographie et ce même dans le cas d'un vidéo-projecteur positionné face à l'écran.

Nous proposons donc une méthode simple pour corriger les mouvements pseudoscopiques à l’aide de la bibliothèque graphique OpenGL.

5.3 Correction des mouvements pseudoscopiques en synthèse d'images

Il existe une analogie entre la scène réelle et la scène virtuelle nous permettant d'établir quelques relations d'équivalence. C'est en conservant ces équivalences valides durant les déplacements de l'observateur que nous supprimerons les mouvements pseudoscopiques.

5.3.1 Création d'une paire d'images en position orthostéréoscopique

Comme nous l'avons décrit précédemment dans le chapitre 4.2, le système composé de deux caméras doit satisfaire les contraintes illustrées sur la figure 119. Il faut noter que cette méthode impose l'utilisation de pyramides de projection non symétriques afin d'assurer le parallélisme entre les axes optiques des caméras.

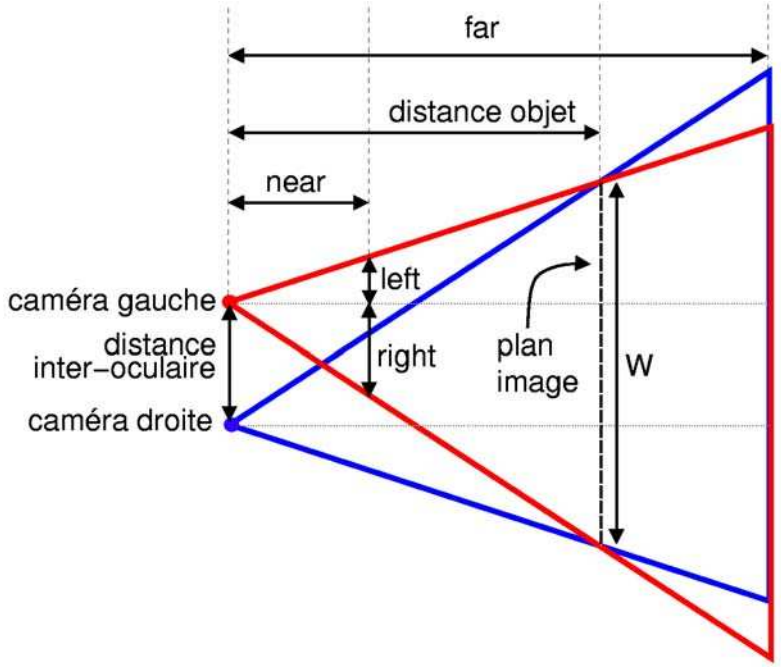


Figure 119 : paramètres d'un systèmes de deux caméras stéréoscopiques.

avec :

- distance interoculaire (aussi notée *dio*): distance séparant les deux caméras. Notons que l'on s'autorise l'emploi de cette dénomination normalement réservée aux yeux.
- W : la largeur du plan image.
- Les paramètres *left*, *right*, *top*, *bottom*, *near* et *far* sont les paramètres utilisées pour définir la pyramide de projection d'une caméra dans la bibliothèque graphique OpenGL.

On remarque qu'un point situé sur le plan image aura la même position en pixel sur les deux images donc une parallaxe nulle. Il sera donc perçu par l'observateur comme étant situé sur l'écran. Les objets situés derrière le plan image seront observés derrière l'écran et réciproquement, les objets situés devant le plan image seront observés devant l'écran.

5.3.2 Analogies entre la scène réelle et la scène virtuelle

Les deux images stéréoscopiques sont générées avec deux caméras virtuelles non-symétriques. Il ne nous reste plus qu'à placer ces caméras correctement et à les paramétrer en fonction de la position de l'observateur afin qu'à chaque instant celui-ci soit en position orthostéroscopique. Pour cela, nous allons établir un certain nombre de correspondances entre la scène virtuelle et la scène réelle. Le principe de cette démarche est d'associer une caméra virtuelle à chaque œil de l'observateur, ce qui suppose une série d'analogies :

largeur de l'écran	⇔	largeur du plan image
distance inter-oculaire$_{obs}$	⇔	distance inter-oculaire$_{cameras}$
distance entre l'observateur et l'écran	⇔	Distance objet
champ de vision de l'observateur à travers l'écran	⇔	champ de vision des caméras virtuelles à travers le plan image
le rapport largeur / hauteur de l'écran	⇔	le rapport largeur / hauteur du plan image

Pour passer du référentiel de l'écran à celui de la scène virtuelle, il faut respecter la relation suivante :

$$\frac{dio_{oeil}}{dio_{caméra}} = \frac{dist\,objet_{oeil}}{dist\,objet_{caméra}} = \frac{W_{écran}}{W_{focale}} = facteur\ d'échelle$$

Ces relations traduisent le fait que les deux modèles (yeux de l'observateur/écran et deux caméras/plan image) sont identiques à un facteur d'échelle près. Certains paramètres de cette équation dépendent de la scène réelle et sont donc fixés à l'avance comme dio_{oeil}, $dist\ objet_{oeil}$ et $W_{écran}$. Il suffit alors de fixer un des paramètres du système de caméras virtuelles pour connaître les deux autres.

Il faut régler les paramètres de la projection perspective de façon à ce que « la pyramide de vision » de chaque caméra passe exactement par les bords de l'écran quelle que soit la position de l'observateur. La figure 120 représente une vue de dessus d'un système de caméras virtuelles compatible avec les yeux d'un observateur.

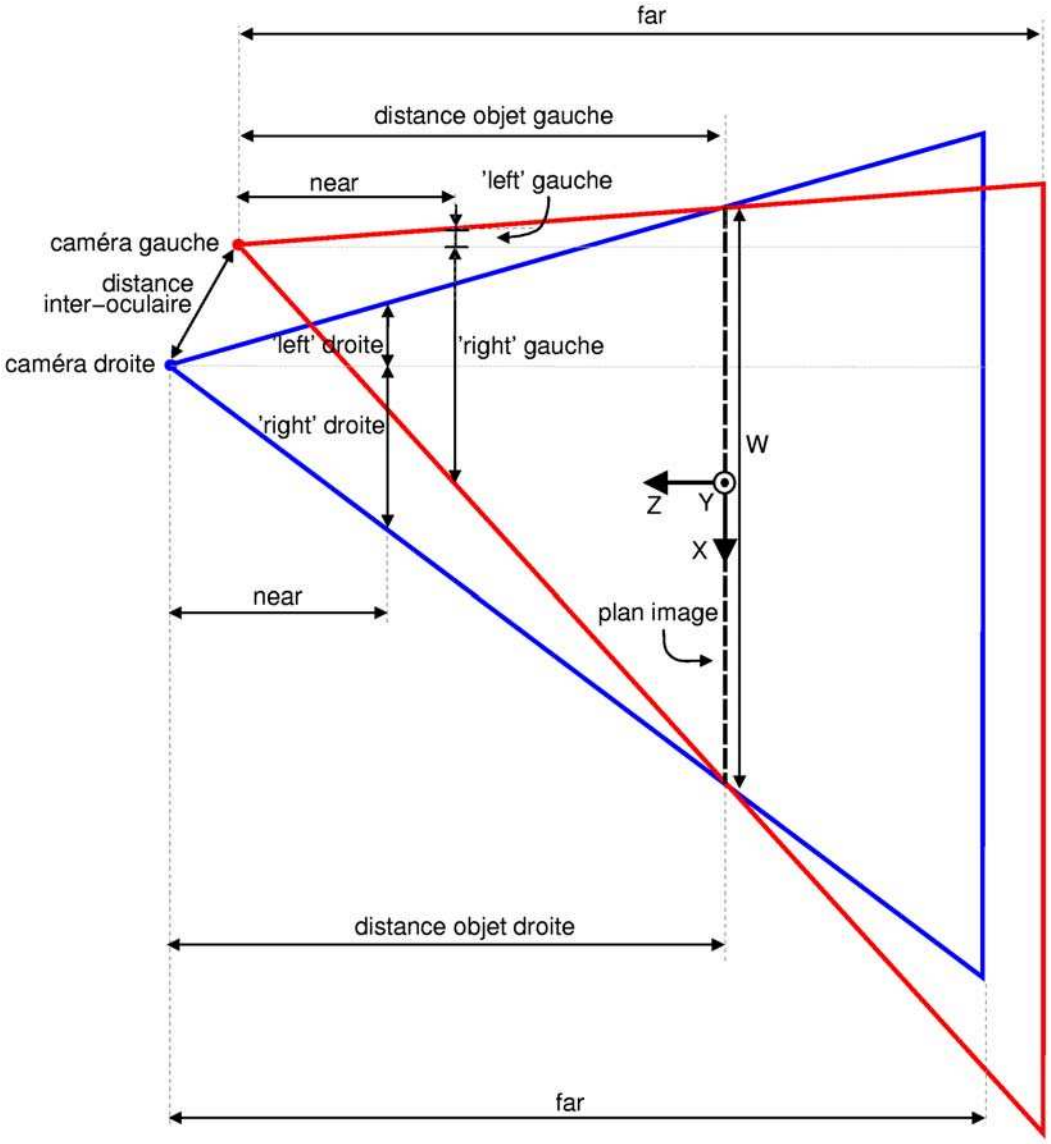


Figure 120 : paramètres d'un systèmes de deux caméras stéréoscopiques excentré par rapport à l'écran.

Voici la liste des relations que nous pouvons extraire de la figure 120:

- $dist\_obj_G = \frac{obs_G.z}{facteur\_echelle}$
- $dist\_obj_D = \frac{obs_D.z}{facteur\_echelle}$
- $left_D = \frac{near}{2.dist\_obj_D}\left(W + \frac{2obs_D.x}{facteur\_echelle}\right)$
- $right_D = \frac{near}{2.dist\_obj_D}\left(W - \frac{2obs_D.x}{facteur\_echelle}\right)$

- $left_G = \frac{near}{2.dist\_obj_G}\left(W + \frac{2obs_G.x}{facteur\_echelle}\right)$

- $right_G = \frac{near}{2.dist\_obj_G}\left(W - \frac{2obs_G.x}{facteur\_echelle}\right)$

avec :

- $letf_D$ le paramètre *left* de la fonction de projection perspective pour la caméra de droite.

- $right_D$: le paramètre *right* de la fonction de projection perspective pour la caméra de droite.

- $obs_D.x$, $obs_D.y$ et $obs_D.z$: position de l'œil doit de l'observateur dans le référentiel de l'écran.

- $dist\_obj_D$: distance objet pour la caméra de droite

(et réciproquement pour la caméra de gauche)

En ce qui concerne les paramètres *top* et *bottom* de la fonction de projection perspective (figure 121), on procède de la même façon que pour les paramètres *right* et *left*.

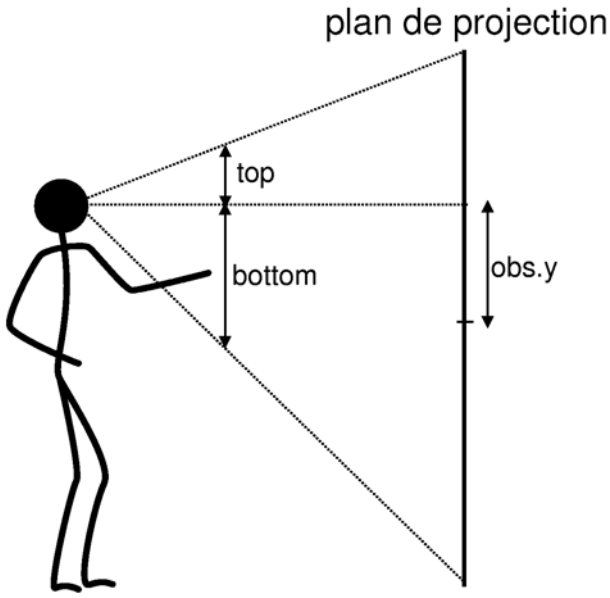


Figure 121 : système de deux caméras stéréoscopiques vu de coté.

On en établit les relations suivantes :

- $top_D = \frac{near}{2.dist\_obj_D}\left(\frac{W}{ratio} - \frac{2obs_D.y}{facteur\_echelle}\right)$
- $bottom_D = \frac{near}{2.dist\_obj_D}\left(\frac{W}{ratio} + \frac{2obs_D.y}{facteur\_echelle}\right)$

où *ratio* correspond au rapport largeur / hauteur de l'écran (et du plan image). On en déduit les mêmes formules pour la caméra de gauche.

5.3.3 Correction des mouvements pseudoscopiques en synthèse d'image

Nous venons de voir un certain nombre d'analogies entre la scène réelle et la scène virtuelle. De ces analogies nous avons extrait des relations que nous allons pouvoir utiliser pour corriger les images. Nous proposons deux modes de résolution : soit on associe le référentiel de la scène virtuelle au référentiel de l'écran (figure 122), soit on l'associe au référentiel de l'observateur (figure 123).

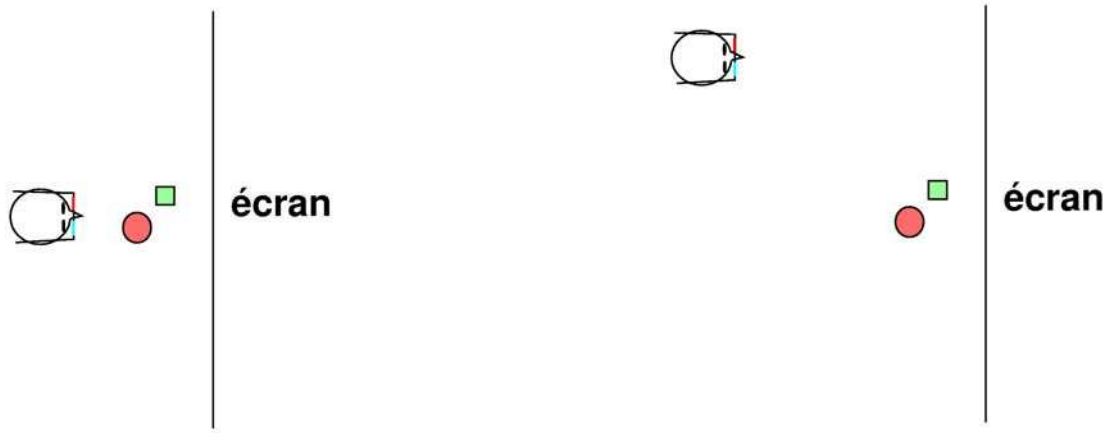


Figure 122 : scène liée à l'écran.

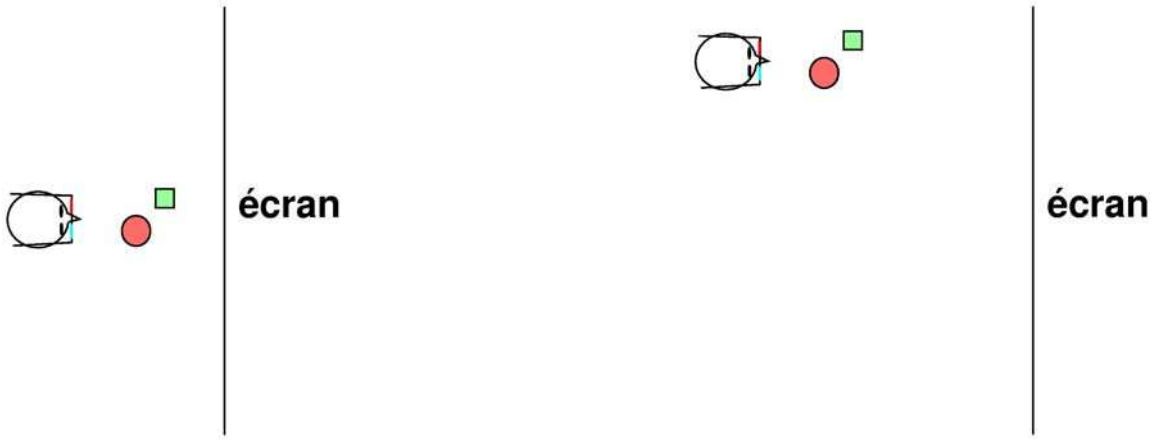


Figure 123 : scène liée à l'observateur.

5.3.3.1 Scène liée à l'écran

Avec ce modèle de restitution, l'observateur doit avoir la sensation que la scène virtuelle est fixe par rapport au référentiel de l'écran. Autrement dit, il peut voir cette scène selon différents points de vue. Cette méthode nécessite donc d'agir non seulement sur les paramètres de projection des caméras mais aussi sur leurs positions dans la scène virtuelle. En ce qui concerne les paramètres de projection, il suffit d'utiliser les équations du chapitre 5.3.2 qui nous assurent d'être à la position orthostéroscopique à chaque instant et évite ainsi tout mouvement pseudoscopique. Il reste ensuite à appliquer au système de caméras les mêmes déplacements que l'observateur, éventuellement à un facteur d'échelle près.

5.3.3.2 Scène liée à l'observateur

Dans le second mode, l'observateur voit la même scène quelque soit sa position, il n'en a donc qu'un seul point de vue et il n'est alors pas nécessaire d'appliquer aux caméras virtuelles une quelconque modification de position. Il est suffisant de mettre à jour les paramètres de projection des caméras.

5.3.4 Implémentation

Nous proposons une implémentation utilisant la bibliothèque graphique OpenGL ainsi que la bibliothèque Glu. Une façon pratique d'utiliser un système de deux caméras est d'utiliser la fonction `gluLookAt` (cf. *OpenGL Programing Guide* [SWN95] de Shreiner et al.) sur chacune des deux caméras. Cette fonction prend en paramètres les coordonnées de la caméra, les coordonnées du point visé (*focus*), et la direction du vecteur « vertical » dans le référentiel de la caméra (*up*). Il faut veiller à ce que les vecteurs *focus*, *up* et le vecteur reliant les deux caméras soient perpendiculaires deux à deux. Ceci nous assure que les axes optiques soient parallèles.

Quelle que soit le référentiel choisi, il faut commencer par initialiser certains paramètres comme la distance inter-oculaire (6,5 cm en moyenne) ou la taille de l'écran permettant de calculer le facteur d'échelle entre la scène virtuelle et la scène réelle (cf. chapitre 5.3.2). Il faut aussi déterminer certains paramètres de la matrice de projection.

```
double w_ecran = W_ECRAN;               // largeur de l'écran en cm
double w_virtuel = W_VIRTUEL;           // largeur du plan image souhaitée
double facteurEchelle = w_ecran / w_virtuel;
double ratio = w_ecran / hauteur_ecran;
double near = NEAR;
double far  = FAR;
```

5.3.4.1 Scène liée à l'écran

Le chapitre 5.3.3.1 indique qu'il est nécessaire d'appliquer certaines transformations (translations et rotations) au système de caméras en plus du paramétrage de la fonction *GLfrustum* gérant la projection sous OpenGL. Voici l'algorithme effectuant la correction des images dans le cas de la scène virtuelle liée à l'écran :

```
void drawFunc(void)
{
 ...
 // calculer le vecteur up
 ...

 //camera droite
 double distObj = obsDroite.z / facteurEchelle;
 double left = near*(w_virtuel+2.0*obsDroite.x/facteurEchelle)/(2.0* distObj);
 double right = near*(w_virtuel-2.0*obsDroite.x/facteurEchelle)/(2.0* distObj);
 double top = near*(w_virtuel/ratio-2.0*obsDroite.y/facteurEchelle)/(2.0* distObj);
 double bottom = near*(w_virtuel/ratio+2.0*obsDroite.y/facteurEchelle)/(2.0* distObj);

 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glFrustum(-left, right, -bottom, top, near, far);

 // calculer le vecteurs focus
 ...

 gluLookAt(cameraDroite.x + obsDroite.x / facteurEchelle,
           cameraDroite.y + obsDroite.y / facteurEchelle,
           cameraDroite.z + obsDroite.z / facteurEchelle,
           focus.x, focus.y, focus.z,
           up.x, up.y, up.z);

 drawScene(); // affiche la scène OpenGL

 //camera gauche
 ...
}
```

5.3.4.2 Scène liée à l'observateur

Dans le cas où la scène virtuelle est liée à l'observateur, il suffit de remplacer la fonction *gluLookAt* du code précédent par :

```
gluLookAt(cameraDroite.x, cameraDroite.y, cameraDroite.z,
          focus.x, focus.y, focus.z,
          up.x, up.y, up.z);
```

5.3.5 Résultats

La figure 124 représente un observateur face à un écran de réalité virtuelle. La scène virtuelle représente une pièce dans laquelle se trouve un pilier surmonté d'une sphère. Les mouvements pseudoscopiques sont particulièrement présents dans ce genre de scènes où l'observateur voit les murs se déformer et se déplacer quand lui-même se déplace. La figure 125 illustre la méthode consistant à associer le référentiel de la scène virtuelle au référentiel de l'observateur. La scène virtuelle suit les déplacements de l'observateur. Les figure 126 et figure 127 illustrent la méthode consistant à associer le référentiel de la scène virtuelle au référentiel de l'écran. Dans la figure 126, l'observateur s'est déplacé par rapport à l'écran et la paire d'images stéréoscopiques a été modifiée en conséquence. Pour un observateur resté face à l'écran, la sphère ne prend plus une forme sphérique mais une forme elliptique. Ce n'est pas le cas pour l'observateur qui a une vision orthostéroscopique de l'image (figure 127).

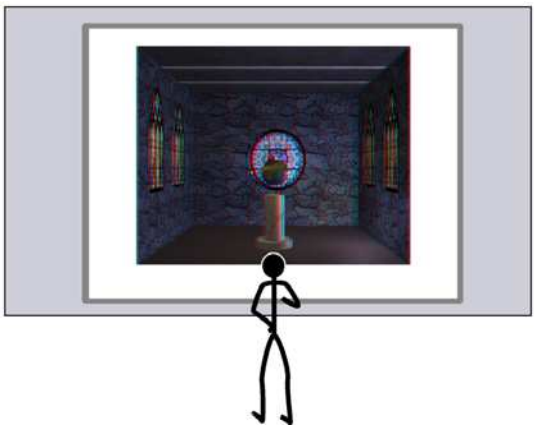


Figure 124 : observateur devant l'écran.

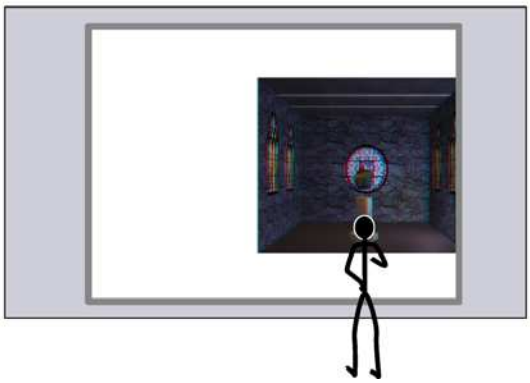


Figure 125 : scène liée à l'observateur.

Figure 126 : scène liée à l'écran.

Figure 127 : scène liée à l'écran vue par l'observateur.

Les deux méthodes présentées ci-dessus permettent donc de corriger les mouvements pseudoscopiques créés lors des déplacements de l'observateur face à son écran stéréoscopique. Leur implémentation en OpenGL est simple et efficace. Ces deux méthodes garantissent une restitution de la scène virtuelle sans distorsion des objets. La première méthode donne à l'utilisateur la sensation de se déplacer face à une scène fixe par rapport à l'écran. Cette méthode lui permet donc de voir cette scène sous plusieurs points de vue. La seconde méthode associe le référentiel de la scène virtuelle à celui de l'observateur qui, par conséquent, voit exactement la même chose, quelle que soit sa position. Certaines applications deviennent alors accessibles aux enfants dont la petite taille constitue un handicap gênant pour une bonne perception d'une image en relief. En outre, cette méthode procure plus de confort à l'utilisateur qui peut se déplacer sans ressentir de gène.

Les méthodes proposées sont indépendantes du système de suivi de la tête de l'utilisateur ainsi que du mode de restitution des images stéréoscopiques (écran d'ordinateur, grand écran rétro-éclairé…). Elle sont simples à implémenter et fonctionnent sur n'importe quel code source OpenGL.

5.4 Adaptation aux Plane Sweep

L'application de ces méthodes de correction des mouvements pseudoscopiques est directe puisque les deux caméras virtuelles utilisées peuvent se paramétrer aussi bien qu'avec des méthodes de synthèse d'images classiques. Les positions des caméras virtuelles sont parfaitement connues et il suffit de les paramétrer en utilisant les formules du chapitre 5.3.2 et en respectant la contrainte imposant d'avoir l'axe optique des cameras perpendiculaire au plan image (figure 128).

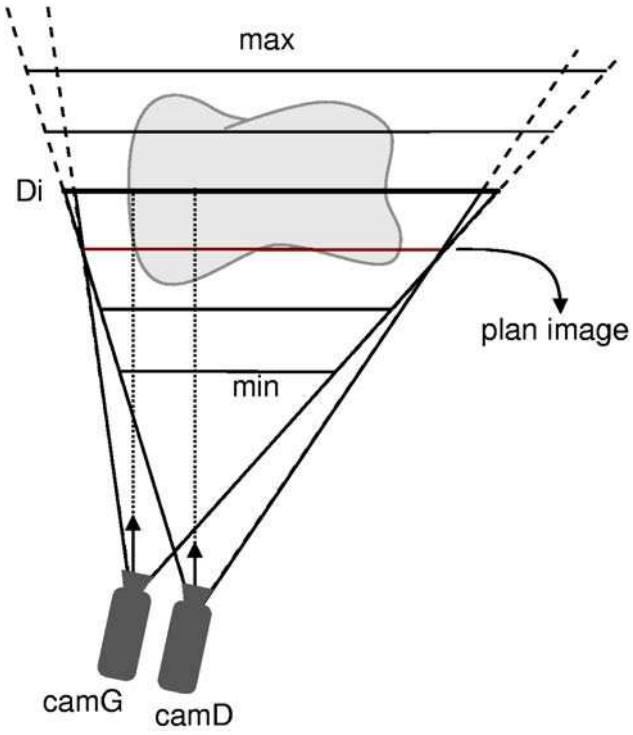


Figure 128 : correction des mouvements pseudoscopiques et méthode des Plane Sweep.

Notons que la méthode considérant que la scène virtuelle est liée au référentiel de l'écran est nettement plus intéressante dans le cas du rendu à base de vidéos car elle permet de visualiser la scène virtuelle sous différents points de vue.

Les quadrilatères (les plans) discrétisant la scène sont déterminés par projection des deux coins de droite de la caméra de gauche et des deux coins de gauche de la caméras de droite. Il n'y a donc pas de changement majeur par rapport à la méthode de rendu stéréoscopique présentée dans las section 4.3.

Conclusion

Bilan

Le rendu à base de vidéo est une discipline jeune et très active. Les résultats obtenus, toutes méthodes confondues, ne sont pas encore pleinement satisfaisants, notamment au niveau du rendu visuel et du temps d'exécution. La méthode des *Plane Sweep*, que j'ai adopté au fil de mes réflexions sur le rendu à base d'images temps réel, fait partie des méthodes prometteuses de rendu à base de vidéos.

Je me suis intéressés à la phase de traitement des données de la méthode des *Plane Sweep* afin d'en améliorer le rendu. J'ai pour cela proposé une nouvelle approche concernant le calcul des scores permettant de faire un choix plus judicieux concernant la couleur des pixels de l'image finale. Cette nouvelle approche permet de se passer de l'emploi d'une caméra de référence sensée représenter au mieux la caméra virtuelle. L'absence d'une telle caméra de référence permet d'éviter des sauts sur les images lors du déplacement de la caméra virtuelle d'une caméra vers une autre. De plus, la caméra virtuelle se trouve soumise à moins de contraintes et devient alors plus maniables.

Ensuite, dans la continuité de mes travaux et dans un souci d'augmenter la qualité visuelle des images générées, je me suis penchés sur la gestion des occlusions partielles. Il s'agit d'un problème qui concerne une très grande majorité des méthodes de reconstruction et de rendu à base d'images. Les méthodes que je propose permettent d'identifier les zones de l'image où figurent des discontinuités de profondeur ainsi que des discontinuités de couleur des objets. Une fois ces zones identifiées, elles sont traitées en sélectionnant l'information pertinente provenant de chaque caméra filmant la scène. Le traitement supplémentaire des données génère une baisse du framerate mais la qualité visuelle des images générées est améliorée.

Mon approche marque donc une évolution par raport aux autres techniques de *Plane Sweep*, tant au niveau de la gestion des occlusions qu'au niveau de la souplessse lors de la navigation. La méthode des *Visual Hulls*, principale concurente de ma méthode, utilise sensiblement le même nombre de caméras que les *Plane Sweep*. Les *Visual Hulls* permettent en outre de gérer les reflets spéculaires et sont moins contraignant que les *Plane Sweep* en ce qui concerne le positionement des caméras. Cependant ils ne peuvent gérer qu'un objet principal dans une scène ne peuvent donc traiter l'arrière plan ce qui est tout à fait réalisable avec les *Plane Sweep*. De plus, les *Plane Sweep* gèrent relativement bien les objets concaves ce qui n'est pas le cas des *Visual Hulls*.

Dans une second temps, je me suis intéressé à la réalité virtuelle et à ses applications, en particulier au niveau du rendu à base de vidéos. En effet, la réalité virtuelle est un domaine de plus en plus actif et dont les technologies émergentes sont très prometteuses pour les prochaines années. Parmi les aspects visuels les plus immersifs, la vision en relief m'a paru être tout à fait prépondérante. De plus, il est apparu que la méthode des *Plane Sweep* est tout à fait adaptée à ce genre de problèmes. Ainsi, je propose une amélioration des *Plane Sweep* appliquée à la création de paires d'images stéréoscopiques permettant de générer la seconde vue à moindre coût alors qu'une majorité des méthodes concurrentes sont contraintes d'effectuer deux rendus indépendants. Mon approche consiste à mettre en commun pour les deux images les données extraites de la scène.

L'affichage d'images stéréoscopiques, en particulier sur des grands écrans, est susceptible de générer des mouvements pseudoscopiques modifiant la perception de la scène virtuelle. En effet, ces mouvements apparaissent lorsque l'observateur se déplace devant son écran

stéréoscopique, il ressent alors une distorsion des proportions de la scène virtuelle et voit les objets se déplacer de façon anormale. Je propose une méthode simple permettant de supprimer ces mouvements pseudoscopiques d'une part sur les méthodes classiques de rendu d'images de synthèse et d'autre part sur mon adaptation stéréoscopique des *Plane Sweep*.

Toutes les méthodes que je présente utilisent largement les possibilités du processeur de la carte graphique à l'aide des *shader programs* et génèrent toutes des images en temps réel.

Perspectives

Quelle que soit leur approche, chaque méthode de rendu à base de vidéo a de larges lacunes à combler. Parmi les problèmes communs figurent le traitement du flux vidéo et le calibrage des caméras. Le problème du flux vidéo risque d'être difficile à régler si l'on considère que la résolution des dispositifs de visualisation de masse (télévision, ordinateur…) ainsi que l'exigence du public ne cessent d'augmenter. En revanche les techniques de calibrage continuent de s'améliorer et de s'automatiser mais nécessitent toujours l'emploi un dispositif particulier, en général une mire. Notons qu'il est tout à fait envisageable de tenter une adaptation des *Plane Sweep* aux *Projective Grid Sapce* de Saito et Kanade [SK99]. Dans ce cas, seules les relations épipolaires entre images définissent les relations entre les caméras. L'estimation des relations épipolaires entre plusieurs images ne nécessite pas nécessairement une mire. Ces relations peuvent se calculer rapidement, une suite logique de l'adaptation précédente serait donc de mettre au point une méthode de *Plane Sweep* utilisant des caméras non calibrées mobiles.

Les problèmes de qualité visuelle concernent eux aussi toutes les méthodes de rendu à base de vidéos. Les *Plane Sweep* sont particulièrement sujets aux problèmes de mauvais appariements. Il paraît donc tout à fait pertinent de continuer à orienter mes recherches sur un renforcement des critères d'appariements entre images.

Il existe ensuite des problèmes directement liés à la méthode des *Plane Sweep*. La taille des images générées fait partie des enjeux de cette méthode car elle a un impact direct sur les temps de calcul requis. Pour l'instant, seules les images de « petite taille », c'est à dire de résolution inférieure à celle d'un poste de télévision (720x576) sont à même d'être générées en temps réel. Il n'est pas raisonnable de compter uniquement sur une évolution des performances des cartes graphiques pour régler ce genre de problèmes. Il est préférable de chercher des améliorations d'optimisation des traitements afin de réduire le nombre de calculs. Une approche intéressante qui augmenterait aussi la qualité visuelle consiste à répartir les plans dans l'espace de façon à ce qu'ils représentent d'avantage la surface des objets de la scène. Dans le même état d'esprit, une seconde approche consiste à faire varier la résolution des plans suivant la partie de la scène sur laquelle ils se trouvent.

Il est probable que mes prochains travaux de recherche s'orientent en premier lieu vers une adaptation des *Plane Sweep* aux *Projective Grid Space* principalement développés dans le laboratoire du professeur Hideo Saito.

Bibliographie

[AB91] E. H. Adelson and J. R. Bergen, *The Plenoptic Function and the Elements of Early Vision*, Computational Models of Visual Processing, Edited by Michael Landy and J. Anthony Movshon, The MIT Press, 1991.

[AC01] Daniel G. Aliaga and Ingrid Carlbom, *Plenoptic Stitching: A Scalable Method for Reconstructing 3D Interactive Walkthroughs*, ACM SIGGRAPH 2001, Computer Graphics Proceedings, pages 443—450, 2001.

[AH02] Tomas Akenine-Möller and Eric Haines, *Real-Time Rendering*, A.K. Peters Ltd., 2nd edition, ISBN 1568811829, 2002.

[AL99] Daniel G. Alilaga and Anselmo Lastra, *Automatic Image Placement to provide a guaranteed frame rate*, in proc. SIGGRAPH'99, pages 307-316, 1999.

[AZP05] Aseem Agarwala, Colin Zheng, Chris Pal, Maneesh Agrawala, Michael Cohen, Brian Curless, David Salesin and Richard Szeliski. *Panoramic Video Textures*, in proc. SIGGRAPH 2005, pages 821-827 ,2005.

[BA93] M.J. Black, P. Anandan, *A framework for the robust estimation of optical flow*, in proc. ICCV 1993, pages 231–236, 1993.

[BBH03] Myron Z. Brown, Darius Burschka, and Gregory D. Hager, *Advances in Computational Stereo*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25(8), pages 993-1008, 2003.

[BBM01] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen, *Unstructured Lumigraph Rendering*, in proc. of ACM SIGGRAPH 2001, Computer Graphics Proceedings, Annual Conference Series, pages 425-432, August 2001.

[BC93] Grigore Burdea and Philippe Coiffet, *La Réalité Virtuelle*, Hermes, ISBN : 2866013867, 1993.

[BD00] E. Borovikov L. Davis, *A Distributed System for Real-Time Volume Reconstruction*, in proc. of the Fifth IEEE International Workshop on Computer Architectures for Machine Perception (CAMP'00), page: 183, 2000.

[BF92] J.L. Barron, D.J. Fleet, S.S. Beauchemin and T.A. Burkitt, *Performance Of Optical Flow Techniques*, CVPR, pages 236-242, 1992.

[BG01] Samuel Boivin and André Gagalowicz, *Image-Based Rendering of Diffuse, Specular and Glossy Surfaces from a Single Image*, in proc. SIGGRAPH 2001, pages 107-116, 2001.

[BJ03] Hermann Birkholz and Dietmar Jackél, *Image warping with feature curves*, Spring Conference on Computer Graphics, proc. of the 19th spring conference on Computer graphics, Budmerice, Slovakia, pages 199-202, 2003.

[BN92] Thaddeus Beier and Shawn Neely, *Feature-based image metamorphosis*, , Proc. SIGGRAPH'92, vol. 26, pages 35-42, 1992.

[BN98] Dinkar N. Bhat and Shree K. Nayar, *Ordinal Measures for Image Correspondence*, IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 415-423, 1998.

[BS72] D. I. Barnea and H. F. Silverman, *A Class Of Algorithms For Fast Digital Image Registration*, IEEE Transactions On Computers, pages 179-186, 1972.

[BT98] Stan Birchfield and Carlo Tomasi, *A Pixel Dissimilarity Measure That Is Insensitive to Image Sampling*, in proc. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, pages 401-406, 1998.

[C95] Shenchang Eric Chen, *QuickTime®VR - An Image Based Approach to Virtual Environment Navigation*, in computer graphics, (SIGGRAPH'95), pages 29-38, august 1995.

[C96] Robert T. Collins, *A Space-Sweep Approach to True Multi-Image Matching*, in proc. Computer Vision and Pattern Recognition Conf., pages 358-363, 1996.

[CBC02] Wei-Chao Chen, Jean-Yves Bouguet, Michael H. Chu, and Radek Grzeszczuk, *Light field mapping: efficient representation and hardware rendering of surface light fields*, in proc. of the 29th annual conference on Computer graphics and interactive techniques, pages 447-456, 2002.

[CC03] S. Chambon, A. Crouzil, *Mesures de corrélation robustes aux occultations,* ORASIS 2003, pages 239-248, Gérardmer, 2003

[CM97] D. Comaniciu and P. Meer, *Robust analysis of feature spaces: color image segmentation*, in proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97), pages 750-755, 1997.

[CMC96] Alain Crouzil, Louis Massip-Pailhes and Serge Castan, *A New Correlation Criterion Based on Gradient Fields Similarity*, 13th International Conference on Pattern Recognition , Vienne, Autriche, pages 632-636, 1996.

[CPS97] M. Czernuszenko, D. Pape, D. Sandin, T. DeFanti, L. Dawe and M. Brown, , in ACM SIGGRAPH Computer Graphics, vol. 31, pages 46-49, 1997.

[CR99] Roberto Cipolla and Duncan Robertson, *Photobuilder - 3D Models of Architectural Scenes from Uncalibrated Images*, in IEEE int. Conf. On Multimedia Computing and Systems, pages 25-31, 1999.

[CSD93] Carolina Cruz-Neira, Daniel J. Sandin and Thomas A. DeFanti, *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*, in proc. of ACM SIGGRAPH'93, pages 135-142, 1993.

[CW93] Shenchang Eric Chen and Lance Williams, *View interpolation for image synthesis*, in proc. of the Annual Conference on Computer Graphics, pages 279-288, Anaheim, CA, USA, August 1993. ACM Press.

[D34] B. Delaunay, *Sur la sphère vide*, Bul. Acad. Sci. URSS, Class. Sci. Nat., pages 793-800, 1934.

[DBY98] Paul E. Debevec, George Borshukov, and Yizhou Yu, *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping*, In 9th Eurographics Rendering Workshop, Vienna, Austria, 1998.

[DF01] F. Devernay and O. Faugeras, *Straight Lines Have to Be Straight: Automatic Calibration and Removal of Distortion from Scenes of Structured Environments*, Machine Vision and Applications, pages 14-24, 2001.

[DG93] R. Deriche and G. Giraudon, *A computational approach for corner and vertex detection*, International Journal of Computer Vision, 1(2):167-187, 1993

[DML96] N. A. Dodgson, J. R. Moore, S. R. Lang and A. R. L. Travis, *Time-multiplexed color autostereoscopic display*, in proc. of SPIE Symposium on Stereoscopic Displays and Applications VII, San Jose, California, Jan 28-Feb 2, pages 10-19, 1996.

[DTM96] Paul E. Debevec, Camillo J. Taylor and Jitendra Malik, *Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach*, in proc. SIGGRAPH'96, vol. 30, pages 11-20, 1996.

[DTR05] Marc-Antoine Drouin, Martin Trudeau and Sébastien Roy, *Geo-consistency for Wide Multi-Camera Stereo*, CVPR, San Diego, USA, June 2005.

[DV99] J. S. De Bonet and P. Viola, *Poxels: Probabilistic Voxelized Volume Reconstruction*, in proc. of ICCV, pages 418-425, 1999.

[DYB98] Paul Debevec, Yizhou Yu and George Boshokov, *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping*, 9th Eurographics Rendering Workshop, Austria, pages 14-27, 1998.

[ESG99] Peter Eisert, Eckehard Steinbach and Bernd Girod, *3-D Shape Reconstruction from Light Fields Using Voxel Back-Projection*, Vision, Modeling, and Visualization Workshop 99, pages 67-74, Erlangen, Germany, Nov. 1999.

[F01] Tom Forsyth, *Impostors: adding clutter*, in Game Programming Gems 2, Mark DeLoura ed., Charles River Media, pages 488-496,2001.

[FB03] Jean-Sebastien Franco, Edmond Boyer, *Exact Polyhedral Visual Hulls*, British Machine Vision Conference (BMVC'03), vol. 1, pages 329-338, September 2003.

[FB81] M. A. Fischler and R. C. Bolles, *Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography*, Comm. Assoc. Comp. Mach., vol. 24, pages 381-395, 1981.

[FH04] Pedro F. Felzenszwalb and Daniel P. Huttenlocher, *Efficient Graph-Based Image Segmentation*, International Journal of Computer Vision, volume 59, pages 167-181, 2004.

[FL04] Olivier Faugeras and Quang-Tuan Luong, *The Geometry of Multiple Images : The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications*, The MIT Press, ISBN 0262562049, 2004.

[FP02] David Forsyth and Jean Ponce, *Computer Vision : a Modern Approach*, Publisher: Prentice Hall, 1st edition, ISBN: 0130851981, 2002.

[FWZ03] Andrew Fitzgibbon, Yonatan Wexler and Andrew Zisserman, *Image-based rendering using image-based priors*, 9th IEEE International Conference on Computer Vision (ICCV 2003), pages 1176-1183, 2003

[GBC95] J. Gomes, J. A. Berton, B. Costa, L. Darsa, L. Velho and G. Wolberg, *Warping and morphing of graphical objects*, Course Notes, SIGGRAPH'95, 1995.

[GG84] S. Geman and D. Geman, *Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 6, pages 721-741, 1984.

[GG91] D. Geiger and F. Girosi, *Parallel and Deterministic Algorithms from Mrfs: Surface Reconstruction*, IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. 13, pages 401-412, 1991.

[GGS96] J. Gortlerand R. Grzeszczuk, R. Szeliski and M. F. Cohen, *The lumigraph*, SIGGRAPH, pages 43-54, 1996.

[GKV04] Indra Geys, T. P. Koninckx and L. Van Gool, *Fast Interpolated Cameras by combining a GPU based Plane Sweep with a Max-Flow Regularisation Algorithm*, in proc. of second international symposium on 3D Data Processing, Visualization & Transmission - 3DPVT'04, pages 534-541, 2004.

[GM03] B. Goldlücke, M. Magnor, *Real-Time Microfacet Billboarding for Free-Viewpoint Video Rendering*, in p*roc. IEEE International Conference on Image Processing (ICIP'03),* Barcelona, Spain, vol. 3, pages 713-716, September 2003.

[GMW02] Bastian Goldlücke, Marcus A. Magnor, Bennett Wilburn, *Hardware-accelerated Dynamic Light Field Rendering*, Proceedings Vision, in proc. of Modeling and Visualization VMV 2002, aka Berlin, Germany, pages 455-462, November 2002.

[GSB94] H. Gagnon, M. Soucy, R. Bergevin and D. Laurendeau, *Registration of Multiple Range Views for Automatic 3-D Model Building*, in IEEE conference of Computer Vision and Pattern Recognition, pages 581-586, 1994.

[GV05] Indra Geys and L.Van Gool, *Extended view interpolation by parallel use of the GPU and the CPU*, in proc. of IS&T SPIE, 17th annual symposium on electronic imaging - videometrics VIII, vol. 5665, pages 96-107, 2005.

[GV96] G. H. Golub and C. F. Van Loan, *Matrix computation*, 3rd edition, Johns Hopkins University Press, ISBN: 0801854148, 1996.

[H96] Hugues Hoppe, *Progressive meshes*, in proc. of the 23rd annual conference on Computer graphics and interactive techniques, pages 99-108. ACM Press, 1996.

[HAA97] Youichi Horry, Ken-Ichi Anjyo and Kiyoshi Arai, *Tour into the picture: using a spidery mesh interface to make animation from a single image*, in proc. SIGGRAPH'97, pages 225-232, 1997.

[HEH05] D. Hoiem, A.A. Efros, and M. Hebert, *Automatic Photo Pop-up*, in proc. SIGGRAPH 2005, pages 577-584, 2005.

[HKK01] Hiroshi, Kawasaki and Katsushi, *Light Field Rendering for Large-Scale Scenes*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01), vol. 2, pages 64-71, 2001.

[HS81] B. K. P. Horn and B. G. Schunk, *Determining optical flow*, Artificial Intelligence, vol. 17, pages 185-203, 1981.

[HS88] C. Harris and M. Stephens, *A combined corner and edge detector*, fourth alvey vision conference, pages 147-151, 1988.

[HZ04] Richard Hartley and Andrew Zisserman, *Multiple View Geometry in Computer Vision*, second edition, Cambridge University Press, ISBN: 052154051, 2004.

[I00] Intel Corporation, *Open Source Computer Vision Library, Reference Manual*, order number 123456-001, December, 2000.

[JW05] Neel Joshi, Bennett Wilburn, Vaibhav Vaish, Marc Levoy and Mark Horowitz, *Automatic Color Calibration for Large Camera Arrays*, UCSD CSE technical report CS2005-0821, May 11, 2005

[JWB04] Zhongding Jiang, Tien-Tsin Wong and Hujun Bao, *Three-Dimensional Image Warping on Programmable Graphics Hardware*, Technical report TR-03-01, Zhejiang University, China, 2004.

[K94] Takeo Kanade, *Development of a video-rate stereo machine*, in proc. of 94 ARPA Image Undersatanding Workshop, pages 549-557, 1994.

[KFL01] Don Kimber, Jonathan Foote and Surapong Lertsithichai, *FlyAbout: spatially indexed panoramic video*, ACM Multimedia, pages 339-347, 2001.

[KNR95] Takeo Kanade, P. J. Narayanan and Peter Rander, *Virtualized reality: concepts and early results*, Proceedings of the IEEE Workshop on Representation of Visual Scenes, page: 69, 1995.

[KO94] T. Kanade and M. Okutomi, *A stereo matching algorithm with an adaptive window: Theory and experiment*, IEEE Trans. Pattern Anal. and Machine Intel., pages 920-932, 1994.

[KPA01] Hyung Woo Kang, Soon Hyung Pyo, Ken-ichi Anjyo and Sung Yong Shin, *Tour Into the Picture using a Vanishing Line and its Extension to Panoramic Images*, Computer Graphics Forum, vol. 20, pages 132-141, 2001.

[KPH99] Reinhard Koch, Marc Pollefeys, Benno Heigl, Luc J. Van Gool and Heinrich Niemann, *Calibration of Hand-Held Camera Sequences for Plenoptic Modeling*, ICCV, pages 585-591, 1999.

[KRN97] Takeo Kanade, Peter Rander and P. J. Narayanan, *Virtualized Reality: Constructing Virtual Worlds from Real Scenes*, IEEE MultiMedia, volume 4, pages 34-47, 1997.

[KS02] H. Kang and S. Y. Shin, *Tour Into the Video: Image-based Navigation Scheme for Video Sequences of Dynamic Scenes*, in proc. of ACM Symposium on Virtual Reality Software and Technology, pages 73-80, 2002.

[KS04] H. Kang and S. Y. Shin. *Creating Walk-through Images from a Video Sequence of a Dynamic Scene*, Presence, MIT Press, vol. 13, pages 638-655, 2004.

[KSV98] T. Kanade, H. Saito, S. Vedula, *The 3D Room: Digitizing Time-Varying 3D Events by Synchronized Multiple Video Streams*, Robotics Institute Technical Report, CMU-RI-TR-98-34, December 1998.

[L01] David P. Luebke, *A developer's survey of polygonal simplification algorithms*, IEEE Computer Graphics and Applications, pages 24-35, 2001.

[L95] Sonia Leach, *Singular Value Decomposition – A primer*, Technical report, Department of Computer Science, Brown University, Providence, RI 02912, 1995.

[LA04] Manolis Lourakis and Antonis Argyros, *The Design and Implementation of a Generic Sparse Bundle Adjustment Software Package Based on the Levenberg-Marquardt Algorithm*, Technical Report No. 340, Institute of Computer Science - FORTH, Heraklion, Crete, Greece, Aug. 2004.

[LAM00] Lee, Aaron, Henri Moreton and Hugues Hoppe, *Displaced Subdivision Surfaces*, in proc. of SIGGRAPH'00, pages 85-94, July 2000.

[LBP01] S. Lazebnik, Edmond Boyer and Jean Ponce, *On How to Compute Exact Visual Hulls of Object Bounded by Smooth Surfaces*, in proc. of the IEEE Conference on Computer Vision and Pattern Recognition, Kauai, Hawaii, USA, page 156-161, 2001.

[LCY00] Kai Li, Han Chen, Yuqun Chen, Douglas W. Clark, Perry Cook, Stefanos Damianakis, Georg Essl, Adam Finkelstein, Thomas Funkhouser, Timothy Housel, Allison Klein, Zhiyan Liu, Emil Praun, Rudrajit Samanta, Ben Shedd, Jaswinder Pal Singh, George Tzanetakis and Jiannan Zhen,. *Building and Using A Scalable Display Wall System*, Computer Graphics and Applications, vol. 4: pages 29-37, 2000.

[LCZ99] D. Liebowitz, A. Criminisi, and A. Zisserman, *Creating Architectural Models from Images*, in proc. EuroGraphics, pages 39-50, 1999.

[LH96] Marc Levoy and Pat Hanrahan, *Light Field Rendering*, SIGGRAPH, pages 31-42, 1996.

[LKH98] W. Li, Q. Ke, X. Huang and N. Zheng, *Light field rendering of dynamic scene*, Machine Graphics and Vision, vol. 7, n°3, pages 551-563, 1998.

[LM97] Zhong-Dan Lan and Roger Mohr, Robust Location based Partial Correlation, Rapport de recherché RR-3186, INRIA, 1997.

[LMS03] M. Li, M. Magnor, H.-P. Seidel, *Online Accelerated Rendering of Visual Hulls in Real Scenes*, in proc. of International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'03), Plzen, Czech Republic, pages 290-297, February 2003.

[LMS03b] M. Li, M. Magnor, H.-P. Seidel, *Hardware-Accelerated Visual Hull Reconstruction and Rendering*, in proc. of Graphics Interface (GI'03), pages 65-71, Halifax, Canada, June 2003.

[LMS04] M. Li, M. Magnor, H.-P. Seidel, *Hardware-Accelerated Rendering of Photo Hulls*, Computer Graphics Forum (special issue Eurographics'04), vol. 23, pages 635-642, September 2004.

[LPT00] A. J. Lacey, N. Pinitkarn and N. A. Thacker, *An Evaluation of the Performance of RANSAC Algorithms for Stereo Camera Calibration*, in electronic proc. of The Eleventh British Machine Vision Conference (BMVC), 2000.

[LS97] J Lengyel and J Snyder, *Rendering with coherent layers*, in proc. SIGGRAPH'97, pages 233-242, 1997.

[LSM02] Ming Li, Hartmut Schirmacher, Marcus Magnor and Hans-Peter Seidel, *Combining Stereo and Visual Hull Information for On-line Reconstruction and Rendering of Dynamic Scenes*, in proc. of IEEE 2002 Workshop on Multimedia and Signal Processing (MMSP'02), pages 9-12, 2002.

[LT97] Kok-Lim Low and Tiow-Seng Tan, *Model simplification using vertex-clustering*, in proc. of the 1997 symposium on Interactive 3D graphics, pages 75-81, 1997.

[M95] E. P. Mücke, *A Robust Implementation for Three-dimensional Delaunay Triangulations*, in proc. of the 1st International Computational Geometry Software Workshop, pages 70-73, 1995.

[M05] Marcus A. Magnor, *Video-Based Rendering*, Éditor : A K Peters Ltd, ISBN : 1568812442, 2005.

[MB95] Leonard McMillan and Gary Bishop, *Plenoptic Modeling: An Image-Based Rendering System*, SIGGRAPH, pages 39-46,1995.

[MBM01] Wojciech Matusik, Chris Buehler and Leonard McMillan, Polyhedral Visual Hulls for Real-Time Rendering, in proc. of Twelfth Eurographics Workshop on Rendering, pages 115-126, London, England, June 2001,

[MBR00] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler and Leonard McMillan, *Image-Based Visual Hulls*, in proc ACM SIGGRAPH, pages 369-374, 2000.

[MP04] W. Matusik and H. Pfister, *3D TV: A Scalable System for Real-Time Acquisition, Transmission, and Autostereoscopic Display of Dynamic Scenes*, In proc. of SIGGRAPH 2004, vol. 23, pages 814-824, 2004.

[MRP98] G. S. P. Miller, S. Rubin and D. Ponceleon, *Lazy Decompression of surface Light Fields for Precomputed Global Illumination*, Eurographics Rendering Workshop 1998, pages 281-292, 1998.

[MS95] Paulo W. C. Maciel and Peter Shirley, *Visual navigation of large environments using textured clusters*, in ACM Siggraph Symposium on Interactive 3D Graphics, pages 95-102, 1995.

[NB03] Vincent Nozick, Venceslas Biri, *Méthode temps réel simple pour la correction des mouvements pseudoscopiques en réalité virtuelle*, AFIG 2003, pages 231-240, 2003.

[NC01] Harsh Nanda, Ross Cutler, *Practical Calibrations for a real-time digital omnidirectional camera*, Technical Sketches, Computer Vision and Pattern Recognition, Hawaii, US, Dec 2001.

[NMA05] Vincent Nozick, Sylvain Michelin and Didier Arquès, *Image-based rendering using plane-sweeping modelisation*, in proc. of IAPR MVA 2005, May 16-18, page 468-471, Tsukuba Science City, Japan, 2005.

[NMA06] Vincent Nozick, Sylvain Michelin and Didier Arquès, *Real-time Plane-sweep with local strategy*, in the Journal of WSCG, The 14-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, ISSN 1213-6972, pages 121-128, Plzen, Czech Republic, January 30 - February 3, 2006.

[NSI99] K. Nishino, Y. Sato and K. Ikeuchi, *Appearance compression and synthesis based on 3D Model*, in proc. of IEEE ICCV'99, pages 38-45, 1999.

[OCD01] Byong Mok Oh, Max Chen, Julie Dorsey, and FrédoDurand, *Image-Based Modeling and Photo Editing*, Proceedings of ACM SIGGRAPH 2001, Computer Graphics Proceedings, Annual Conference Series, August 2001, pages 433-442.

[OK93] M. Okutomi, T. Kanade, *A Multiple-Baseline Stereo*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15, pages 353-363, April 1993.

[P00] Marc Pollefeys, *Tutorial on 3d modelling from images*, In conjunction with ECCV 2000, Dublin, Ireland, june 2000.

[P87] Michael Potmesil, *Generating octree models of 3D objects from their silhouettes in a sequence of images Source,* Computer Vision, Graphics, and Image Processing, vol. 40, pages: 1-29, 1987.

[PAB99] Dave Pape, Josephine Anstey, Mike Bogucki, Greg Dawe, Tom DeFanti, Andy Johnson and Dan Sandin, *The ImmersaDesk3 - Experiences With A Flat Panel Display for Virtual Reality*, in proc. of . third Int'l Immersive Projection Technology Workshop, ACM Press, pages 107-112, 1999.

[PGM03] Christoph Petz, Bastian Goldlücke and Marcus A. Magnor, *Hardware-accelerated Autostereogram Rendering for Interactive 3D Visualization,* Stereoscopic Displays and Virtual Reality Systems X, Vol. 5006, SPIE Proceedings, pages 359-366, 2003.

[PGV04] Marc Pollefeys, Luc Van Gool, Maarten Vergauwen, Frank Verbiest, Kurt Cornelis, Jan Tops and Reinhard Koch, *Visual Modeling with a Hand-Held Camera*, International Journal of Computer Vision, pages 207-232, 2004.

[PKV00] M. Pollefeys, R. Koch, M. Vergauwen and L. Van Gool, *Automated reconstruction of 3D scenes from sequences of images*, ISPRS Journal Of Photogrammetry And Remote Sensing (55)4, pages 251-267, 2000.

[PSC05] Stephane Pelletier, Stephen P. Spackman and Jeremy R. Cooperstock, *High-Resolution Video Synthesis from Mixed-Resolution Video Based on the Estimate-and-Correct Method*, Seventh IEEE Workshops on Application of Computer Vision (WACV/MOTION'05), ,vol. 1, pages 172-177, 2005.

[PTV02] William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing*, Cambridge University Press, 2nd edition, ISBN: 0521750334, 2002.

[PVV00] Marc Pollefeys, Maarten Vergauwen and Luc Van Gool, *Automatic 3D modeling from image sequences*, proc. ISPRS congress, Amsterdam, pages 7-42, 2000.

[PZB00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross, *Surfels: surface elements as rendering primitives*, in proc. of the 27th annual conference on Computer graphics and interactive techniques, pages 335-342, 2000.

[R00] Ramesh Raskar, *Immersive Planar Display using Roughly Aligned Projectors*, in proc. of IEEE VR 2000, pages 109-116, March 2000.

[R02] Sidney F. Ray, *Applied Photographic Optics*, publisher: Focal Press, 3 edition, ISBN: 0240515404, March 2002.

[RB93J] Rossignac and P. Borrel, *Multi-resolution 3D approximations for rendering*, Modeling in Computer Graphics. Springer-Verlag, pages 455-465, June-July 1993.

[RBY99] R Raskar, M Brown, R Yang, W Chen, G Welch, H Towles, B Seales and H Fuchs, *Multi-Projector Displays using Camera-Based Registration*, IEEE Visualization 99, pages 161-168, 1999.

[RM95] Detlef Ruprecht and Heinrich Müller, *Image Warping with scattered data interpolation methods*, IEEE Computer Graphics and Applications, vol 15, pages 37-43, 1995.

[RNK97] P. Rander, P.J. Narayanan, and T. Kanade, *Virtualized reality: constructing time-varying virtual worlds from real world events*, Proc. of IEEE Visualization 1997, pages 277-283, October 1997.

[RP05] J. Repko, M. Pollefeys, *3D Models from Extended Uncalibrated Video Sequences: Addressing Key-frame Selection and Projective Drift*, Proc. 3DIM'05, pages 150-157, 2005

[S02] Jonathan Shade, *Approximating the Plenoptic Function*, UW CSE Technical Report, December 2002.

[S90] D. B. Smithe, *A two-pass mesh warping algorithm for object transformation and image interpolation*, Technical memo #1030, Industrial Light and Magic, 1990.

[S97] Gernot Schaufler, *Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes*, in proc. 8th Eurographics Workshop on Rendering, pages 151-162, 1997.

[SAK02] H. Sawhney, A. Arpa, R. Kumar, S. Samarasekera, M. Aggarwal, S. Hsu, D. Nistér and K. Hanna, *Video-flashlights – real time rendering of multiple videos for immersive model visualisation*, in proc. of Thirteenth Eurographics Workshop on Rendering (EGRW'02), pages 163-174, 2002.

[SD96] Steven M. Seitz and Charles R. Dyer, *View Morphing*, Computer Graphics, in proc. SIGGRAPH'96, vol. 30, pages 21-30, 1996.

[SD97] S. M. Seitz and C. R. Dyer, *Photorealistic Scene Reconstruction by Voxel Coloring*, in proc. Computer Vision and Pattern Recognition Conference, pages 1067-1073, 1997.

[SGH98] Jonathan Shade, Steven J. Gortler, Li-wei He and Richard Szeliski, *Layered Depth Images*, in proc. SIGGRAPH'98, pages 231-242, 1998.

[SH99] Heung-Yeung Shum and Li-Wei He, *Rendering with concentric mosaics*, in proc. of the 26th annual conference on Computer graphics and interactive techniques, pages 299-306, 1999.

[SK98] Steven M. Seitz and Kiriakos N. Kutulakos, *Plenoptic Image Editing*, ICCV'98, pages 17-24, 1998.

[SK99] Hideo Saito, Takeo Kanade, *Shape Reconstruction in Projective Grid Space from Large Number of Images*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99), Fort Collins, June 1999.

[SM04] A. Smolic; D. McCutchen, *3DAV exploration of video-based rendering technology in MPEG*, Circuits and Systems for Video Technology, IEEE Transactions, vol. 14, pages 348–356, 2004

[SMS01] Hartmut Schirmacher, Li Ming and Hans-Peter Seidel, *On-the-fly Processing of Generalized Lumigraphs*, in proc. of EUROGRAPHICS 2001, Eurographics Association, pages 165-173, 2001.

[SS02] Daniel Scharstein and Richard Szeliski, *A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms*, IJCV, 47, pages 7-42, 2002.

[SS96] Gernot Schaufler and Wolfgang Stürzlinger, *A Three Dimensional Image Cache for Virtual Reality*, in proc. of Eurographics'96, vol. 15, pages 227-236, August 1996.

[SS98] Daniel Scharstein and Richard Szeliski, *Stereo Matching with Nonlinear Diffusion*, International Journal of Computer Vision, pages 155-174, 1998.

[SWN95] Dave Shreiner, Mason Woo, Jackie Neider, *OpenGL Programming Guide: The Official Guide To Learning OpenGL*, Version 2, Éditeur : Addison-Wesley Professional, ISBN : 0321335732, 1995.

[T87] R. Tsai, *A Versatil Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses*, IEEE J. Robotics and Automation, Vol. RA-3, No.4, pages 323-344, Aug. 1987.

[TE99] A. Tal, G. Elber, *Image morphing with feature preseving texture*, Computer Graphics Forum, proc. Eurographics'99, vol. 18, pages 339-348, September 1999.

[TG99] X. Tong and R. Gray, *Compression of light fields using disparity compensation and vector quantisation*, proc. IASTED Conf. Computer Graphics and Imaging, pages 300-305, October 1999.

[TH86] Q. Tian and M. N. Huhns, *Algorithms for subpixel registration*, in proc. Computer Vision, Graphics, and Image Processing, vol. 35, pages 220-233, 1986.

[TLM03] C. Theobalt, M. Li, M. Magnor, H.-P. Seidel, *A Flexible and Versatile Studio for Synchronized Multi-view Video Recording*, in proc. IMA Vision, Video, and Graphics (VVG'03), Bath, UK, pages. 9-16, July 2003.

[TMH00] Bill Triggs, P. McLauchlan, Richard Hartley and A. Fitzgibbon, *Bundle Adjustment -- A Modern Synthesis*, Vision Algorithms: Theory and Practice, Lecture Notes in Computer Science, volume 1883, Springer-Verlag, pages 298-372, 2000.

[TSK01] Hai Tao, Harpreet S. Sawhney, and Rakesh Kumar, *A global matching framework for stereo computation*, in proc. IEEE International Conference on Computer Vision, ICCV'01, pages 532-539 , 2001.

[TZ00] P.H.S. Torr and A. Zisserman, *MLESAC: A New Robust Estimator with Application to Estimating Image Geometry*, Computer Vision and Image Understanding, vol. 78, pages 138-156, 2000.

[UKI01] Farhan Ullah, Shun'ichi Kaneko, Satoru Igarashi, *Orientation Code Matching For Robust Object Search*, IEICE Transactions on Information and Systems, pages.999-1006, 2001.

[VBR05] S. Vedula, S. Baker, P. Rander, R. Collins and T. Kanade, *Three-Dimensional Scene Flow*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, pages 475-480, 2005.

[VG05] Vaibhav Vaish, Gaurav Garg, Eino-Ville Talvala, Emilio Antunez, Bennett Wilburn, Mark Horowitz and Marc Levoy, *Synthetic Aperture Focusing using a Shear-Warp Factorization of the Viewing Transform*, in A3DISS 2005 (at CVPR), San Diego, 2005.

[VWJ04] Vaibhav Vaish, Bennett Wilburn, Neel Joshi and Marc Levoy, *Using Plane + Parallax for Calibrating Dense Camera Arrays*, in proc. of CVPR 2004, pages 2-9, 2004.

[W90] Lee Westover, *Footprint evaluation for volume rendering,* in proc. SIGGRAPH'90, pages 367-376, 1990.

[WAA00] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle, *Surface light fields for 3d photography*, in proc. of the 27th annual conference on Computer graphics and interactive techniques, pages 287-296, 2000.

[WJ05] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Mark Horowitz and Marc Levoy, *High Performance Imaging Using Large Camera Arrays*, in proc. of ACM SIGGRAPH 2005, vol. 24, pages 765-776, July 2005.

[WJK04] Jan Woetzel, Koch and Reinhard, *Multi-camera real-time depth estimation with discontinuity handling on PC graphics hardware*, in proc. of 17th International Conference on Pattern Recognition (ICPR 2004), pages 741-744, 2004.

[WLS02] Stephan Wurmlin, Edouard Lamboray, Oliver G. Staadt and Markus H. Gross, *3D Video Recorder*, in proc. of 10th Pacific Conference on Computer Graphics and Applications (PG'02), page 325, 2002.

[WSK02] Bennett Wilburn, Michael Smulski, Hsiao-Heng Kelin Lee and Mark Horowitz, *The Light Field Video Camera*, in proc. of Media Processors 2002, SPIE Electronic Imaging 2002, pages 29-32, 2002.

[WWS01] Michael Wimmer, Peter Wonka and François Sillion, Point-Based Impostors for Real-Time Visualization, 12th Eurographics Workshop on Rendering, pages 163-176, 2001.

[WY05] Huamin Wang and Ruigang Yang, *Towards Space-time Light Field Rendering*, ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D), page 125-132, 2005.

[YDM99] Yizhou Yu, Paul Debevec, Jitendra Malik and Tim Hawkins, *Inverse Global Illumination : Recovering Reflectance Models of Real Scenes from Photographs*, in proc. SIGGRAPH'99, pages 215-224, 1999.

[YEB02] Jason C. Yang, Matthew Everett, Chris Buehler and Leonard McMillan, *A real-time distributed light field camera*, in proc. of the 13th Eurographics workshop on Rendering, pages 77-86, Italy, 2002.

[YS05] Yosuke Ito and Hideo Saito, *Free viewpoint image synthesis using uncalibrated multiple moving cameras*, Computer Vision / Computer Graphics Collaboration Techniques and Applications (MIRAGE2005), pages 173-180, March, 2005.

[YS06] Satoshi Yaguchi and Hideo Saito, *Improving Quality of Free-Viewpoint Image by Mesh Based 3D Shape Deformation*, in journal of WSCG'06, the 14-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, pages 57-64, Prague, Czech Republic, January 30 - February 3, 2006.

[YSK02] Shuntaro Yamazaki, Ryusuke Sagawa, Hiroshi Kawasaki, Katsushi Ikeuchi, and Masao Sakauchi, *Microfacet billboarding*, in proc. of the 13th Eurographics workshop on Rendering, pages 169-180, 2002.

[YWB02] Ruigang Yang, Greg Welch and Gary Bishop, Real-Time Consensus-Based Scene Reconstruction using Commodity Graphics Hardware, in proc. of Pacific Graphics, pages 225-234, 2002.

[Z00] Z. Zhang, *A flexible new technique for camera calibration*, in proc. of IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, pages 1330-1334, 2000.

[Z97] Zhengyou Zhang, *Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting*, International Journal of Image and Vision Computing, Vol.15, No.1, pages 59-76, January 1997.

[ZBU04] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder and Richard Szeliski, *High-quality video view interpolation using a layered representation*, In proc. ACM SIGGRAPH, pages 600-608, august 2004.

[ZC04] C. Zhang and T. Chen, *A Self-Reconfigurable Camera Array*, Sketch presentation in SIGGRAPH 2004, Los Angeles, pages 243-254, Aug 2004.

[ZK01] Ye Zhang, Chandra Kambhamettu, *On 3D Scene Flow and Structure Estimation*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Kauai Marriott, pages 778-785, 2001.

[ZL00] Cha Zhang and Jin Li, *Compression of Lumigraph with Multiple Reference Frame (MRF) Prediction and Just-in-Time Rendering*, Data Compression Conference, pages 253-262, 2000.

[ZW94] Ramin Zabih and John Woodfill, *Non-parametric Local Transforms for Computing Visual Correspondence*, European Conference on Computer Vision (ECCV), pages 151-158, 1994.

Sites web

[netBREP05] *Boundary Representation Library (BREP)*, nouvellement renommée *GNU Triangle Surface Library (GTS)*, disponible sur http://gts.sourceforge.net

Annexes

Décomposition en valeurs singulières

Présentation

La décomposition en valeurs singulières, notée SVD de l'anglais *singular value decomposition*, est l'une des décompositions matricielles les plus utiles, surtout en vision par ordinateur. Elle permet d'effectuer de nombreuses opérations mathématiques telles que l'inversion d'une matrice et par conséquent la résolution de systèmes linéaires, le calcul de la matrice pseudo-inverse et donc la résolution de systèmes surdéterminés etc.

Soit A_{mn} une matrice de m lignes et n colonnes telle que $m \geq n$. La décomposition en valeurs singulières de A donne la factorisation suivante :

$$\begin{bmatrix} & & \\ & A_{mn} & \\ & & \end{bmatrix} = \begin{bmatrix} & & \\ & U_{mn} & \\ & & \end{bmatrix} \begin{bmatrix} & & \\ & D_{nn} & \\ & & \end{bmatrix} \begin{bmatrix} & & \\ & V_{nn}^{\mathsf{T}} & \\ & & \end{bmatrix}$$

Les matrices U_{mn} et V_{nn} sont orthogonales (ie. $U^{\mathsf{T}}U = Id$ et $V^{\mathsf{T}}V = Id$) et la matrice D est diagonale à éléments non négatifs. Par convention, on note la dernière matrice V^{T} plutôt que V. De plus, la matrice U préserve la norme : $\| U\mathbf{x} \| = \| \mathbf{x} \|$ quelque que soit le vecteur $\mathbf{x}$. La SVD peut tout à fait être appliquée sur une matrice A singulière.

Les éléments de la diagonale de D correspondent aux valeurs singulières de A, à ne pas confondre avec les valeurs propres :

$$A=UDV^{\mathsf{T}} \quad \Rightarrow \quad AA^{\mathsf{T}} = VDU^{\mathsf{T}}UDV^{\mathsf{T}} = VD^2V^{\mathsf{T}}$$

et comme V est orthogonale, on obtient :

$$AA^{\mathsf{T}}=VD^2V^{-1}$$

La matrice D^2 est diagonale et les éléments de sa diagonale sont les valeurs propres de AA^{T} associées aux vecteurs propres correspondants aux colonnes de V.

Enfin, certaines applications de la SVD nécessitent que les valeurs singulières soient triées par ordre décroissant. A chaque permutation de valeur singulière, il suffit de permuter les lignes correspondantes V^{T} et les colonnes de U.

Inverse et pseudo-inverse

Une des propriétés importantes de la SVD est la facilité avec laquelle elle permet d'inverser une matrice. En effet, à partir de la décomposition, il est facile d'inverser A car U et V sont orthogonales et D est une matrice diagonale. Pour inverser U et $V^\top$, il suffit de calculer leur transposée. En ce qui concerne D, il suffit d'inverser les éléments de sa diagonale s'ils sont non nuls et de laisser un 0 dans le cas contraire. Cette dernière règle correspond plutôt au calcul de la pseudo-inverse que de l'inverse de D. On obtient donc :

$$A=UDV^\top \quad \Rightarrow \quad A^{-1}=(UDV^\top)^{-1} = V^\top D^{-1} U^{-1} = V\left[\frac{1}{d_i}\right]U^\top$$

Dans le cas où A est singulière, ce calcul donne A^+ qui est la pseudo-inverse de A.

Grâce à cette propriété, la SVD permet de résoudre des systèmes linéaires $A\mathbf{x}=\mathbf{b}$. Trois cas sont envisageables :

- Cas d'une matrice A_{mn} avec $m = n$:

  Si A est régulière, la SVD inverse A et la solution est $\mathbf{x}=A^{-1}\mathbf{b}$.

- Cas d'une matrice A_{mn} avec $m \geq n$:

  Ce cas correspond à un système surdéterminé où il y a plus d'équations que d'inconnues. Il est rare que toutes les équations soient « en accord entre elles » (ie. rang $A = n$), cependant si c'est le cas, la SVD donne la solution exacte. Dans le cas contraire, la SVD calcule la pseudo-inverse A^+ de A qui résout le système au sens des moindres carrés.

- Cas d'une matrice A_{mn} avec $m \leq n$:

  Ce genre de systèmes n'a pas de solution unique, mais plutôt un espace vectoriel de solutions. La SVD permet là encore de trouver cet espace vectoriel à condition de compléter la matrice A avec des lignes de zéros de telle sorte qu'elle soit carrée.

La SVD est particulièrement utile en vision par ordinateur ou en synthèse d'image car elle permet de résoudre au sens des moindres carrés un système linéaire $A\mathbf{x}=\mathbf{0}$ en coordonnées homogènes sans donner la solution triviale $\mathbf{x}=\mathbf{0}$. La solution $\mathbf{x}$ est alors la dernière colonne de V.

Rang d'une matrice

Comme il est précisé plus haut, la SVD est un outil mathématique polyvalent qui permet d'une part de résoudre des systèmes linéaires mais aussi d'effectuer certaines opérations mathématiques bien utiles. La SVD d'une matrice M permet par exemple de calculer la matrice M' la plus proche de M mais de rang inférieur. En effet, il est possible de déterminer le rang d'une matrice en comptant le nombre de valeurs singulières non nulles dans sa décomposition. Pour diminuer le rang de la matrice, il suffit d'annuler sa plus petite valeur singulière. Cette application est utilisée en géométrie épipolaire pour imposer $\det F = 0$ afin que les droites épipolaires se croisent en un seul point. La SVD permet enfin de générer une base orthonormée à partir d'un espace vectoriel de départ.

Pour plus d'informations sur la décomposition en valeurs singulières, le lecteur peut se référer à [L95] Sonia Leach et Golub et Van Loan [GV96]. De multiples applications de la SVD à la vision par ordinateur sont présentées dans [HZ04] de Hartley et Zisserman. Enfin, une implémentation efficace (bien qu'illisible) de la SVD est disponible dans le *numerical recipes in C++* [PTV02]. Veillez bien à ce qu'il s'agisse de la seconde édition, la SVD proposée dans la première édition étant défaillante.

Historique des techniques de rendu d'images en relief

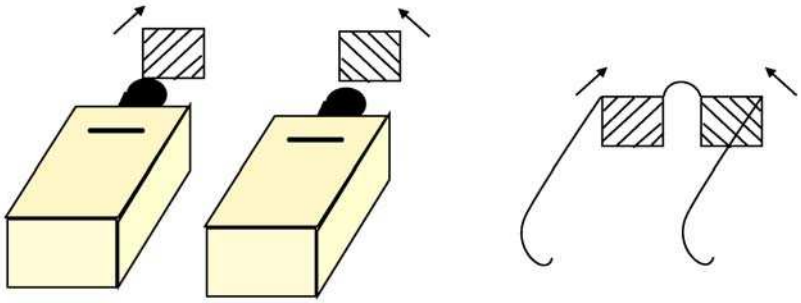
Voici un historique des méthodes et des dispositifs permettant de voir en relief. La quasi-totalité de ces méthodes nécessitent l'acquisition préalable de deux images stéréoscopique d'une même scène. Pour illustrer cette vue d'ensemble des techniques de stéréoscopie, nous avons choisi d'utiliser la fameuse théière de la bibliothèque graphique OpenGL.

- La méthode la plus ancienne est aussi la plus simple : placer les deux images l'une à coté de l'autre et loucher.

Vision croisée (il faut loucher)

Vision parallèle (il faut regarder derrière l'image/l'écran)

- 1815 : lunettes polarisées. Découvertes en 1815 et mises au point en 1917.



Le principe de polarisation permet des systèmes d'affichage assez souples. Les utilisateurs portent des lunettes polarisées, légères et peu coûteuses. L'écran est fabriqué avec une matière qui conserve et restitue la polarité des rayons projetés. Deux vidéo-projecteurs chacun muni d'un filtre, affichent les deux images stéréoscopiques. Les verres polarisés captent alors seulement l'image appropriée. Cette méthode permet l'utilisation d'images en couleurs (même si le vert passe moins bien que les autres couleurs) et plusieurs utilisateurs peuvent l'utiliser en même temps. Cette méthode est souvent choisie dans les cinémas dédiés à la 3d comme par exemple celui d'un grand parc d'attraction situé près de Paris.

Notons qu'il existe plusieurs sortes de filtres polarisants : les filtres linéaires polarisés en vertical et horizontal et les filtres circulaires polarisés dans le sens trigonométrique et dans le inverse.

- 1858 : anaglyphe découvert par Charles D'Almeida à Paris

Anaglyphe en rouge / cyan

Un anaglyphe est un couple d'images stéréoscopiques utilisant deux couleurs complémentaires (habituellement rouge et cyan). Cette technique fonctionne assez mal sur des images couleurs, notamment sur des images comportant du rouge. Elle est toutefois très facile à mettre en oeuvre et fonctionne très bien avec des images en « noir et blanc ».

- 1858 : les lunettes à obturateurs découvertes par Charles D'Almeira en 1858 et mises au point par Hammondet et Cassidy en 1922 aux USA.

Cache alternativement l'œil droit puis l'œil gauche et affiche simultanément à l'écran l'image associée à l'œil actif. Ce sont en général des cristaux liquides montés sur les lunettes qui génèrent l'obturation.

- 1896 : les réseaux tramés et les réseaux lenticulaires

Image de droite

Image de gauche

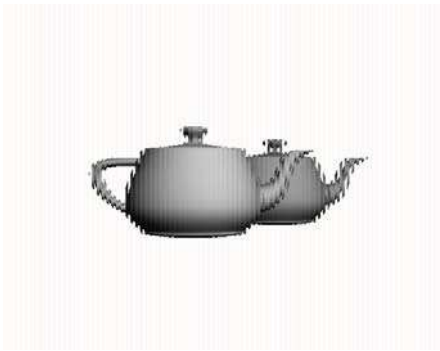
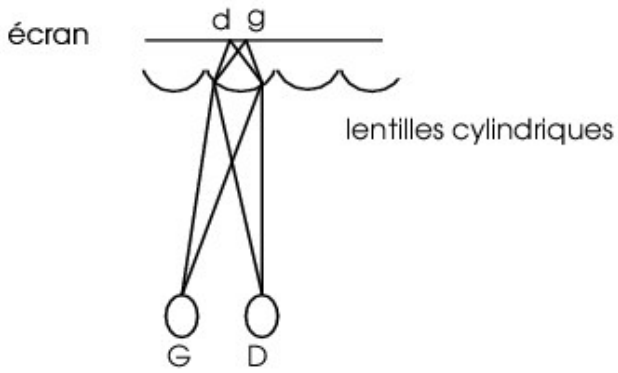


Image tramée

Dans un premier temps, les deux images stéréoscopiques sont découpées en fines lamelles. L'image finale est construite en intercalant les lamelles provenant d'une image avec les lamelles provenant de l'autre image. Cette image est alors placée derrière un réseau lenticulaire qui renvoie la couleur d'une des deux lamelle suivant la position de l'observateur.



Cette technique très prometteuse ne nécessite aucune lunette ou dispositif sur l'utilisateur.

- 1922 : Effet Pulfich.

Un filtre sombre devant l'œil droit crée du relief dans une l'animation en utilisant des propriétés de la persistance rétinienne. Cette méthode est principalement valable sur les travellings latéraux qui offrent un bon effet de parallaxe. C'est cette méthode qui fut utilisée sur TF1 pour l'élection de Miss France 96 ou bien sur la chaîne voyage.

- 1948 : l'hologramme découvert par Dennis Gabor et mis au point par Leith et Upatneik en 1960 aux USA.

Cette méthode utilise deux faisceaux laser. L'un est projeté directement sur une plaque photographique spéciale et l'autre est envoyé sur l'objet à holographier où il est réfléchi sur la plaque. Les interférences entre les deux faisceaux vont imprimer une image 3D sur la plaque. Pour restituer l'image, il faut envoyer un faisceau laser sous la plaque. Contrairement à une photographie où seule l'intensité est enregistrée sur la plaque, l'hologramme contient également une notion de distance (phase de l'onde) qui résulte de l'interférence avec le second faisceau. L'holographie numérique est un domaine émergent.

- 1960 : SIRDS (Single Image Random Dot Stereogram) aussi connues sous le nom de stéréogramme, d'auto-stéréogramme ou de *magic eye*. Découvert par le Docteur Bela Julesz et perfectionné par Christopher Tyler en 1979.

Image de départ

Carte de profondeur

Il s'agit d'une des rares techniques de stéréoscopie à n'utiliser qu'une seule image. Elle nécessite une carte de profondeur de l'objet à restituer. L'observateur louche devant l'image jusqu'à ce que les deux points noirs en haut de l'image se superposent. La méthode de création d'un stéréogramme s'appuie sur le fait qu'un même pixel représente un point 3d de la scène pour l'œil droit et un autre point 3d pour l'œil gauche soit en tout deux points 3d. Le point 3d correspondant à l'œil droit aura donc un projeté pour l'œil gauche qui sera de la même couleur que le pixel en question. Ainsi on peut en général observer un phénomène périodique sur les images générées. Petz et al. [PGM03] proposent une implémentation de stéréogrammes temps réelle utilisant le GPU.

Stéréogramme aléatoire

Stéréogramme avec motif

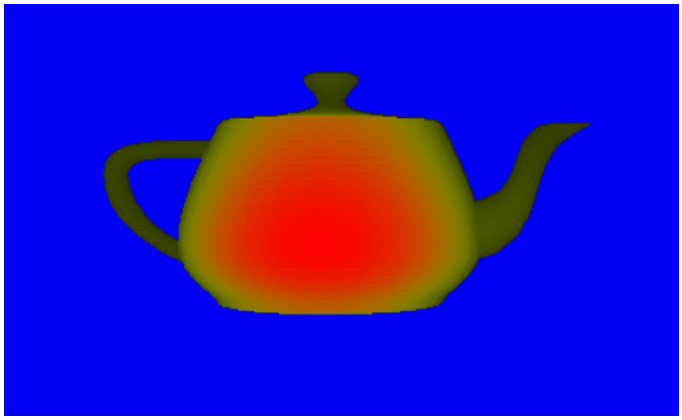
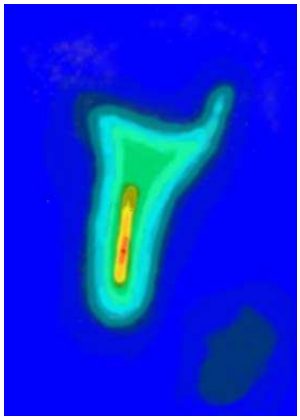
- Omniview : il s'agit d'un mode de restitution par images volumiques.

Cette méthode consiste à créer le « négatif » d'un objet 3d comme par exemple un masque que l'on regarderait coté intérieur. Quand l'observateur bouge, il a la sensation que le masque le suit du regard.

- Photostéréosynthèse :

Cette technique nécessite un appareil photo dont l'objectif a une profondeur de champ spécialement petite. Une série de photos de la même scène est alors prise en effectuant la mise au point à différentes profondeurs. Les images obtenues sont alors disposées sur des plaques semi transparentes ce qui donne une impression de volume.

- 1983 : Chromadepth inventé par Richard Steenblik.

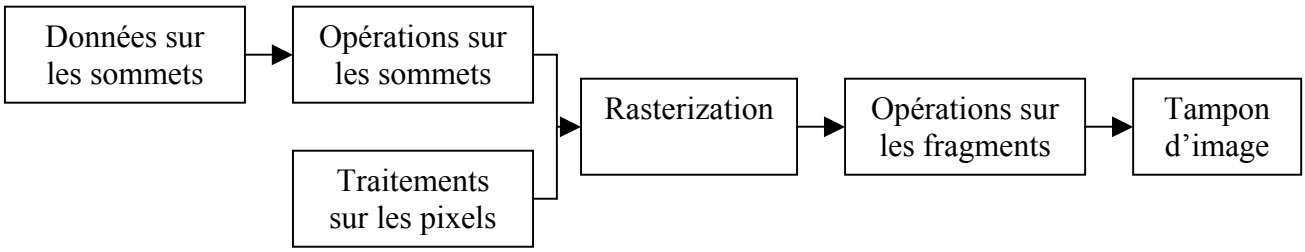



Ce dispositif nécessite une paire de lunette constituée d'une lentille mince diffractive et d'une autre réfractive (deux prismes, mais fins). On voit alors un objet plus ou moins loin selon sa couleur (dans l'ordre des couleurs de l'arc en ciel). Cette technique est très impressionnante sur une carte d'altitude (une île par exemple). A une époque, on pouvait trouver des lunettes pour Chromadepth dans des paquets de céréales !

Fragment shaders et vertex shaders dans le pipeline openGL

Le processus de restitution d'openGL

La plupart des implémentations d'OpenGL intègrent une série d'étapes de traitement regroupées sous le terme de « *OpenGL rendering pipeline* » dont un résumé est présenté dans le schéma suivant :



La première étape consiste à convertir un ensemble de sommets (points dans l'espace tridimensionnel) en primitives (points, lignes, polygones…). Dans un même temps les coordonnées de textures sont transformées via la matrice de texture et le modèle d'illumination est appliqué aux sommets.

En parallèle, un certain nombre de données présentes dans la mémoire de l'ordinateur sous forme de tableaux sont lues pour être intégrées au processus de restitution. Ces données peuvent être utilisées par exemple pour la création de textures ou alors pour mettre en correspondance un ensemble de valeurs chromatiques avec un rectangle de pixels de l'écran

La rastérization est la conversion simultanée de données géométriques et de pixels en fragments. Chaque fragment correspond à un seul pixel et possède des informations telles qu'une valeur de profondeur, des valeurs chromatiques... Dans l'étape qui suit la rastérization, les fragments subissent une série d'opérations qui permettent des les comparer entre eux (test de profondeur, stencil...) et de leur associer les textures correspondantes.

La dernière étape correspond au rendu final, les fragments ayant passé l'étape précédente sont stockés dans un tampon qui sera visible à l'écran. On ne parle alors plus de fragment mais de pixel.

Le processeur graphique (GPU)

Souvent sous-exploités, les processeurs des cartes graphiques sont des outils extrêmement performants (117 millions de transistors sur les cartes ATI Radeon 9800 PRO et 222 millions sur les cartes 6800GT de nVidia) qui peuvent travailler en parallèle du processeur principal. Ils peuvent donc apporter, par le biais des *shaders* qui sont des programmes spécifiques aux processeurs graphiques, un gain de performance non négligeable.

Principe

Comme nous l'avons vu précédemment, le pipeline d'OpenGL effectue des opérations sur les sommets et sur les fragments. Les *shaders* permettent de redéfinir temporairement, par le biais de programmes, une partie de ces opérations effectuées en privilégiant l'utilisation du processeur de la carte graphique plutôt que le processeur central d'où le gain de performances. Il y a deux types de programmes, ayant une syntaxe proche du C, qui composent les *shaders* :

- Le programme sur les sommets (ou *vertex shader*) s'emploie au niveau des opérations sur les sommets. Il reçoit un sommet sur lequel il va effectuer des transformations. Les étapes que le programme peut remplacer sont présentées dans le tableau suivant :

Fonctionnalités redéfinissables dans le processeur sur les sommets	**Fonctionnalités effectuées par *OpenGL***
Transformation des sommets	Limitation de la distance d'affichage (Clipping)
Transformation des normales	Afficher uniquement les faces avant (Culling)
Eclairage	Création des primitives
Application des couleurs d'un matériau	Sélection de l'éclairage sur deux faces
Limitation des valeurs des couleurs	Décalage des polygones
Génération des coordonnées de texture	Mode polygonal
Transformation des coordonnées de texture	Division de la perspective et cadrage

Le lecteur peut se référer à la figure 129 pour plus de détails sur ces étapes.

- Le programme sur les pixels (ou *fragment shader*) s'utilise au niveau des opérations sur les fragments. Les étapes que ce programme remplace sont l'application de texture, la sommation des couleurs et le calcul du brouillard. D'autres opérations comme le test de profondeur, le test de scission, le test de stencil ou le test alpha ne sont pas modifiées. Son rôle est de définir les caractéristiques finales des fragments. Le lecteur peut se référer à la figure 130 pour plus de détails sur ces étapes.

Enfin la figure 131 illustre la façon dont les *vertex shaders* et le *fragment shaders* s'insèrent dans le pipeline OpenGL.

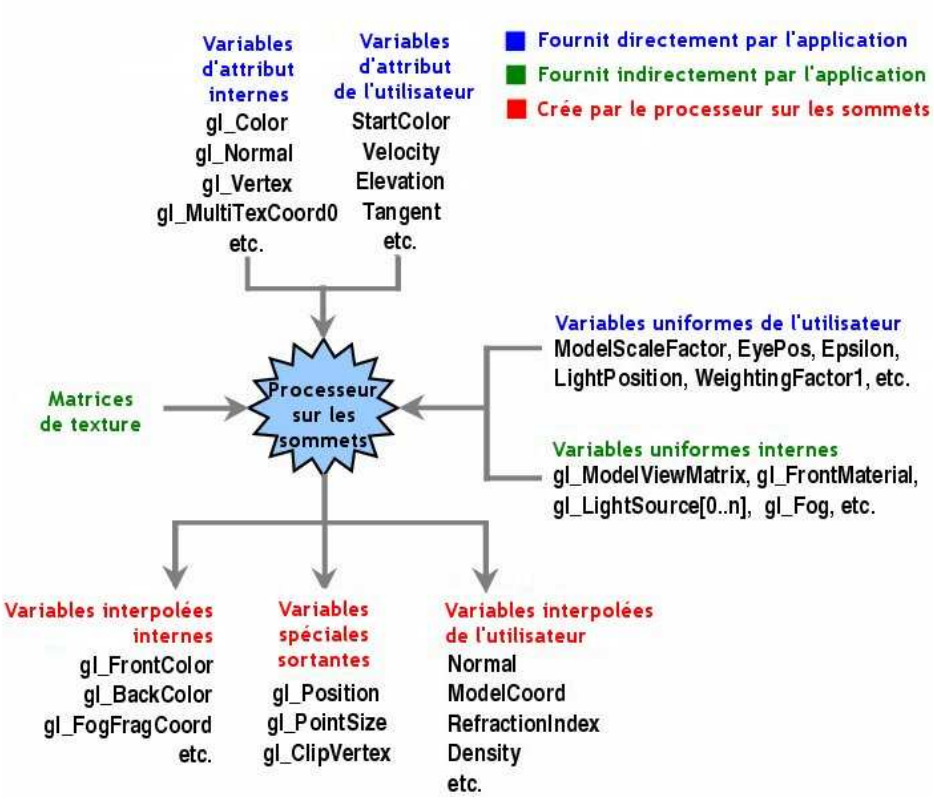


Figure 129 : attributs des vertex shaders.

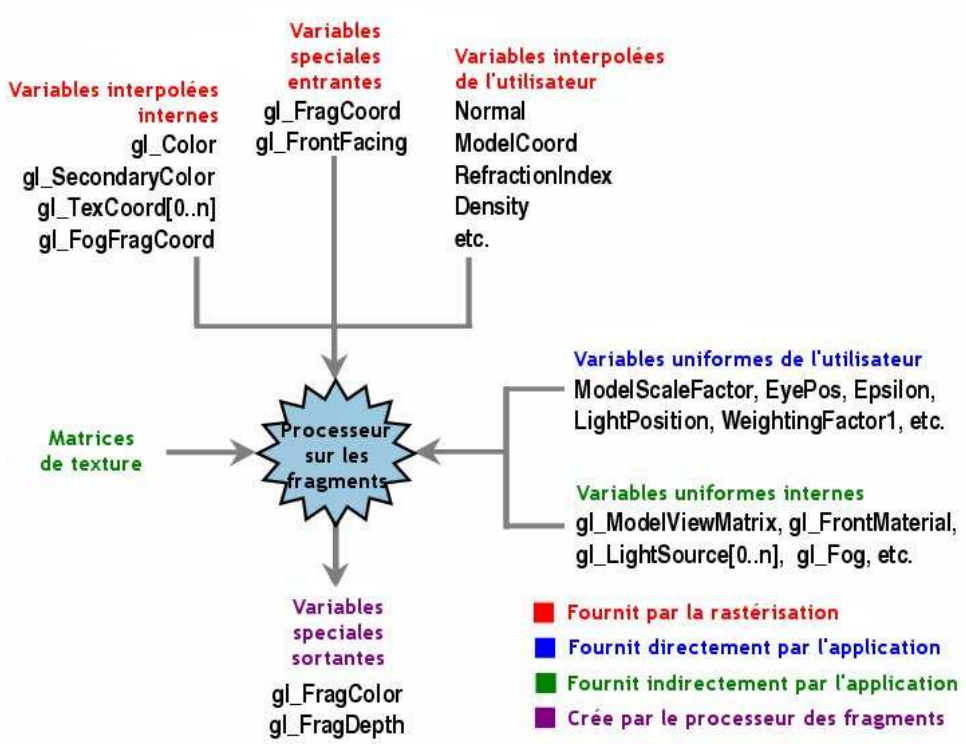


Figure 130 : attributs des frgments shaders.

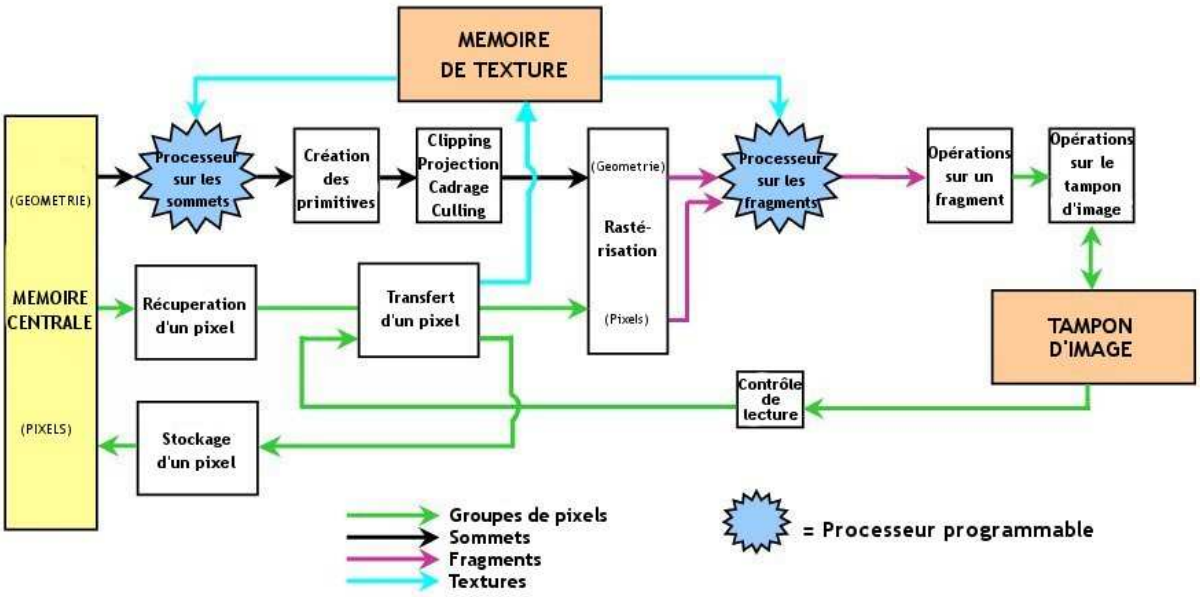


Figure 131 : vertex shaders et fragments shaders dans le pipeline OpenGL.

Les différents langages

En fonction d'une *API* (*Application Programming Interface*) ou d'un constructeur de cartes graphiques, des langages spécifiques ont été conçus :

- *Cg* [Nvi04] est un langage de haut niveau qui a été développé par le constructeur de carte graphique nVidia et est spécialement dédié aux produits de la même marque (son fonctionnement est limité sur les autres cartes graphiques). Ce langage repose sur l'utilisation d'un compilateur qui permet d'obtenir un code assembleur à partir d'un code écrit en *Cg*. Le code traduit est ensuite assemblé par les drivers de la carte graphique pour être utilisé dans l'application en cours de fonctionnement. Pour pouvoir utiliser le langage *Cg*, il faut posséder les extensions *OpenGL* suivante : *GL_ARB_fragment_program* et *GL_ARB_vertex_program*.
- *GLSL* [Ros04] ou *OpenGL Shading Language* est quant à lui dédié à *OpenGL* (à partir de la version 1.4). Le code source est envoyé à l'application *OpenGL* qui le compile, crée un exécutable (qui consiste à lier les *shaders* entre eux) et l'installe dans le processus de rendu. Toutes ces étapes se font à l'aide de fonctions *ARB* (qui seront probablement intégrées dans *OpenGL 2.0* [Ros03]). L'avantage de ce langage est qu'il peut être utilisé sur n'importe quelle carte graphique qui prend en charge les extensions suivantes : *GL_ARB_fragment_shader* et *GL_ARB_vertex_shader*. A l'heure actuelle, les cartes graphiques du type *ATI Radeon 9XXX* ne peuvent utiliser GLSL que sous un environnement *Windows* car la version d'*OpenGL* sous *Unix* pour les drivers *ATI* est encore 1.3.
- HLSL [PM03] est spécifique à l'*API Direct3D* (sous-couche, permettant la création de scènes 3D, de l'*API DirectX* de *Microsoft*) fonctionne sur le même principe de *GLSL* à la différence près que le compilateur n'est pas intégré dans les drivers.

RESUME :

Étant donné un ensemble de caméras filmant une même scène, le rendu à base de vidéos consiste à générer de nouvelles images de cette scène à partir de nouveaux points de vue. Dans cette thèse, nous détaillons en particulier la méthode des plane sweep sur laquelle porte la plupart de nos travaux.

La méthode des plane sweep consiste à discrétiser la scène en plans parallèles et à traiter séparément chaque point de ces plans afin de déterminer s'ils se trouvent ou non sur la surface d'un objet de la scène. Nous proposons un nouveau mode de calcul permettant de gérer les occlusions partielles tout en rendant la navigation de la caméra virtuelle plus souple. Nous proposons en outre une adaptation des plane sweep permettant de générer des images stéréoscopiques à moindre coût.

Enfin, nous proposons une méthode de suppressions des mouvements pseudoscopiques dans le cadre de l'utilisation des plane sweep en réalité virtuelle.

TITLE : Video-Based Rendering and Virtual Reality

ABSTRACT :

Given a set images of the same scene, the goal of video-based rendering methods is to compute new views of this scene from new viewpoints.
In this thesis,we focus on the Plane Sweep method. This method divides space in parallel planes and each point of each plane is processed independently in order to know if it lies on the surface of an object of the scene.
Our main contribution concerns Plane Sweep adaptations that handle partial occlusions and decrease the constraints on the position of the virtual camera. We also propose an adaptation of the Plane Sweep to stereoscopic rendering with low additional computation time. Finally, we propose a method that removes pseudoscopic movements in a virtual reality application.
All these methods reach real-time rendering using shader programs.

DISCIPLINE : Informatique

MOTS CLES : Rendu à base de vidéos, reconstruction, vision par ordinateur, temps réel.

INTITULE ET ADRESSE DE L'U.F.R.OU DU LABORATOIRE :

UFR arts et technologies,
Equipe SISAR,
6 cours du Danube,
77 700, Serris, France.