UNIVERSITE DE LIMOGES

ECOLE DOCTORALE Science – Technologie – Santé

FACULTE des Sciences et Techniques

Année: 2004 **Thèse N° 46-2004**

Thèse

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE LIMOGES

Discipline / Spécialité : Informatique

présentée et soutenue publiquement par

Damien PORQUET

le 29 Novembre 2004

Rendu en temps réel de scènes complexes

Thèse dirigée par : Djamchid Ghazanfarpour
Co-directeur : Jean-Michel Dischler

JURY:

Rapporteurs :	**M. Didier Arquès,**	**professeur à l'Université de Marne la Vallée**
	M. Michel Meriaux,	**professeur à l'Université de Poitiers**
Examinateurs :	**M. Jean-Michel Dischler,**	**professeur à l'Université de Strasbourg**
	M. Djamchid Ghazanfarpour,	**professeur à l'Université de Limoges**
	M. Dimitri Plemenos,	**professeur à l'Université de Limoges**

Remerciements

Je tiens à remercier Djamchid Ghazanfarpour, mon directeur de thèse, pour sa patience et la confiance qu'il m'a accordée.

Je ne saurais assez remercier Jean-Michel Dischler, professeur à l'université de Strasbourg, pour avoir accepté de co-encadrer ma thèse et qui m'a apporté aide et encouragements aux moments difficiles, sans compter son temps.

Je remercie vivement messieurs Michel Meriaux, professeur à l'université de Poitiers, et Didier Arques, professeur à l'université de Marne la Vallée, pour avoir accepté d'examiner mes travaux malgré leur emploi du temps chargé.

Merci également à la région Limousin pour son soutien financier durant ces trois années.

Merci à Sébastien, Romain et X-Man. Merci à Stéphane pour m'avoir fait découvrir Vi (entre autre), ainsi que pour les longues discussions et ses encouragements.

Merci à l'équipe de nuit du laboratoire MSI, à savoir Kosovar, Boucle-d'Or, Kevin "vivier", Yaye et Sylvain qui m'ont soutenu tout au long de la rédaction de ce mémoire et sans qui ces longues soirées de travail auraient été bien ternes.

Merci également à Suzie pour son aide précieuse dans les méandres de l'administration. Merci également à Laurent "passe-partout" pour ses encouragements.

Un grand merci à Philippe Picard, commissaire politique de la RAM[1], pour son système de protection antimoustiques Géorgien, d'une incroyable facilité d'utilisation.

Merci au Quick de Vanteaux pour l'intendance à toute heure.

Merci à MM. Linus Torvalds, Richard Stallman et Braam Moolenaar sans lesquels cette thèse aurait été bien plus difficile à terminer.

Boujnick !

[1] République Autonome de Migaria

A ma famille.

Table des matières

Introduction **1**

Chapitre 1 Rendu en temps réel de scènes complexes **7**

1.1 Introduction 7

1.2 Simplification de maillage 9

1.2.1 Simplification à niveaux de détails statiques 10

1.2.2 Simplification à niveaux de détails continus 11

1.2.3 Simplification dépendante du point de vue 14

1.3 Rendu à base de points 16

1.3.1 Le point 3D comme primitive de rendu 16

1.3.2 Surfels 18

1.4 Rendu à base d'images 20

1.4.1 Placage de texture 22

1.4.1.1 Placage de texture classique 22

1.4.1.2 Placage de bosselures 23

1.4.1.3 Auto-ombrage 23

1.4.1.4 Déplacement de surface 25

1.4.1.5 *Parallax Mapping* 31

1.4.1.6 Placage d'environnement 32

1.4.1.7 Placage de texture dépendant du point de vue 33

1.4.2 Imposteurs 36

1.4.2.1 Billboards et Sprites 36
1.4.2.2 Imposteurs statiques 38
1.4.2.3 Imposteurs dynamiques 39
1.4.2.4 Nailboards et imposteurs en couche 40
1.4.2.5 Imposteurs maillés 41
1.4.3 Déformation d'images 42
1.4.3.1 Reprojection de pixels 42
1.4.3.2 Déformation 3D d'images 43
1.4.3.3 Images à plans de profondeurs 44
1.4.4 Interpolation de points de vue 46
1.4.4.1 Avec connaissance implicite de la géométrie 46
1.4.4.2 Avec connaissance explicite de la géométrie 47
1.4.5 Interpolation de rayons lumineux 49
1.4.5.1 Fonction plénoptique 49
1.4.5.2 Reconstruction globale 50
1.4.5.3 Reconstruction locale 52
1.4.6 Autres techniques de reconstruction 53
1.4.6.1 Movie-maps 53
1.5 Conclusion 54

Chapitre 2 Extension au temps réel de l'interpolation d'images **57**

2.1 Introduction 57
2.2 Interpolation d'images de référence en fonction du point de vue 59
2.2.1 Principe 59
2.2.2 Points de vue de référence 59
2.2.2.1 Répartition et acquisition 59
2.2.2.2 Recherche des meilleurs points de vue 59
2.2.2.3 Schéma de pondération 60

2.2.2.4 Référentiel de mélange 61
2.2.3 Rendu 61
2.2.4 Résultats 62
2.2.4.1 Artefacts visuels 63
2.2.4.2 Interpolation des normales 66
2.2.5 Discussion 66
2.3 Interpolation de points de vue en temps réel 68
2.3.1 Principe 68
2.3.2 Interpolation avec précalcul du déplacement 68
2.3.2.1 Réalisation 69
2.3.2.2 Résultats 70
2.3.3 Méthode accélérée 71
2.3.3.1 Principe 71
2.3.3.2 Problématique 71
2.3.3.3 Réalisation 71
2.3.3.4 Résultats 72
2.3.4 Interpolation avec calcul au vol du déplacement 72
2.3.4.1 Problématique 72
2.3.4.2 Réalisation 72
2.3.4.3 Résultats 74
2.3.4.4 Discussion 76
2.3.5 Interpolation par blocs de pixels 76
2.3.5.1 Principe 76
2.3.5.2 Maillage régulier 76
2.3.5.3 Maillage adaptatif 78
2.3.5.4 Discussion 80
2.4 Conclusion 81

Chapitre 3 Textures directionnelles 4D **83**

3.1 Introduction 83

3.2 Textures directionnelles brutes 85

3.2.1 Création d'une texture 85

3.2.2 Structure des données 86

3.2.2.1 Discrétisation de l'hémisphère des directions 86

3.2.2.2 Quantifications 87

3.2.3 Rendu 88

3.2.4 Résultats 89

3.2.5 Discussion 91

3.3 Textures directionnelles de surfels quantifiés 92

3.3.1 Acquisition d'une texture directionnelle 92

3.3.1.1 Décomposition en surfels du modèle complexe 92

3.3.1.2 Calcul de la texture 93

3.3.1.3 Hémisphère des directions 94

3.3.2 Modèles d'éclairement implémentés 95

3.3.2.1 Repère local des points 96

3.3.2.2 BRDF par acquisition physique 96

3.3.2.3 Auto-ombrage par hémisphère d'ombrage quantifié 97

3.3.3 Représentation des données 98

3.3.3.1 Points 98

3.3.3.2 Textures directionnelle 99

3.3.3.3 Boîte englobante 99

3.3.4 Rendu 99

3.3.5 Résultats 99

3.3.6 Discussion 102

3.4 Conclusion 103

Chapitre 4 Interpolation de points de vue par pixel **105**

4.1 Introduction . . . 105

4.2 Interpolation pour une trajectoire fixée . . . 106

4.2.1 Principe . . . 106

4.2.2 Simplification de la surface . . . 107

4.2.2.1 Acquisition des échantillons . . . 107

4.2.2.2 Compression . . . 107

4.2.2.3 Décompression . . . 108

4.2.3 Reconstruction du relief et de la couleur de la surface . . . 109

4.2.3.1 Acquisition des points de vue de référence . . . 109

4.2.3.2 Interpolation des points de vue de référence . . . 109

4.2.4 Rendu . . . 110

4.2.4.1 Affichage . . . 110

4.2.4.2 Disparités de profondeur . . . 111

4.2.4.3 Algorithme multi-passes . . . 111

4.2.5 Résultats . . . 112

4.2.6 Discussion . . . 115

4.2.6.1 Avantages . . . 115

4.2.6.2 Inconvénients . . . 115

4.2.6.3 Perspectives . . . 115

4.3 Placage de relief en temps réel . . . 117

4.3.1 Principe . . . 117

4.3.2 Acquisition des points de vue de référence . . . 117

4.3.3 Rendu . . . 117

4.3.3.1 Calcul de l'auto-ombrage . . . 118

4.3.4 Résultats . . . 119

4.3.5 Discussion . . . 119

4.3.5.1 Avantages 121
4.3.5.2 Inconvénients 121
4.3.5.3 Travaux futurs 121
4.4 Conclusion 123

Conclusion et perspectives **125**

Annexe A Shaders **127**

A.1 Le pipeline de rendu 127
A.2 Vertex shader 128
A.3 Pixel shader 128
A.4 Unité de rastérisation 129
A.5 Caractéristiques des shaders 130
A.6 Langages 130
A.7 Extensions 131

Annexe B Compression de l'hémisphère d'ombrage d'un point **133**

B.1 Fonction d'ombrage 133
B.2 Transformée de Fourier 2D discrète 134
B.3 Harmoniques 134
B.4 Reconstruction de l'hémisphère 135

Bibliographie **137**

Table des figures

1.1 Le "GMan" du jeu vidéo *Half-Life*. A gauche, le modèle originel (environ 500 polygones, 1999), à droite le nouveau modèle (environ 5000 polygones, 2004) 7

1.2 Simplification géométrique ([Hop96]) 9

1.3 Réduction d'arête et séparation de sommets 12

1.4 Simplification avec préservation de l'apparence ([COM98]). 13

1.5 Objet 3D représenté sous forme d'un nuage de points (sous échantillonné pour les mettre en évidence) 16

1.6 Illustration d'un effet de parallaxe : la *désocclusion.* L'image de référence utilisée pour afficher la scène selon un autre point de vue ne contient pas assez d'informations et des trous apparaissent dans l'image à l'endroit ou on devrait voir une partie de la scène. 20

1.7 Les deux principaux types d'images de profondeur. A gauche : carte d'élévations. A droite : carte de profondeurs issue de la projection de la scène sur le plan de projection d'une caméra. 21

1.8 Les principales techniques de rendu à base de carte d'élévation ([WWT$^+$03]). 24

1.9 Calcul de l'auto-ombrage dans le relief induit par une carte d'élévation selon Heidrich *et al.* [HS99]. Les points noirs correspondent aux vecteurs dont le rayon intersecte le relief, tandis que les autres points représentent les directions pour lesquelles la surface au point C est visible. L'ensemble des directions, contenues dans le cercle de rayon $\overrightarrow{a}$ (la base de l'hémisphère des directions), sont partitionnées par une ellipse de rayons r_1 et r_2 contenant les directions pour lesquelles C est visible. Les points extérieurs à l'ellipse et contenus dans le cercle correspondent aux directions d'ombrage. 25

1.10 Déplacement de surface dépendant du point de vue 26

1.11 Définition d'une *VDM* ([WWT$^+$03]) 27

1.12 Affichage par pixel de surface déplacée ([HEGD04]). 28

1.13 Déplacement de surface basé image ([KS01]) 29

1.14 Déplacement de surface selon Schaufler ([SP99]) 30

1.15 Placage de texture en relief ([OBM00]) 30

1.16 *Parallax Mapping* 31

1.17 *Parallax mapping* avec et sans décalage borné 32
1.18 Placage d'environnement cubique 32
1.19 Mélange d'images 34
1.20 Pondérations utilisées dans [DTM96] 34
1.21 Schémas de pondération de Pulli *et al.* 35
1.22 *View-Based Rendering.* 36
1.23 Horry *et al.* [HAA97] 37
1.24 Décoret *et al.* [DDSD03] 38
1.25 Yamazaki *et al.* [YSK+02] 39
1.26 Imposteur. A gauche, l'objet original, à droite, l'imposteur. 39
1.27 *Imposteurs en couches* ([Sch98]) 41
1.28 *Imposteurs maillés* de Sillion ([SDB97]). 42
1.29 Principe de la rétroprojection de pixels 43
1.30 *3D Image Warping* 44
1.31 *Post-Rendering 3D Warping* [MMB97]. Les zones noires des images correpondent aux trous dûs aux effets de parallaxe. 44
1.32 *Morphing* [SD96] 46
1.33 *View morphing* ([SD96]) 47
1.34 *View Morphing* de l'image I_0 vers l'image I_1. L'image intermédiaire I_s est créée par *post-warping* de l'image $\hat{I}_s$ elle même obtenue par transfert linéaire entre les images $\hat{I}_0$ et $\hat{I}_1$. 48
1.35 *Images à multiples centres de projection* [RB98] 48
1.36 *Multiple Viewpoint Rendering* [Hal98] 49
1.37 Une tranche de lumière (*Light-Slab*) (à gauche) et ses deux plans positionnés dans l'espace de la scène (à droite). 50
1.38 Acquisition des échantillons d'images de référence pour la construction d'un *light-slab.* 51
1.39 Images panoramiques. 52

2.1 Répartition des points de vue sur la sphère. 60
2.2 Calcul de la contribution des points de vue de référence 61
2.3 Exemples d'images de référence obtenues avec notre méthode 61
2.4 Référentiel des images de référence 62
2.5 Mélange pondéré de couleurs de trois points de vue de référence. 63
2.6 Placage de texture dépendant du point de vue. Les trois points de vue de références utilisés sont éloignés les uns des autres d'un angle supérieur à 45°. La figure du centre montre les effets de *ghosting* qui apparaissent (principalement autour du visage). . . 64

2.7 Interpolation de deux points de vue. A gauche, l'image de départ ($t = 0$). A droite, l'image d'arrivée ($t = 1$). A centre, le résultat de l'interpolation à $t = 0.5$. 64

2.8 Artefacts visuels lors du mélange d'images 65

2.9 *Ghosting*. (a) : les trois points de vue de référence. (b), (c) et (d) : résultat de l'interpolation de textures selon différents angles, avec éclairage fixé. 65

2.10 Texture de normales 66

2.11 Placage de texture avec interpolation des normales pour le calcul de l'éclairement. Le maillage simplifié utilisé est composé de 3000 faces. 67

2.12 Principe de l'interpolation de points de vue : déplacement linéaire à l'écran des points de l'image $p(u_{0,}v_0)$ de l'image de départ, à la position (u_1, v_1) qu'il occupe selon le point de vue d'arrivée. 68

2.13 Interpolation avec précalcul du déplacement. Ligne supérieure : évolution des points du premier point de vue. Ligne inférieure : évolution des points du second point de vue. Colonne de gauche (pt. vue de départ) : $t \approx 0$, colonne du milieu : $t = 0.5$, colonne de droite (pt. vue d'arrivée) : $t \approx 1$. 69

2.14 Interpolation avec précalcul du déplacement. En haut, sans filtrage bilinéaire. En bas, avec filtrage bilinéaire de l'image. 70

2.15 Interpolation avec calcul au vol du déplacement des points. 73

2.16 Principe de l'interpolation. 73

2.17 Détermination du meilleur point de vue. 74

2.18 Interpolation avec calcul "au vol" du déplacement dans les parties de superposant des images de référence. De gauche à droite : $t = 0.0$, $t = 0.5$ et $t = 1.0$. 74

2.19 Utilisation d'une boîte englobante pour résoudre le problème des points extérieurs à l'objet. 75

2.20 Interpolation avec calcul "au vol" du déplacement. De gauche à droite : $t = 0.0$, $t = 0.5$ et $t = 1.0$. 75

2.21 Problème dus à l'interpolation linéaire : dans les cas complexes, l'approximation par un segment de l'évolution des points est impossible. En haut, les points de la scène balayés par le rayon optique issu d'un pixel de la caméra virtuelle ; en bas les points calculés. Les points rouges correspondent à $t = 0$ et les points verts à $t = 1$. Le modèle utilisé est le Dragon numérisé du Laboratoire d'Informatique Graphique de Stanford. 76

2.22 Maillage régulier. 77

2.23 Déplacement des sommets du maillage de l'image de départ vers la position qu'ils occupent à l'écran selon le second point de vue. 77

2.24 Attribution d'une profondeur virtuelle aux points extérieurs du maillage. 78

2.25 Partitionnement de l'image en un nombre fixé de niveaux de profondeurs servant à la subdivision du maillage. 79

2.26 Problème de la subdivision d'arête. 79

2.27 Application de la subdivision d'arêtes avec prise en compte des *T-vertices* (image (a)). Les images (c) et (d) correspondent à la subdivision appliquée à l'ensemble de l'image plutôt qu'au maillage régulier (b). 80

2.28 Comparaison des deux algorithmes de subdivision. A gauche, la première version générant des *T-vertices.* A droite, la version corrigée. 80

3.1 Comparaison ne notre méthode avec le placage de texture dépendant du point de vue. 83

3.2 Remplacement d'une partie de la scène par un polygone texturé (un imposteur). . . . 84

3.3 Précalcul : acquisition des informations géométriques pour chaque point des faces de la boîte englobante par lancer de rayons. 85

3.4 Précalcul : lancer d'un rayon par point, pour chaque direction d_i de l'hémisphère. . . 86

3.5 Discrétisation de l'hémisphère des directions. 87

3.6 Quantification des normales et des couleurs. 87

3.7 Rendu : reconstruction de la texture point par point en fonction des hémisphères précalculés stockant les intersections des rayons. 89

3.8 Résultat de la méthode appliquée à un modèle de tore, en rendu à ombrage plat. La définition de la texture (figure (c) et (d)) est de 128x128 texels, l'hémisphère est constitué de 256 directions quantifiées. Temps de précalcul de 8 mn environ. Rendu à 15 FPS. . 90

3.9 Résultat de la méthode avec une boîte englobante complète. La définition des texture est 64x64 texels. La résolution de l'hémisphère est 250 directions. 90

3.10 Création du nuage de points par subdivision récursive des triangles du maillage complexe. 93

3.11 Résultat de notre méthode de discrétisation appliquée à un tore à surface perturbée avec une carte d'élévation (par *déplacement de surface*) et à un maillage représentant un noeud torique. 93

3.12 Calcul d'une texture directionnelle 94

3.13 Variation de l'échantillonnage de la surface de la scène, avec des rayons issus de l'hémisphère des directions d'un texel de la boîte englobante. 95

3.14 Répartition adaptée des directions sur l'hémisphère. La répartition de l'angle d'élévation est basée sur une portion de la fonction racine carrée et permet d'augmenter le nombre de directions aux angles rasants. 95

3.15 Construction du repère
local d'un point. 96

3.16 BRDF par acquisition physique. 97

3.17 Résultats pour une variation de l'éclairement (de haut en bas). Définition des textures : 128x128 ; 1742 directions ; QLS : 4.97 Mo ; Hémisphères de visibilité : 326.81 Mo. Colonnes de gauche à droite : Phong (20 FPS), BRDF de Satin (15 FPS), BRDF synthétique (15 FPS). Toutes les images emploient l'auto-ombrage. 100

3.18 Méthode appliquée à un modèle composé de petits objets : le dirigeable, ligne du haut. Définition des textures : 64x64, Nb. directions : 4902, 50 FPS, pas de d'interpolation bilinéaire. Ligne du bas : résultat de la méthode appliquée au tore pour différentes définition de texture : 128^2 (20 FPS), 64^2 (50 FPS) et 32^2(70 FPS) de gauche à droite. 101

3.19 Modèles d'illuminations gérés par notre méthode. La définition des textures utilisées pour ces images est 128x128. Pour une qualité d'image maximale, nous avons positionné l'observateur à un pôle de l'hémisphère. De gauche à droite : Phong, Phong+placage d'environnement, BRDF satin, BRDF synthétique. 101

3.20 Application de l'auto-ombrage par compression en harmoniques principaux de l'hémisphère d'ombrage. Images de gauche : rendu standard (Phong) 30 FPS. Images de droite : application de l'auto-ombrage, 20 FPS. Le segment blanc indique la direction de l'éclairage. Définition de texture 128^2. 102

4.1 Trajectoire de la caméra. 106

4.2 Segmentation de la variation de profondeur d'un pixel de l'écran. Abscisses : abscisse curviligne de la caméra sur la trajectoire. Ordonnées : profondeur du pixel. 108

4.3 Principe de la reconstruction de la surface. 110

4.4 Disparités de profondeur. 111

4.5 Origine des points utilisés pour reconstruitre les pixels de l'image. 112

4.6 Résultats de l'interpolation. 113

4.7 Comparaisons de la méthode avec différentes compressions. 114

4.8 Application de la méthode au modèle *Armadillo*, compression sur 10 segments. Ligne du haut : interpolation à différentes position de la trajectoire. Ligne du bas : variation de la position de l'éclairage. 116

4.9 Acquisition des points de vue de référence. 118

4.10 Principe du placage de géométrie sur le modèle simplifié. 118

4.11 Application de notre méthode d'interpolation de points de vue au modèle du Buddha, composé de 1.1 million de faces. 119

4.12 A gauche, résultats de la méthode avec différents modèles simplifiés. A doite, application de la méthode au modèle de dragon. En bas à droite, vue détaillée de l'application de l'auto-ombrage. 120

4.13 Améliorations. Plutôt que de reprojeter un à un les points affichés d'un triangle t du maillage simplifié (en gris) sur chaque point de vue de référence, nous pourrions obtenir ces points directement en pré-projetant t dans les images de référence. 122

4.14 Echantillonnage d'un rayon optique $\overrightarrow{R}$. P_1 = le point que nous utilisons actuellement pour afficher P. P' = le point réel de la surface du maillage complexe de base. L'idée est de converger vers P' par échantillonnage de $\overrightarrow{R}$ à partir du point P. 123

A.1 *Vertex Shader* 128

A.2 *Pixel Shader* 129

A.3 Unité de rastérisation 130

B.1 Hémisphère d'ombrage. 133

Introduction

Les images virtuelles sont omniprésentes dans notre quotidien. L'industrie du cinéma, des jeux vidéos, de la publicité ou de la visualisation scientifique, pour ne citer que quelques exemples, en font un usage intensif.

Un des objectifs de la synthèse d'images est de reproduire visuellement la réalité. Dans le domaine de la post-production cinématographique, par exemple, l'avantage principal de ces images est que leur coût de production est bien moindre que leur équivalent réel : il est en effet plus facile de créer informatiquement une légion Romaine en ordre de marche en dupliquant l'image d'une vingtaine de figurants, que de rassembler effectivement cinq mille hommes. Dans le même ordre d'idées, les artistes peuvent créer des situations telles que la rencontre avec des dinosaures avec un "réalisme" impossible à obtenir à l'aide de marionnettes télécommandées.

Un autre domaine d'application des images de synthèse concerne les applications d'immersion, de *Réalité Virtuelle*, qui permettent, à l'opposé du cinéma, une interaction directe de l'utilisateur. Les simulateurs de vol, d'opérations chirurgicales, la visite de sites archéologique disparus ou les jeux vidéos en sont autant d'applications. Cette forme de simulation est plus immersive car l'utilisateur est libre d'interagir avec l'environnement simulé. Elle est également plus délicate à mettre en oeuvre car la génération des images doit satisfaire des contraintes de temps afin de préserver l'interactivité.

Le processus de création des images se décompose en deux étapes : la *modélisation* et le *rendu.* La modélisation consiste à définir la scène que l'on souhaite visualiser en données interprétables par l'ordinateur. Ces données, telles que la position des surfaces et leur nature ou l'éclairement de la scène, sont utilisées lors de l'étape de rendu pour obtenir des *image*s.

Ces images sont la projection visuelle*,* compréhensible par l'utilisateur, de modèles mathématiques et physiques de la réalité. Dans ce cadre, le *réalisme* d'une image est dépendant de la précision avec laquelle on définit la scène, ainsi que de la précision avec laquelle on simule la propagation de la lumière. En d'autres termes, pour obtenir une image visuellement réaliste d'une scène, il faudra d'une part la définir le plus précisément possible et d'autre part appliquer les modèles physiques de propagation de la lumière les plus complets. Idéalement, le but à atteindre est le *photoréalisme*, c'est à dire une image de synthèse indiscernable d'une photographie.

Bien sûr en pratique il est impossible de prendre en compte tous les phénomènes physiques : les ordinateurs ont un espace de stockage limité et les calculs doivent être accomplis dans un temps raisonnable. Ainsi, de très nombreuses méthodes existent dont le but est d'approximer les modèles physiques ou de contourner les limitations matérielles. Ces méthodes permettent d'obtenir différents niveaux de réalisme, en fonction des besoins.

Cadre de la thèse

Le cadre dans lequel s'inscrit cette thèse est le rendu en temps réel de scènes composées de modèles complexes non déformables, très finement détaillés. Ils peuvent être obtenus, par exemple, à l'aide de dispositifs d'acquisition 3D qui génèrent des millions de triangles [LPC+00], et sont alors impossibles à afficher en temps réel à cause du trop grand espace mémoire nécessaire et du temps de calcul requis.

Par rendu en temps réel, nous entendons les méthodes permettant comme au cinéma une fréquence d'affichage d'au moins 24 images par seconde (*FPS*, "Frame per second"). Ce temps ($1/24$ s) correspond approximativement à la persistance rétinienne de l'homme et permet de donner l'illusion du mouvement à partir de l'affichage séquentiel d'images fixes. Néanmoins, dans le domaine de la simulation d'environnements virtuels, l'affichage ne correspond qu'à la dernière partie du processus. Il est en effet nécessaire de tenir compte des calculs à effectuer pour la simulation physique de l'environnement (variation de l'éclairement, mouvement des objets, tests de collisions, tests de visibilité, etc...). Il est alors nécessaire que le processus d'affichage soit le plus léger donc le plus rapide possible : de 50 à 300 FPS peuvent alors être requis pour que le système soit utilisable.

A cet effet, les cartes graphiques permettent d'effectuer matériellement une partie du travail. C'est à dire que certains des algorithmes de base utilisés en rendu sont implantés dans un ou plusieurs processeurs auxiliaires (*GPU*, "Graphics Processing Unit"), indépendants du processeur central, et dédiés exclusivement à cette tâche. Ces cartes graphiques ouvrent de nouvelles perspectives. Il est désormais possible d'afficher interactivement des scènes composées de millions de primitives géométriques. De plus, des algorithmes comportant des calculs lourds peuvent maintenant être envisagés pour la synthèse d'images en temps réel et l'augmentation de la flexibilité de ces cartes permet d'implanter des algorithmes de complexité croissante.

Problématique

Bien que les performances du matériel doublent approximativement tous les ans, la demande de réalisme et donc la complexité des scènes est constamment croissante et dépasse toujours les capacités de calcul. Partant de cette constatation, d'autres approches pour contourner ce problème sont récemment apparues : le *rendu à base d'images* (*IBR*, "Image-Based Rendering") et le *rendu à base de points* (*PSR,* "Point Sample Rendering"). Ces deux approches proposent une représentation alternative de la scène permettant de reformuler l'expression de sa complexité géométrique.

Le principe du rendu à base de points est d'afficher une version discrétisée de la surface de la scène plutôt que des primitives géométriques. Les avantages de cette représentation s'expriment particulièrement dans le cas de l'affichage de très gros modèles géométriques (*MMR*, "Massive Model Rendering"). Comme les points sont traités isolément des autres, il n'y a pas besoin de maintenir une structure de connexité comme dans le cas de maillages géométriques : la mémoire utilisée est d'autant réduite et les techniques de niveau de détail (*LOD*, "Level Of Details",) sont plus faciles à mettre en oeuvre que les algorithmes de simplification de maillage géométriques. D'autre part, le rendu effectif d'un point isolé est très rapide comparé à la "rastérisation"[2] d'une primitive géométrique. Le problème principal inhérent à ce mode de représentation est qu'il ne garantit pas la cohérence topologique et géométrique (continuités) des modèles affichés. En particulier, des zones sous échantillonnées

[2]Conversion d'une image vectorielle en une matrice de pixels, stockés et manipulés comme des groupes de bits.

apparaissent si l'on se rapproche trop de la surface. D'autre part, il n'y a aucune information locale dépassant l'espace du point ce qui rend très difficile l'utilisation de cette méthode dans un environnement physique virtuel.

Le principe du rendu à base d'images est d'utiliser non plus des modèles géométriques pour générer une image, mais un ensemble pré-acquis d'images de la scène. Ces images de référence, prises à des positions données (*points de vue de référence*), sont utilisées pour reconstruire l'apparence de la scène selon d'autres points de vue. Ainsi, la complexité géométrique de la scène n'intervient pas dans le temps de rendu. Néanmoins, différents problèmes tels que le très grand espace mémoire nécessaire, les difficultés rencontrées pour faire varier l'éclairement de la scène ou les effets de parallaxe (occlusion, masquage) font qu'il est impossible d'utiliser cette approche telle qu'elle. De très nombreuses méthodes de rendu hybrides (*HIBGR*, "Hybrid Image-Based and Geometry Rendering") ont depuis été proposées. Elles permettent de combiner dans une certaine mesure les avantages du rendu conventionnel avec les avantages du rendu basé image.

Motivations

En rendu en temps réel, les objets sont traditionnellement représentés par leur surface (*B-Rep*[3]). Elles peuvent être modélisées par un maillage de polygones, des surfaces implicites, des NURBS[4] ou des surfaces de subdivision. Néanmoins, durant le rendu, ces primitives de haut niveau sont décomposées en un ensemble de triangles qui sont alors affichés par la carte graphique. La limitation principale de cette approche est le nombres de triangles affichables en temps réel. Ainsi, lors de la modélisation, il est souvent nécessaire de faire des sacrifices en baissant le nombre de polygones pour une partie donnée de la scène afin que l'ensemble puisse garder un taux d'affichage raisonnable. Bien sûr, baisser le nombre de polygones revient à dégrader la qualité du modèle.

Pour contrebalancer la simplification des objets, différentes techniques permettent de rehausser leur apparence de façon à donner l'illusion de l'objet original. Néanmoins, plusieurs problèmes font qu'il est difficile de les utiliser en l'état. La plupart ne fonctionnent que pour des cas particuliers concernant la nature de la surface à imiter ou donnent des résultats médiocres, suffisants seulement pour des vues lointaines. Quand elles donnent de bon résultats visuels, elles sont trop lourdes en mémoire et ne sont alors pas utilisable dans une application complète de simulation pour lesquelles le rendu doit prendre qu'une petite part des ressources. Le dernier écueil est que souvent ces méthodes demandent trop de temps de calcul pour une application de simulation d'environnement 3D interactive.

Nous nous sommes donc donnés comme base un modèle géométrique très complexe et nous nous sommes proposés de pouvoir l'afficher en temps réel avec une perte visuelle minimale et de pouvoir l'intégrer facilement dans un moteur de rendu traditionnel. C'est à dire faire en sorte d'utiliser au maximum les capacités de calcul délocalisé au niveau du pixel des cartes graphiques de façon à alléger le processeur central. De plus, pour pouvoir être facilement intégrable, les méthodes devront supporter le rendu par tampon de profondeur ("Z-Buffer") et gérer les modèles d'éclairement classiques ainsi

[3] Représentation par les *bords* des objets.

[4] *Non-Uniform Rational Basic Spline*. Outil mathématique permettant de créer des courbes paramétrées par des noeuds (points de contrôle) positionnés sur un plan. Les surfaces NURBS permettent de créer de la même manière des surfaces apparemment complexes simplement à l'aide de quelques points de contrôle.

que les techniques d'ombrage en temps réel.

Pour réaliser cela nous nous sommes attachés à combiner les avantages des différents modes de représentation de la scène (voir paragraphe précédent) en développant des techniques de rendu hybrides. Plus précisément, ce qui nous a guidé a été la volonté de délocaliser dans des images une partie du relief de la scène afin que le temps de rendu ne dépende plus de la puissance brute d'affichage géométrique de la carte et puisse ainsi s'effectuer en temps interactif.

Apports de nos travaux

Nous avons tout d'abord étudié, adapté et étendu les méthodes d'interpolation d'images au rendu en temps réel de modèles complexes en développant des structures permettant d'intégrer ces techniques à un moteur de rendu temps réel conventionnel, et en particulier de prendre en compte un éclairement variable de l'objet :

1. Premièrement, nous avons adapté les méthodes de placage de texture dépendant du point de vue (section 1.4.1.7) à des modèles très simplifiés d'objets 3D de façon à éviter les opérations manuelles inhérentes à ces méthodes, et de façon à pouvoir intégrer facilement le modèle à un environnement 3D temps réel. Nous avons également étendu le placage de texture dépendant du point de vue de façon à prendre en compte un éclairement variable des textures, effet non pris en compte dans le placage de texture dépendant du point de vue conventionnel. Nous avons mis en évidence les problèmes rencontrés et nous présentons les solutions que nous avons mises en oeuvre.
2. Deuxièmement, nous avons étudié les techniques d'interpolation de point de vue (section 1.4.4). De la même façon que pour les textures dépendantes du point de vue, nous avons étendu le procédé de façon à ce qu'il s'exécute en temps réel. De même, nous y intégrons un éclairement variable et permettons une intégration dans un moteur de rendu en temps réel conventionnel.
3. Troisièmement, nous avons étendu l'interpolation de points de vue de façon à ce que l'interpolation ne se fasse plus sur les pixels de l'écran mais sur des parties de l'image, des blocs automatiquement créés à partir du traitement automatique d'images de référence.

Nous avons ensuite étudié l'utilisation dans le domaine du temps réel des textures 4D se modifiant en fonction du point de vue, en les appliquant à un cube remplaçant le modèle géométrique de base. Dans ce cadre, nous avons étendu le *lancer de rayons virtuels* de Dischler[Dis98] de façon à ce que le précalcul et la génération des textures soit très rapide. Notre méthode se base sur une décomposition de l'objet originel en un nuage de points qui permettent de calculer la couleur des texels, et sur une structure de visibilité 4D qui permet de prendre en compte les effets de parallaxe. Les points s'apparentent à des *surfels* (voir section 1.3.2 de l'état de l'art) mais sont étendus de façon à pouvoir prendre en compte différents modèles d'illumination locale ainsi que l'*auto-ombrage* (section 1.4.1.3) de l'objet, et à occuper le moins d'espace mémoire possible. Nous avons appelé QLS (*"Quantized Light Surfel"*) ce nouveau type de surfel.

Finalement, nous avons développé une technique d'interpolation de points de vue originale qui permet de plaquer en temps réel de la géométrie sur une surface simplifiée d'un objet géométrique de façon à ce qu'il n'y ait que très peu de différence visuelle discernable entre l'objet simplifié et l'objet originel.

Nous nous basons pour cela sur un nombre restreint d'images de référence de l'objet complexe qui nous servent à stocker la géométrie de sa surface sous la forme de cartes de profondeurs. Le coeur de notre méthode est l'algorithme qui permet, sur la base de la version simplifiée de la surface du modèle et des images, de reconstruire au niveau du pixel la géométrie de la scène complexe.

Pour la capture des images de référence, nous avons développé une technique permettant de contrôler l'acquisition des meilleurs points de vue de façon à minimiser le nombre d'images nécessaires.

La surface simplifiée sur laquelle nous appliquons notre méthode d'interpolation de points de vue peut être une *image de profondeurs* représentant le relief approché du modèle original selon un point de vue donné, ou un maillage 3D simplifié :

1. Dans le premier cas, nous proposons une méthode permettant de retrouver une version approchée de la surface de la scène complexe pour une position quelconque sur une trajectoire prédéfinie au sein de la scène. Cette structure permet de compresser l'ensemble des variations de profondeur de chaque pixel, lorsque la caméra se déplace le long de cette trajectoire, sous une forme segmentée. Nous avons également mis au point un algorithme rapide de décompression s'exécutant en temps réel, bien qu'utilisant le processeur central. Cette reconstruction se fait au niveau du pixel et s'ajuste facilement en fonction de la précision voulue et de la mémoire disponible.
2. Dans le second cas, nous avons implémenté notre système d'interpolation de points de vues en employant la simplification de maillage sur le modèle complexe. Ceci nous permet de combiner la rapidité d'affichage du rendu du maillage simplifié à la qualité visuelle de l'objet complet. L'intégration dans un moteur de rendu conventionnel est immédiate, les modèles d'éclairement habituels peuvent être utilisés ainsi que des techniques réputées coûteuses en temps de calcul, telle que l'auto-ombrage.

Un des avantages de cette méthode est que le temps de rendu ne dépend plus de la complexité de la scène mais essentiellement du nombre de pixels affichés, c'est à dire de la surface qu'occupe l'objet sur l'image finale. D'autre part, l'algorithme d'interpolation étant effectué intégralement par la carte graphique, nous obtenons un taux d'affichage élevé ainsi qu'une excellente qualité visuelle comparée au modèle original.

Organisation du mémoire

Tout d'abord nous commençons par présenter un état de l'art des différentes techniques de rendu en temps réel de scènes complexes. Ce chapitre est séparé en trois parties principales.

- La première traite des techniques de simplification de maillage. Nous y présentons succintement, en les comparant, les principaux algorithmes caractéristiques des différentes approches existantes.
- La seconde porte sur les méthodes de rendu à base de points. De la même façon que pour les algorithmes de simplification de maillage, nous y présentons les principales approches de façon comparée.
- La troisième et plus importante partie porte sur le rendu à base d'images. Nous y faisons un survol des différentes méthodes, que nous classons en fonction de leur indépendance croissante vis-à-vis de la géométrie.

Le chapitre 2 traite des travaux que nous avons effectués sur l'extension au temps réel de méthodes d'interpolation de points de vue. Nous décrivons nos travaux portant sur le *mélange pondéré de couleurs*, la *migration de points en 2D*, le *déplacement de blocs d'images* et la *création d'un maillage 3D sur la base d'une image de profondeurs*.

Dans le chapitre 3 nous présentons nos travaux portant sur la génération et le rendu en temps réel de textures directionnelles 4D appliquées à une boîte englobante remplaçant le modèle complexe. Dans une première partie, nous présentons une version "brute" de la méthode. Dans la seconde partie, nous décrivons le modèle de représentation de la scène à base de points que nous avons développé : les éléments de surface et de lumière quantifiés. (*QLS, "Quantized Light Surfels"*). Ce modèle permet de générer rapidement les textures 4D et de prendre en compte différents modèles d'éclairement locaux.

Le chapitre 4 présente nos travaux portant sur l'interpolation de points de vue en temps réel. La première partie porte sur l'application de cet algorithme à un point de vue résidant sur une trajectoire fixée et la seconde partie porte sur l'application de la méthode à un modèle géométriquement simplifié.

En Annexe A nous présentons les *Pixels et Vertex Shaders*, fonctionnalités récentes des cartes graphiques qui permettent une souplesse inégalée dans le contrôle du pipeline de rendu et que nous avons utilisées tout au long de nos travaux.

En Annexe B nous détaillons la méthode que nous avons employée pour représenter l'hémisphère d'ombrage d'un point sur quelques octets, méthode que nous avons utilisée dans le chapitre 3.

1

Rendu en temps réel de scènes complexes

1.1 Introduction

Le réalisme des environnements virtuels interactifs augmente régulièrement d'année en année. Pour s'en convaincre, il suffit de regarder l'évolution des capacités des moteurs de jeux videos qui constituent une bonne vitrine des possibilités techniques en matière de rendu en temps réel.

Ce réalisme est fortement lié à la finesse de modélisation des objets constituant l'environnement. La figure 1.1 montre l'évolution technique de la complexité des modèles utilisés dans les deux versions successives du jeu vidéo *Half-Life* (©2004 Valve Corporation) : elle a été multipliée par dix en cinq ans.

FIG. 1.1 – Le "GMan" du jeu vidéo *Half-Life*. A gauche, le modèle originel (environ 500 polygones, 1999), à droite le nouveau modèle (environ 5000 polygones, 2004)

Cette hausse de qualité suit directement l'évolution des capacités de calcul des cartes graphiques qui peuvent traiter de plus en plus de données. Néanmoins, cette puissance de calcul brut ne fait pas tout. La demande en réalisme dépassant toujours les capacité des cartes, d'autres approches du problème de la complexité visuelle ont été développées pour optimiser l'utilisation des ressources de calcul. Ces approches permettent de reformuler le problème de façons plus adaptées.

Nous avons séparé cet état de l'art en trois parties principales correspondant aux grandes approches utilisées pour l'affichage d'objets géométriques complexes.

La première partie concerne les méthodes d'optimisation de maillage. Elles sont basées sur la constatation qu'il est inutile d'afficher l'intégralité des détails géométriques d'un objet si il est éloigné de l'observateur (approche *multi-échelle*). On utilise alors un maillage simplifié à la place du modèle complet pour une distance donnée d'observation. Quand le point de vue se rapproche du modèle, un maillage de plus en plus détaillé est utilisé, jusqu'à revenir au maillage d'origine. Ceci permet de réduire le nombre de polygones à afficher et donc d'optimiser les ressources. Néanmoins, ces algorithmes sont adaptés à des cas particuliers de modèles : une étendue d'herbe, par exemple, les met en échec.

La seconde partie concerne les méthodes de rendu à base de points. Ces méthodes n'utilisent plus un maillage de polygones pour représenter un objet, mais un nuage de points. Récemment, elles ont été développées pour effectuer le rendu interactif d'objets numérisés avec des dispositifs d'acquisition 3D générant de grandes quantités de points. Le rendu d'un point est extrêmement rapide comparé à l'affichage d'un polygone, et ces méthodes permettent d'afficher quasiment en temps réel des modèles composés de plusieurs centaines de millions de points. Néanmoins, pour une qualité visuelle optimale, le coût en temps de calcul est très élevé.

La troisième partie porte sur les techniques de rendu à base d'images. Nous avons classé les méthodes en fonction de leur indépendance croissante vis à vis de la géométrie. La plus simple des techniques à base d'images est le placage de texture. A l'extrême opposé, nous trouvons les techniques basées sur un échantillonnage de la fonction plénoptique, reconstruite par l'interpolation de rayons lumineux. A l'intermédiaire, nous décrivons les techniques de déformation d'images dont le principe provient de la Vision par Ordinateur. Ces techniques permettent de déplacer les pixels des images de façon à ce qu'elles redonnent les effets de parallaxe dus aux déplacements de l'observateur.

1.2 Simplification de maillage

Des techniques permettent d'optimiser l'affichage d'objets composés d'un très grand nombre de polygones en en réduisant le nombre, ce sont les techniques de simplification de maillage. Dans le cas de la simulation d'environnements virtuels, ces techniques sont utilisées quand l'objet à visualiser est loin de la caméra et que les détails fins de sa géométrie ne sont pas ou peu apparents et ne nécessitent donc pas d'être affichés. Par exemple, la figure 1.2 montre différentes versions d'un modèle de base composé de 10000 polygones. Chacune des versions simplifiées du modèle de base constitue un niveau de détail (*LOD*, "Level Of Detail") de celui-ci.

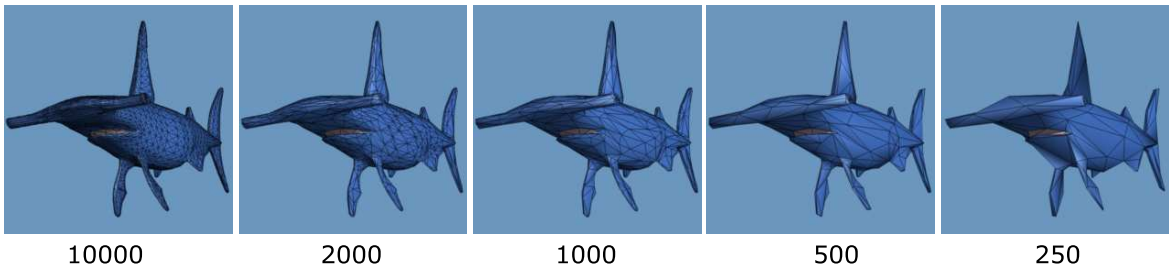


FIG. 1.2 – Simplification géométrique ([Hop96])

Ces techniques sont adaptées à différents cas de figure et peuvent être différenciées selon des critères tels que :

- le traitement de la topologie du maillage originel[5] :
  - L'algorithme peut nécessiter une surface topologiquement correcte (non dégénérée) pour fonctionner, ce qui peut s'avérer restrictif et contraignant. En effet, les modèles produits par les modeleurs ne garantissent pas toujours ces critères et les outils de réparation de surface (QSlim, [GH97]) ne fonctionnent pas dans tous les cas.
  - L'algorithme peut produire un maillage simplifié topologiquement identique à l'original, ou non. Dans ce dernier cas, des simplification drastiques peuvent être produites alors qu'un algorithme préservant la topologie du maillage originel empêche par exemple un tore de "se transformer" en sphère, alors que du point de vue de l'observateur, de loin, il n'y a quasiment aucune différence visuelle.
- la continuité de la transition entre deux niveaux de détails
- la prise en compte des attributs des sommets, comme les coordonnées de texture, qui sont modifiées lors de la simplification et affectent le résultat visuel final
- l'adaptation au point de vue, c'est à dire le but de la simplification : obtenir a tout prix un nombre fixé de polygones (orientation *budget*) ou obtenir un maillage simplifié minimisant l'erreur commise (orientation *erreur*)

On distingue trois classes d'algorithmes : les algorithmes à niveaux de détails statiques, les algorithmes à niveaux de détails continus et ceux qui s'adaptent au point de vue.

[5] Préservation du *genre* du modèle et des propriétés topologiques de sa surface telles que sa connexité, nombre de bords, orientabilité.

1.2.1 Simplification à niveaux de détails statiques

Principe La simplification statique correspond aux méthodes générant un ensemble discret de versions simplifiées d'un modèle complexe donné. Lors du rendu, il suffit de choisir le bon modèle à afficher en fonction de la distance d'observation. Différents types d'algorithmes ont été développés pour générer ces maillages simplifiés. Nous ne détaillerons pas leur fonctionnement en profondeur, mais décrirons leurs principales caractéristiques, avantages et inconvénients. Pour plus de détails, voir [Lue01], sur lequel est basée cette partie.

Méthodes existantes Une première classe d'algorithmes est formée par ceux qui emploient la *décimation* de maillage, comme [SZL92]. Le principe est de choisir des sommets dans le maillage complexe et de les supprimer. Les trous ainsi créés sont remaillés en employant diverses méthodes qui réduisent le nombre final de faces tout en préservant la topologie du modèle. L'algorithme procède en plusieurs passes en choisissant à chaque fois si un sommet peut être supprimé selon deux critères : le sommet sera enlevé si sa suppression préserve la topologie du maillage et si la retriangulation du trou qu'il crée donne des faces à une distance du maillage originel inférieure à un certain seuil.

Avec cette méthode, les maillages simplifiés créés sont des sous-ensembles du maillage de base, ce qui permet de préserver les attributs des sommets (couleur, normale, coordonnées de texture) De plus, la qualité des modèles simplifiés peut être améliorée en déplacant les sommets créés pour qu'ils améliorent l'aspect visuel final [GH97].

Néanmoins le fait qu'ils préservent la topologie du maillage empêche une très grande simplification. D'autre part, ces algorithmes sont très long à calculer.

Une autre type d'algorithme, le *regroupement de sommets* (*Vertex Clustering*) de Rossignac [RB93] procède, comme son titre l'indique, en regroupant certains sommets entre eux. Pour cela dans un premier temps, une importance est attribuée à chaque sommet du maillage complexe selon divers critères tels que la courbure de la surface en ce point ou la taille des faces qui lui sont connexes. Ainsi, des sommets attachés à de grandes faces ou des sommets de grande courbure seront considérés plus importants que les autres et donc seront gardés le plus longtemps possible. Ensuite, une grille 3D est superposée au modèle puis tous les sommets appartenant à une cellule sont regroupés au sommet le plus important de la cellule. Ainsi, plus la grille est grosse, plus la simplification est importante.

Des améliorations ont été proposées [LT97] permettant d'adapter localement la résolution de la grille, ou de calculer l'importance d'un sommet.

Cette techniques est très rapide ($o(n)$, avec n = le nombre de sommets) et ne demande pas beaucoup de mémoire.

Néanmoins, la topologie du modèle original n'étant pas préservée, les résultats visuels sont moins satisfaisants qu'avec la méthode précédente. De plus, on ne peut pas pas prévoir le nombre de polygones générés.

Une autre approche basée sur des cellules 3D (*Voxels*, Eléments de volume) a été proposée par [HHK+95]. La première passe consiste à "voxéliser" le modèle complexe, c'est dire à le transformer en une représentation volumétrique. Un voxel sera "plein" ou "vide" en fonction de la position

de son centre : à l'intérieur ou à l'extérieur du modèle. Ensuite, une filtre passe-bas est appliqué sur le volume, qui est ainsi rééchantillonné de façon plus grossière. Ainsi, les petites variations de géométrie (de haute fréquence) sont éliminées. Finalement, le maillage est reconstruit en appliquant l'algorithme du *marching cube* [LC87].

Cette technique ne préserve pas la topologie du modèle et a tendance à lisser les arêtes ce qui réduit son intérêt car on perd la silhouette du modèle.

La simplification par *surfaces enveloppantes* de Cohen [CVM+96] permet d'obtenir des très bons résultats graphiques en forçant la déviation géométrique entre le modèle original et le modèle simplifié à résider dans un espace fixé. Cet espace est défini en positionnant deux surfaces autour et à l'intérieur de la surface du modèle à simplifier. Ces surfaces sont créées en déplaçant la surface originelle d'un petit intervalle ε le long des normales des sommets : $+\varepsilon$ pour la surface externe et $-\varepsilon$ pour la surface interne. Cet intervalle est spécifié par l'utilisateur. Les deux enveloppes englobantes ne doivent pas s'intersecter : l'intervalle ε est modifié par sommet le cas échéant. Le processus de simplification est le même que la décimation de maillage : des sommets sont supprimés, mais les choix des sommets sont faits de façon à ce que le maillage résultant reste dans les deux enveloppes.

Le principal avantage de cette méthode est la fidélité du maillage résultant car la topologie est respectée. De plus, l'intervalle de simplification ε peut être basé sur la distance d'observation : croissant avec l'éloignement et fonction de la différence visuelle avec la surface originelle.

Néanmoins, les calculs sont lourds, les modèles ne peuvent pas être très simplifiés et la méthode n'est pas très robuste : si la surface n'est pas orientable ou topologiquement correcte[6], les calculs échouent.

Avantages De façon générale, ces algorithmes sont faciles à utiliser dans une application de réalité virtuelle interactive car les modèles sont précalculés à l'avance : on génère les différents niveaux de détails d'un modèle et on les utilise directement lors du rendu, en choisissant l'un ou l'autre en fonction de la distance d'observation.

Inconvénients Le plus gros problème de ces modèles pré-générés utilisés dans le cadre d'une simulation est que la transition entre un niveau de détail et un autre peut donner un effet de saut (*popping*) très gênant. Ceci est dû au fait qu'entre un niveau de détail et un autre, des polygones apparaissent ou disparaissent brusquement.

1.2.2 Simplification à niveaux de détails continus

Principe Pour résoudre ce problème de transition brutale entre les différents niveaux de détails, une autre classe de méthode est apparue. Le principe est non plus de générer séparément des maillages, mais de créer une structure générant au vol un maillage adapté au niveau de détail que l'on souhaite. Une telle structure est dite à *niveaux de détails continus.* (*CLOD,* "Continuous Level Of Detail"). La totalité des méthodes que nous présentons ici utilisent le principe de *réduction des arêtes* pour simplifier le maillage. Elles diffèrent sur le choix des arêtes à réduire, ou le moment de le faire. La réduction d'arête, et son dual la *séparation de sommet*, sont présentés sur la figure 1.3.

[6] Cf. note 5 page 9.

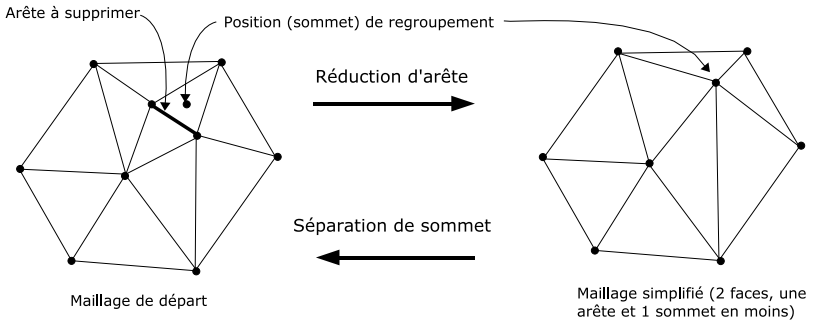


FIG. 1.3 – Réduction d'arête et séparation de sommets

Méthodes existantes La technique de *simplification avec préservation de l'apparence* [COM98] peut être vue comme une extension de la méthode de simplification par enveloppe. Au lieu de simplement contraindre la déviation de la surface à un intervalle donné, la déviation (erreur) maximale est basée sur plusieurs critères visuels de la surface simplifiée : la variation de la position de sa surface, la variation de sa couleur et la variation de sa courbure. La variation de couleur correspond aux distorsions induites entre une texture plaquée sur le maillage originel et la même texture plaquée sur le maillage simplifié[7]. La variation de courbure correspond à la perte de détails géométriques. Afin de minimiser la variation de ces deux derniers paramètres lors de la simplification, la couleur et la courbure du maillage originel sont enregistrées dans des textures utilisées par la suite pour atténuer l'erreur visuelle de la simplification. La courbure est représentée par l'ensemble des normales à la surface.

Finalement, le processus de simplification consiste à créer un maillage simplifié par réduction d'arêtes et à plaquer la texture de couleur et de normales sur celui-ci afin de minimiser les erreurs visuelles commises[8]. Le choix des arêtes à réduire est fait de façon à rester dans les limites autorisées.

Cette technique donne de très bon résultats visuels (figure 1.4) car le placage de la carte de normales redonne un aspect de relief saisissant (Cf. section 1.4.1.2 sur le *Bump-Mapping*). Cette méthode est employée actuellement dans divers moteurs de rendu en temps réel commerciaux et les toutes dernières cartes graphiques disposent de fonctionnalités dédiées à la compression de cartes de normales (ATI), spécialement utilisées dans le cas de la simplification de modèles géométriques.

Néanmoins, cette méthode ne permet pas une très grande simplification du modèle, car l'estimation d'erreur basée sur des critères visuels est très contraignante.

Garland et Heckbert [GH97] proposent une autre approche utilisant la réduction d'arêtes. Dans cette méthode, les sommets non reliés par une arête peuvent être regroupés : ils doivent se situer à une distance fixée par l'utilisateur. Parmi les paires de sommets candidates au regroupement, le choix est fait par rapport à un seuil d'erreur : le regroupement minimisant cette erreur est choisi. L'opération est effectuée itérativement jusqu'à ce que plus aucune paire de sommets ne soit regroupable. L'erreur commise lors d'un regroupement de sommets est évaluée selon un critère nommé Métrique d'Erreur Quadrique (*Quadric Error Metric*) qui représente la distance entre un sommet et l'ensemble de

[7] En pratique, la déviation de la couleur est évaluée comme une variation des coordonnées de texture.

[8] Cette technique est appelée *Bump-mapping* et est détaillée en section 1.4.1.2.

(a) Le modèle original

(b) Le modèle simplifié

(c) Le modèle simplifié sur lequel est plaqué une texture représentant le relief du modèle original

FIG. 1.4 – Simplification avec préservation de l'apparence ([COM98]).

ses faces adjacentes, le tout codé dans une matrice 4x4. Cette métrique est également utilisée pour positionner le nouveau point sur l'arête en minimisant l'erreur.

Les résultats obtenus avec cette méthode sont rapides et robustes car le maillage ne nécessite pas d'être topologiquement correct. De plus, cet algorithme donne de très bons résultats lors d'une très grande la simplification. Erikson et Manocha [EM99] améliorent encore les performances de l'algorithme en terme de temps de calcul et de préservation des attributs des sommets (coordonnées de texture, normale et couleur).

Lindstrom et Turk [LT00] proposent encore une autre approche pour calculer et minimiser l'erreur commise lors de la simplification. Ils analysent l'image obtenue lors d'une simplification par réduction d'arête avec l'image du modèle originel. Ils effectuent pour cela la simplification à un niveau de détail donné, puis comparent le résultat selon un nombre fixé de points de vues positionnés autour du modèle. Les images sont comparées en termes de luminance par rapport à l'image du modèle originel. D'une autre façon, on peut dire qu'ils évaluent l'impact visuel qu'une réduction d'arête peut avoir sur l'image. C'est idéalement la meilleure façon de procéder, mais le temps de calcul est énorme, bien que les résultats obtenus soient très bons.

Les maillages progressifs (*Progressive Meshes*) de Hoppe [Hop96] procèdent également par réduction d'arête mais généralisent le processus de plusieurs façons. Le choix des arêtes à réduire est basé sur l'évaluation d'une fonction d'énergie à minimiser. Par ailleurs, il enregistre dans une structure de données adaptée les opérations de réduction de façon à ce que l'opération inverse de création de sommet puisse s'effectuer avec la même fonction. De plus, il guide l'évaluation de cette fonction sur les attributs du maillage, qu'il sépare en deux catégories : les attributs discrets (indice du matériau,

identifiant de texture) et continus (position, coordonnées de texture, normale). Ainsi, la simplification s'adapte aux caractéristiques du maillage et plus uniquement à une erreur géométrique.

La rapidité du calcul est telle que l'algorithme peut être utilisé en temps réel directement [9]. De plus, la simplification adaptée aux attributs du maillage de façon unifiée donne de très bon résultats : la silhouette et l'apparence générale sont préservées. Néanmoins, comme tous les algorithmes respectant la topologie du modèle originel, les grosses simplifications ne sont pas possibles : les trous restent et jamais deux parties du modèle ne fusionnent.

Avantages Les avantages de la simplification continue de maillage sont donc en premier lieu que les problèmes de saut entre niveaux de détails disparaissent. De plus, il n'est pas nécessaire de stocker en mémoire les différents maillages car ils sont générés au vol directement.

Inconvénients Néanmoins ces algorithmes demandent dans l'ensemble beaucoup de temps de calcul. Même les *Maillages Progressifs* de Hoppe qui sont très rapides à calculer ne sont guère employés encore car dans un environnement virtuel ce n'est pas un seul modèle qui doit être simplifié mais souvent des dizaines en même temps. Dans ce cadre, les maillages statiques sont toujours utilisés car la vitesse de l'application prime sur l'espace mémoire demandé.

Un autre problème dont souffrent ces méthodes est leur inadéquation avec un environnement virtuel complet : elles ne simplifient qu'un modèle isolément alors que l'on imagine aisément que des groupes d'objets vus de loin pourraient se ramener à un seul maillage simplifié.

1.2.3 Simplification dépendante du point de vue

Principe Ces méthodes adaptent la simplification en fonction de la position d'observation. Ainsi, un modèle occupant différents niveaux de profondeurs à l'écran verra sa partie en avant-plan très détaillée et sa partie en arrière plan simplifiée.

Méthodes existantes Hoppe étend sa méthode des Maillages Progressifs en les raffinant en fonction du point de vue [Hop97]. Il utilise trois type de simplification : d'une part il supprime classiquement les parties du modèle hors du champ de vision ainsi que les faces non orientées vers l'observateur, d'autre part il adapte sa fonction d'énergie de façon à ce que l'erreur qu'elle évalue soit basée sur une déviation visuelle à l'écran. L'erreur visuelle commise par la déviation géométrique du maillage simplifié est bornée par un seuil fixé. Un des aspects positifs de cette évaluation d'erreur sur l'écran est qu'en particulier les bordures du maillage sont bien préservées.

Cet algorithme fonctionne assez rapidement pour du temps réel, mais est toujours adapté à un seul modèle.

La *Simplification Dynamique Hiérarchique* de Luebke et Erikson [LE97] permet, elle, d'effectuer la simplification de modèles quelconques et de scènes complètes (pas seulement d'un modèle isolé). La scène est composée d'un seul ensemble de sommets organisés sous une forme arborescente dont les noeuds stockent les sommets utilisés pour différents niveaux de simplification. Les critères d'erreur

[9] Il est d'ailleurs employé dans la bibliothèque graphique de Microsoft (DirectX)

sont une déviation géométrique à l'écran, la préservation de la silhouette des modèles et un budget de triangles à garantir. De plus, cette méthode est rapide et robuste car elle peut fonctionner sur des maillages dégénérés. Elle ne préserve pas la topologie des modèles, ce qui permet de simplifier énormément. Le pendant est qu'elle obtient de moins bon résultats visuels. Par ailleurs, étant donné que cette méthode utilise également la technique de réduction d'arêtes, les critères de réduction utilisés dans les méthodes précédemment décrites peuvent être adaptés.

Le champ typique d'utilisation des méthodes de simplification dépendant du point de vue d'observation sont les méthodes de rendu de terrain. Un terrain est un maillage créé par triangulation d'une *carte de hauteurs* (cf. section 1.4) sur lequel les critères d'erreur des techniques de simplification vues précédemment peuvent s'adapter. Garland propose une méthode permettant de raffiner l'affichage de ce maillage jusqu'à ce qu'un critère visuel soit atteint [GH95]. De même, Hoppe adapte ses *Maillages Progressifs* au rendu de terrain dans [Hop98]. L'affichage de terrains en temps réel est un domaine à part entière qui a généré beaucoup de travaux. Parmi ceux ci, les méthodes proposées par Lindstrom *et al.* [LKR+96] et celle de Duchaineau *et al.* (*ROAM*, "Real-time Optimally Adapting Meshes")[DWS+97] ont été très fertiles. Plus de détails peuvent être trouvés dans [AMH02].

Avantages Le fait d'adapter la résolution du maillage au point de vue permet d'éviter une simplification arbitraire : les critères de simplification sont adaptés au dispositif d'affichage, et correspondent à une erreur visuelle concrète.

Inconvénients Comme pour toute méthode évoluée, l'adaptation au point de vue de la simplification est plus lourde à calculer. Néanmoins, les avantages peuvent être drastiques, comme typiquement dans le cas de terrains.

1.3 Rendu à base de points

Avec la constante augmentation de la complexité des scènes, l'utilisation de maillages de primitives géométriques pour le rendu est apparue de moins en moins attractive comparée à la manipulation directe de points. De fait, lors du rendu, les traitements traditionnellement faits au niveau de la géométrie de la scène, comme l'élimination des faces cachées, deviennent très coûteux quand la scène est très complexe. Par ailleurs, plus la scène est finement détaillée, plus les triangles qui la constituent sont petits. Quand un de ces triangle est projeté sur l'écran, il est possible que sa surface apparente ne dépasse pas quelques pixels, pourtant tout le traitement standard (transformation des sommets, interpolation des couleurs, des normales, des coordonnées de texture et la rastérisation) sera appliqué indistinctement là ou il aurait simplement suffit de projeter un ou deux points.

1.3.1 Le point 3D comme primitive de rendu

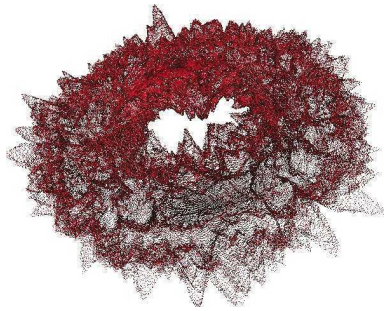


FIG. 1.5 – Objet 3D représenté sous forme d'un nuage de points (sous échantillonné pour les mettre en évidence)

Partant de ces constatations, Levoy et Whitted en 1985 [LW85] ont été les premiers à proposer d'utiliser des points comme unique primitive de rendu. Leur méthode permet de modéliser et d'afficher avec des points une surface 3D continue texturée.

Tout d'abord la scène définie par un maillage de polygones est transformée en un ensemble de points. Cet ensemble de points est généré par discrétisation régulière de la surface à l'aide des coordonnées de texture : il est donc défini comme une grille 2D de points de coordonnées (u, v) dans l'espace de la texture. Chaque point de cette grille stocke la position 3D de la surface (x, y, z) ainsi que la couleur (r, v, b) du point . Ce mode de représentation permet d'appliquer des déformations à l'objet, comme le déplacement de points[10] *(displacement mapping)* exemple typique de modélisation de surface très coûteux à afficher.

Cet ensemble de points déplacés est ensuite transformé et projeté à l'écran. La contribution de chaque point projeté à la couleur d'un pixel donné de l'écran est fonction de sa distance avec le centre de ce pixel, maximum au centre et décroissante en fonction de l'éloignement. Pour calculer cette contribution, la distance point/pixel est pondérée par une fonction d'atténuation, représentée par un filtre Gaussien. *Un point projeté contribue donc à plusieurs pixels de l'image, et est appelé* ***Splat*** [Wes90].

[10] La normale en un point (u, v), donnant la direction de déplacement, est calculée de façon conventionnelle, à l'aide des coordonnées de ses voisins, par dérivées partielles, ce qui nécessite que la surface soit différentiable.

Dans leur méthode, ceci permet d'éviter les trous dus aux zones sous échantillonnées[11], d'antialiasser les bordures de l'objet et de gérer la visibilité des points (avec l'utilisation d'un A-buffer [Car84] modifié). Les points peuvent être dessinés dans n'importe quel ordre, l'image se construisant au fur et à mesure qu'ils sont projetés. Ainsi Levoy et Whitted proposent dans leur conclusion d'utiliser leur méthode pour afficher la scène de façon graduelle : l'utilisateur visualiserait d'abord une version vectorielle (ou "fil de fer") de l'objet pour choisir son point de vue, puis quand il a choisi, les points seraient progressivement affichés de façon à donner une première version grossière de la scène. Ce principe permettant une visualisation en temps réel du modèle sera utilisé de façon étendue dans les *QSplat* [RL00].

Les avantages de cette méthode sont d'une part une haute qualité de rendu (bordures et surfaces antialiassées quelle que soit la finesse de la géométrie originelle) et d'autre part la rapidité d'affichage. De plus, comme les points 3D projetés ne sont pas connectés, un traitement en parallèle est possible. Néanmoins, l'image peut paraître floue quand le sous-échantillonnage est trop important ou que le noyau Gaussien utilisé est trop grand.

Grossman [Gro98] étend le principe de l'utilisation de points comme primitive de rendu et modélisant un objet sous la forme d'un ensemble dense de points. Les points sont acquis automatiquement à partir de la projection orthogonale d'un modèle géométrique conventionnel selon différents angles, jusqu'à obtenir un échantillonnage suffisant. Puis les points sont affichés et les trous comblés en utilisant la technique du *push-pull*[12] [GGSC96] sur une hiérarchie de tampons de profondeurs.

Le rendu à base de points est devenu plus récemment d'actualité avec l'apparition des scanners LASER qui numérisent une surface 3D sous forme de trames de points. Avec le projet Italien "The Digital Michaelangelo Project : 3D scanning of large statues" [LPC$^+$00], des modèles composés de centaines de millions de points sont devenus courants et leur affichage nécessitait une technique adaptée. L'utilisation d'un maillage polygonal standard basé sur un tel nuage de points est tellement coûteuse en temps de rendu qu'il est en pratique impossible de l'afficher en temps réel, sans parler de la taille en mémoire d'une telle structure. Avec les progrés récents des cartes graphiques, il est devenu possible d'effectuer le rendu de millions de points par seconde.

Rusinkiewicz et Levoy [RL00] ont développé un système de rendu à base de points appelé *QSplat* permettant d'afficher en temps réel de telles structures numérisées. Dans cette méthode, les points sont répartis dans une structure hiérarchique de sphères englobantes permettant d'effectuer les tests de visibilité et de gérer des niveaux de détails pour le rendu. Chaque noeud de l'arbre contient le centre de la sphère, son rayon, un cône de visibilité[13] et éventuellement une couleur[14]. Lors du rendu, l'arbre est parcouru jusqu'aux feuilles, représentant un point isolé, ou jusqu'à ce que l'aire de la sphère d'un noeud, projetée sur l'écran, soit inférieure à un certain seuil. Le seuil permet de contrôler le niveau de détail que l'on souhaite : plus le seuil est grand, plus on affiche les noeuds de faible profondeur, c'est à dire que l'on affiche une version approximative du modèle. Chaque sphère (noeud ou feuille) est donc projetée sur l'écran avec une taille basée sur son diamètre apparent. Le calcul de la couleur de la

[11] Pour un pixel de l'écran, la largeur du filtre Gaussien dépend du nombre de points qui y sont projetés. Moins il y a de points, plus le rayon augmente.

[12] La technique de push-pull permet de "réparer" une image dégradée en effectuant deux opérations successives de redimensionnement sur cette image. La première opération (*push*) réduit la taiile de l'image en interpolant les couleurs des pixels voisins. La seconde opération (*pull*) agrandit l'image jusqu'à sa taille d'origine, lissant au passage les défauts originels.

[13] Soit une normale et une largeur, cf. [Grossman98] pour plus de détails.

[14] Pour sauvegarder de l'espace mémoire, les modèles scannés peuvent être dessinés avec une couleur uniforme.

sphère projetée est basé sur le modèle d'éclairement de Phong et utilise les informations stockées dans le noeud (normale et couleur). La taille d'une sphère englobante de la hiérarchie est choisie de façon à ce qu'elle chevauche légèrement les sphères du niveau inférieur afin d'éviter les trous. Le rendu est effectué en utilisant le tampon de profondeur pour résoudre les problèmes de visibilité.

Trois différentes primitives sont utilisées pour l'affichage des points : le point OpenGL (un carré), le cercle (un polygone avec une texture représentant un disque) et un "point flou" (un *billboard*, cf. section 1.4.2.1) dont l'opacité baisse radialement selon une Gaussienne. Des ellipses orientées en fonction de la direction d'observation[15] peuvent être utilisées pour améliorer le rendu des bordures, mais elles génèrerent des trous et ralentissent les performances.

Au final, cet algorithme peut afficher de 1,5 à 2,5 millions de points par secondes sur des machines de milieu de gamme (Processeur de 1Ghz). Il permet également un affichage adaptif : l'utilisateur navigue autour de l'objet avec un affichage rapide et approximatif puis l'affichage se raffine automatiquement quand il arrête de se déplacer.

Les données (points, sphères et hiérarchies) sont toutes quantifiées et représentées sur un nombre optimal de bits de façon à réduire au maximum l'espace nécessaire en mémoire. Notamment, les positions des points contenus dans une sphère sont relatives à la sphère de niveau supérieur.

1.3.2 Surfels

Le concept de *surfel* [vB98, Oos98], une abréviation pour *élément de surface* ou encore *voxel de surface*, a été développé par [PZvG00]. Un surfel peut être vu comme un échantillon ponctuel de la surface d'un objet, composé d'attributs tels que : position, texture, normale, etc... . Comparé au nuage de points brut utilisé dans les méthodes de la section précédente, les surfels embarquent plus d'informations sur la surface échantillonnée, comme par exemple les coordonnées de texture, et sont liés au dispositif d'affichage. En particulier, la façon dont les surfels texturés sont filtrés dans l'espace de la texture avant d'être écrasés à l'écran par *splatting* différencie les méthodes à base de surfels, et dépend du dispositif d'affichage. Le filtrage de texture utilisé avec les surfels est le filtre par pondération moyenne elliptique (*EWA Filter*,"Elliptical Weighted Average")[Hec89] qui donne de très bons résultats visuels mais à un lourd coût en temps de calcul[PZvG00, ZGPvB01]. Différentes méthodes accélérées de ce filtre ont été présentées [GP03, RPZ02], mais leurs performances ne sont pas suffisantes dans le cadre d'une application de simulation d'environnement 3D, où plusieurs modèles doivent être rendus simultanéments.

Dans [PZvG00], un objet est représenté par des échantillons de surface composés d'une profondeur, une normale et différents niveaux de couleurs précalculés (issus de la texture appliquée à l'objet) appelés "*surfels mip-map*" ainsi que d'autres informations. Les surfels sont regroupés dans une structure à base d'octree appelée LDC-tree[16] . Cette structure est créée lors de la transformation de l'objet 3D en surfels et est basée sur trois vues orthographiques de l'objet.
Durant le rendu, une déformation 3D hiérarchique basée sur l'algorithme de McMillan (cf. section 1.4.3.2) est utilisée pour projeter les surfels dans le *z-buffer*. Le choix du noeud de l'octree contenant les surfels à projeter est fait en traversant l'octree de bas en haut et en estimant le nombre de surfels

[15] Une telle ellipse correspond à la projection dans l'espace objet du disque représentant le pixel, cf. section 1.4.1.1 sur le placage de texture.

[16] Un *LDC ("Layered Depth Cube")*, est composé de trois LDI ("*Layered Depth Image*", cf. section 1.4.3.3) orthogonaux dont les plans image partagent un même sommet du cube englobant l'objet.

projetés par pixel. Ainsi on peut choisir de n'utiliser qu'un surfel par pixel (rendu rapide) ou plusieurs surfels par pixel (sur-échantillonnage de l'image pour une meilleure qualité visuelle). Afin de résoudre les problèmes de visibilité, les projections orthographiques des disques tangeant des surfels sont rastérisés directement dans le *z-buffer*. Une nouvelle méthode appelée "*visibility splatting*" est utilisée pour détecter les trous, bien que l'on puisse indifféremment utiliser le sur-échantillonnage, le *splatting* ou le *push-pull*. Le rendu s'effectue assez rapidement car la rastérisation et le placage de texture sont délocalisés dans la phase de précalcul.

1.4 Rendu à base d'images

Les techniques de rendu à base d'images (*IBR*, "Image-Based Rendering") utilisent principalement des images comme primitives de rendu, c'est à dire que les informations contenues dans une image sont utilisées pour synthétiser de nouvelles images.

Le concept d'image a évolué graduellement. La texture, la plus simple des images (un tableau 2D de pixels, un pixel stockant une couleur) ne permet pas, quand elle est déformée, de prendre en compte les effets de parallaxe. Ces effets peuvent être pris en compte en ajoutant une profondeur à chaque pixel. Une telle image contenant couleurs et profondeurs est appelée *image de profondeurs* (*depth image*). Néanmoins, une seule image de profondeurs ne contient pas assez d'informations pour créer un nouveau point de vue : les effets de parallaxe génèrent, par exemple, l'apparition de certaines parties de la scène qui ne sont pas dans l'image de base (**effets d'occlusion** ou **de masquage**, voir figure 1.6). Pour prendre en compte ce problème, les *images à plans de profondeurs* (*LDI*, "Layered Depth Images") stockent par pixel, non plus simplement une couleur et une profondeur, mais une liste de couleurs et de profondeurs, c'est à dire un échantillonnage des parties cachées.

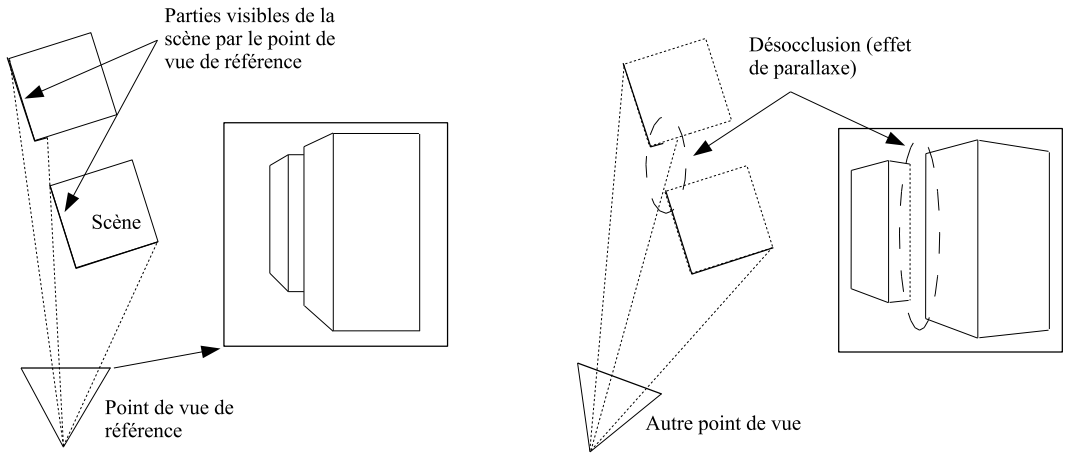


FIG. 1.6 – Illustration d'un effet de parallaxe : la *désocclusion.* L'image de référence utilisée pour afficher la scène selon un autre point de vue ne contient pas assez d'informations et des trous apparaissent dans l'image à l'endroit ou on devrait voir une partie de la scène.

Ces trois types d'images : *texture*, *image de profondeurs* et *image à plans de profondeurs* constituent les primitives de base des techniques de rendu à base d'image, et sont résumée dans le tableau 1.1. Par la suite, nous utiliserons le terme de *texel* (*Texture Element*) pour désigner les données stockées dans une texture. Dans notre cas, il ne s'agit pas des *Texels* au sens des textures volumiques de Kajiya-Neyret [Ney95, KK89].

Les images de profondeurs peuvent être de deux types en fonction de ce que représente cette information de profondeur. Le premier type correspond au *champ de hauteurs (Height Field)* ou *carte d'élévation,* utilisé par exemple dans le *placage de bosselures*(*Bump-mapping*) et représentant un déplacement orthogonal à la surface, par pixel. Le second type d'image de profondeurs est le *tampon de profondeurs* (*Z-Buffer*), dont la profondeur représente la distance caméra/surface.

Dans le premier cas (figure 1.7, image de gauche), la hauteur représente la distance entre une surface

Nom	Données par pixel
Texture	(R,V,B)
Image de profondeurs	(R,V,B,P)
Image à plans de profondeurs (LDI)	Liste de (R,V,B,P)

TAB. 1.1 – Types d'images et leurs composantes : R=Rouge, V=Vert, B=Bleu, P=Profondeur.

de référence et la surface réelle, sous forme d'une élévation perpendiculaire à la normale au plan de référence. Par la suite nous dénommerons une telle image de profondeurs par le terme *carte de hauteurs.*

Dans le second cas (image de droite), la profondeur d'un pixel correspond à la distance entre le centre de projection de la caméra et la surface. Cette distance est la composante z de la coordonnée 3D d'un point dans l'espace de la caméra. L'image de profondeurs correspond au *z-buffer* de la scène et nous utiliserons le terme *carte de profondeurs* pour dénommer une telle image.

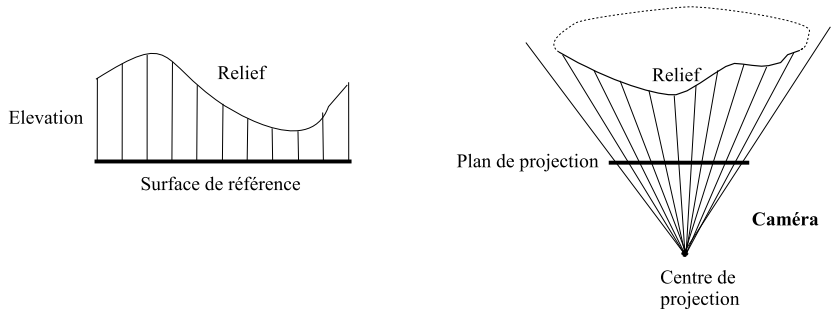


FIG. 1.7 – Les deux principaux types d'images de profondeur. A gauche : carte d'élévations. A droite : carte de profondeurs issue de la projection de la scène sur le plan de projection d'une caméra.

La façon dont sont utilisées ces informations permet de distinguer les algorithmes à base d'images. En effet, ils utilisent, déforment et combinent une ou plusieurs images selon différentes techniques qui vont du simple placage de texture au transfert épipolaire en passant par la déformation 3D ainsi que différentes méthodes d'interpolation. Dans cet état de l'art, nous détaillons ces techniques.

Les méthodes que nous allons décrire sont également caractérisées et différenciées par la façon dont il est fait usage des images :

1. La scène finalement synthétisée peut être simplement composée en partie d'éléments obtenus à l'aide de techniques basées images, ou être intégralement obtenue à l'aide d'images. Dans le premier cas, il s'agit par exemple de techniques telles que les *imposteurs* ou les *sprites* (section 1.4.2).
2. Les techniques de rendu basé image (IBR) peuvent être *pures*, c'est à dire n'utiliser que des images, ou être combinées avec les techniques de rendu standard, soit pour accélérer certaines techniques de rendu existantes, soit pour tirer parti des avantages des deux approches. Nous parlerons alors de techniques *hybrides* basées images / basées géométrie.
3. Finalement, les techniques d'IBR peuvent être distinguées en fonction de la façon dont sont obtenues les images utilisées pour les calculs.

(a) Elles peuvent être pré-acquises (comme dans le cas d'images réelles, par exemple des photographies) ou précalculées (dans le cas d'images virtuelles).

(b) Les images peuvent également être calculées "au vol", dynamiquement, lors du rendu, en fonction des besoins.

1.4.1 Placage de texture

Le placage de texture (*texture mapping*) est la plus simple et la plus ancienne des technique à base d'images. Le principe est de remplacer un objet géométrique complexe par un objet plus simple sur lequel on applique une image de la surface de l'objet complexe d'origine. C'est une technique puissante et éprouvée, actuellement intégrée dans toutes les cartes graphiques qui obtiennent dans ce domaine des performances impressionnantes et toujours croissantes[17].

Dans cette section, nous présentons le placage de texture et ses extensions que sont le *placage de bosselures* (*Bump Mapping*), l'*ombrage par carte d'horizon (Horizon Mapping)* et le *Déplacement de Surface* (*Displacement Mapping*). La technique du déplacement de surface a été implémentée de façon approchée dans plusieurs méthodes que nous détaillons.

1.4.1.1 Placage de texture classique

Le placage de texture a été introduit par Catmull [Cat74] et étendu par Blinn et Newell [BN76]. Le placage de texture permet d'ajouter du réalisme à une scène pour une relativement faible augmentation de la complexité du programme et du temps de rendu. Dans sa forme la plus simple, le placage de texture dépose une image (une texture) sur un objet de la scène. L'algorithme se décompose en deux phases : la transformation de l'image de l'espace texture à l'espace écran et le filtrage pour traiter les problèmes d'aliassage. Une description des différentes techniques de placage peut être trouvée dans [Hec86]. Le placage peut être effectué, entre autres, par balayage des pixels dans l'espace de l'écran (placage inverse) ou dans l'espace de la texture (placage direct). Le placage inverse est le plus employé en pratique et est implanté en particulier dans toutes les cartes graphiques.

Une fois que le placage a été effectué et la texture déformée sur la surface, l'image (la texture projetée à l'écran) doit être filtrée. Les techniques de filtrage de l'image, issues du Traitement du Signal, sont très nombreuses et les détails peuvent être trouvés dans [PM88, GW92]. En rendu en temps réel toutefois, les méthodes utilisées sont celles qui sont les plus rapides à calculer : interpolation *bilinéaire*, *mip-mapping*, filtrage *trilinéaire* et le filtrage *anisotropique*. Pour une description de ces algorithmes, voir [AMH02]. Le filtrage anisotropique est une approximation du filtre par moyenne pondérée elliptique (*EWA*,"Elliptical Weighted Average") introduit par Heckbert[Hec89], et est le filtre qui donne les meilleurs résultats visuels actuellement. Les cartes graphiques ne l'implémentent pas directement car il est trop coûteux en accès mémoire, mais utilisent différentes formes approximées de ce filtre en se basant sur quelques échantillons de l'ellipse. Les positions de ces échantillons (et leur nombre) peuvent être choisies de façon fixée, ou de façon stochastique, ou encore de façon guidée par l'orientation de la surface.

[17] Les performances du placage de texture des cartes graphiques sont calculées en termes de débit de remplissage *(fillrate)*, c'est à dire le nombre de pixels texturés affichés à l'écran, par seconde. Actuellement, la plus puissante carte graphique du moment (ATI Radeon X800) atteint un débit de 8,32 Gpixels/s. A titre de comparaison, les premiers accélérateurs graphiques atteignaient 90 Mpixels/s (3DFx Voodoo 2)

Le placage de texture ne fonctionne bien que pour des surfaces planes ou légèrement courbées. De plus, des objets réalistes trés détaillés nécessitent beaucoup de textures, qui peuvent de plus nécessiter d'être traitées en plusieurs passes, ralentissant d'autant les performances. Des phénomènes tels que la fumée, le feu ou l'eau sont difficiles à traiter en employant cette méthode.

1.4.1.2 Placage de bosselures

Le placage de texture ajoute des détails à une surface en en modifiant la couleur, mais cette couleur reste la même quelque soient les conditions d'éclairement, ce qui révèle que le relief représenté par l'image n'est qu'une illusion. Pour améliorer un peu les choses, Blinn [Bli78] a introduit le *bump mapping* (litt. "placage de bosselures"). Il s'agit d'une technique permettant de modifier l'apparence de la surface en fonction de l'éclairement, de façon a donner l'illusion d'une surface en relief. Le principe est de perturber la normale à la surface en fonction d'une carte de hauteurs. Cette carte de hauteurs permet de déterminer l'orientation de la normale en chaque pixel, normale qui est utilisée en lieu et place de celle de la surface lors des calculs d'éclairement. Ainsi, des zones sombres apparaissent sur la surface, ce qui est interprété par l'oeil comme de l'ombre dûe à du relief. Les résultats sont trés probants, bien qu'une fois encore, un angle rasant ou l'observation des bords (droits) montre qu'il s'agit d'une surface plane (cf. figure 1.8 (a) page 24).

Le placage de bosselures est implémenté dans toutes les cartes graphiques actuelles, bien que sous différentes formes simplifiées (*Emboss Bump Mapping*[18] [Sch94]) en raison de la perte de performances qu'il induit. En effet, il faut stocker et plaquer d'une part la texture de couleur, et d'autre part la carte de hauteurs. De plus, le calcul de la perturbation de la normale doit être évalué en chaque pixel, et il s'agit là d'une opération coûteuse. Depuis peu, le bump mapping peut être implémenté sans approximations car les calculs effectués par la carte graphique peuvent être programmés directement, au même titre que le processeur central.

Néanmoins, avec la hausse de la taille mémoire des cartes graphiques, le calcul de perturbation peut maintenant être précalculé et stocké directement. Au lieu d'utiliser une carte de hauteurs (une valeur de profondeur par texel) pour perturber la normale à la surface, on stocke directement la valeur de la normale (trois composantes (dx, dy, dz)) en chaque texel. Cette technique est appelée *dot3 bump-mapping* par référence au produit scalaire effectué entre la position de la lumière et la normale à la surface, lors du calcul d'éclairement de Phong. Les normales peuvent être exprimées dans l'espace objet (objet statique) ou dans l'espace tangeant[19](objet déformable) auquel cas la position de la lumière doit être transformée dans ce système de coordonnées lors du calcul d'éclairement.

1.4.1.3 Auto-ombrage

Ombrage par carte d'horizon Afin d'augmenter encore l'impression de relief, Max introduit la technique de l'*horizon mapping* [Max88]. Une carte d'horizon est créée sur la base d'une carte de

[18]L'idée de base de l'*embossing* (litt. "gravure") provient de l'observation que soustraire à une carte de hauteurs la même carte mais légèrement décalée donne l'apparence d'une surface bosselée / gravée. La direction du décalage est basée sur de la position de la lumière. Mathématiquement, cette soustraction dans la carte de hauteurs est une approximation de la dérivée de la hauteur dans la direction du décalage, c'est à dire une approximation du bump-mapping.

[19]L'*espace tangeant* est défini en un point $p(x, y, z)$ de la surface par une base formée par le vecteur $\overrightarrow{N}$ normal à la surface, la tangeante $\overrightarrow{T}$ et la binormale $\overrightarrow{B}$. $\overrightarrow{T}$ est orienté dans le sens croissant des coordonnées de texture, et $\overrightarrow{B} = \overrightarrow{N} \wedge \overrightarrow{T}$.

hauteurs. Cette carte d'horizon stocke, pour un nombre fixé d'azimuths (8 dans la publication originale), l'angle d'élévation à partir duquel un rayon issu du texel n'intersecte plus le relief. Lors du rendu, cette information est utilisée pour savoir si un pixel est dans une zone d'ombre ou pas en fonction de la position de lumière. L'impression de relief est encore plus saisissante que pour le bump mapping, bien que les bords du modèles révèlent qu'il s'agit d'une surface plane (cf. figure 1.8 (b) page 24).

Bien que Sloan et Cohen [SC00] aient proposé un algorithme en plusieurs passes permettant d'effectuer cette technique intégralement par la carte graphique, l'horizon mapping tel quel reste une technique coûteuse à utiliser car d'une part le rendu en plusieurs passes prend du temps, et d'autre part la carte de visibilité (d'horizon) occupe beaucoup d'espace mémoire.

(a) Bump mapping

(b) Horizon mapping

(c) Déplacement de surface conventionnel (subdivision en micro-polygones)

FIG. 1.8 – Les principales techniques de rendu à base de carte d'élévation ([WWT+03]).

Auto-ombrage par cartes de visibilité Dans [HS99], Heidrich et Seidel présentent une autre méthode permettant de calculer en temps réel l'auto-ombrage induit par une carte d'élévation. Ici, l'implantation de l'auto-ombrage est déduite de leur structure de visibilité qui permet également de prendre en compte les inter-réflexions de la lumière dans le relief de la carte de hauteurs. Pour la projection d'ombre, leur méthode commence par précalculer la visibilité de chaque texel de la carte de hauteur pour un certain nombre (fixé) de directions réparties sur un hémisphère centré sur le texel (image de gauche, figure 1.9). Ensuite, le vecteur unité de chaque direction est projeté sur le plan de base de l'hémisphère (image de droite). Etant donné la nature du relief, les directions intersectant la carte de hauteurs se retrouvent toujours à la périphérie du disque représentant la base de l'hémisphère. Leur méthode consiste alors à regrouper approximativement les points visibles dans une ellipse, et à stocker ses paramètres ($r_1$et r_2 sur la figure 1.9). Ainsi, les directions *en dehors* de l'ellipse correspondent à des rayons intersectant le relief, tandis que les directions *à l'intérieur* de l'ellipse correspondent à des directions d'observation pour lesquelles le texel est visible. Lors du rendu, pour savoir si le texel est éclairé par la lumière, la projection de la direction de la lumière est simplement comparée à l'ellipse : si elle est à l'intérieur de l'ellipse, le texel est éclairé.

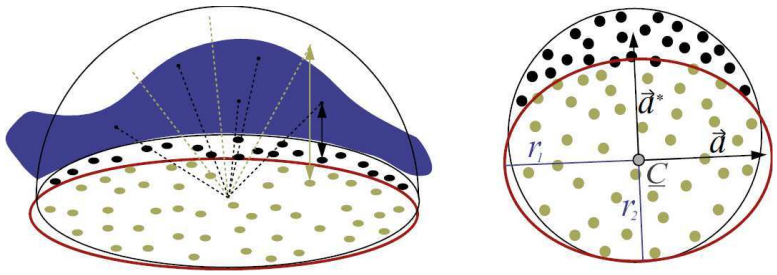


FIG. 1.9 – Calcul de l'auto-ombrage dans le relief induit par une carte d'élévation selon Heidrich *et al.* [HS99]. Les points noirs correspondent aux vecteurs dont le rayon intersecte le relief, tandis que les autres points représentent les directions pour lesquelles la surface au point C est visible. L'ensemble des directions, contenues dans le cercle de rayon $\overrightarrow{a}$ (la base de l'hémisphère des directions), sont partitionnées par une ellipse de rayons r_1 et r_2 contenant les directions pour lesquelles C est visible. Les points extérieurs à l'ellipse et contenus dans le cercle correspondent aux directions d'ombrage.

Tous ces calculs sont effectués à l'aide du matériel, notamment, le calcul de comparaison de position par rapport à l'ellipse est décomposé en quelques multiplications, additions et produits vectoriels effectués par la carte, les données définissant l'ellipse étant stockées dans une texture (les composantes de couleur de la texture faisant office de paramètre de l'ellipse). Les détails de cette méthode peuvent être trouvés dans [KHD00]. L'ensemble de la méthode fonctionne en temps réel (20 FPS à l'aide d'une Ge-Force 2, c'est à dire une carte ancienne) mais le taux d'affichage n'est pas suffisant pour une application interactive nécessitant plusieurs modèles rendus avec cette technique, et ce, malgré les évolutions des cartes graphiques. La taille de leur structure stockée dans les textures est d'environ 2 Mo pour 100 directions de l'hémisphère et une résolution de la carte de hauteurs de 32x32 texels.

1.4.1.4 Déplacement de surface

Principe Le déplacement de surface *(displacement mapping)* [Coo84] permet, à la différence des techniques précédentes, d'afficher réellement la surface du relief stocké dans une carte d'élévations. Ainsi, en particulier, les bordures de l'objet apparaissent réalistes (cf. figure 1.8 (c) page 24).

Les deux principales techniques permettant d'en effectuer l'affichage sont la subdivision en micro-polygones de la surface, et le lancer de rayons. La subdivision en micro-polygones se fait en fonction du point de vue : les parties de la surface proches de l'écran sont plus finement subdivisées que les parties éloignées. Depuis peu, les cartes graphiques permettent d'effectuer l'affichage d'une telle surface déplacée. La surface a afficher est transmise à la carte graphique, qui la subdivise en micropolygones et déplace leurs sommets en fonction des données stockées dans la carte de hauteurs. Finalement, l'ensemble des micropolygones sont affichés de manière conventionnelle.

Cette technique est extrêmement coûteuse en puissance de calcul car la subdivision des polygones peut nécessiter d'être très fine, si la carte d'élévations est trés perturbée, afin de rendre tous les effets de relief. Néanmoins, cette méthode est bien adaptée à des reliefs de moyenne échelle (*mésostructures*[20]), exprimable sous forme de champ de hauteurs. Finalement, l'avantage principal de cette approche (qui

[20]Le terme de mésostructure est employé pour désigner un relief trop grand pour être modélisé par une fonction de réflectance et trop petit pour être modélisé efficacement par des polygones.

a motivé son invention) est que l'espace mémoire nécessaire est minime comparé au relief produit. La création d'un tel relief de façon conventionnelle serait en effet prohibitive car elle nécessiterait de stocker et de transmettre à la carte graphique des millions de triangles.

Dernièrement, des techniques permettant d'effectuer le rendu en temps réel de telles surfaces déplacées ont été proposées [WWT+03, HEGD04]. Elles opèrent différemment à la subdivision en micropolygones en ce sens qu'elles déportent le problème dans l'espace de l'écran : au lieu d'afficher la surface sur laquelle est plaquée la carte d'élévations, elles cherchent à déterminer la position dans la carte d'élévation d'un pixel de l'écran. Le problème est inversé, et, de façon générale, correspond à l'approche qui caractérise beaucoup de méthodes actuellement proposées en rendu basé images en temps réel. Cette approche est motivée par le fait qu'il est plus efficace de résoudre les problème de visibilité *par pixel* plutot que par des traitement conventionnels dans l'espace de la scène, car la résolution d'un écran est fixe, relativement faible (généralement de 1024x768 pixels) et n'évolue quasiment pas (pas d'évolution significatives depuis 10 ans) tandis que parallèlement, la puissance de calcul des cartes graphiques, ainsi que leur flexibilité[21], double tous les ans, voire les six mois. Un traitement effectué par pixel, permettant de synthétiser une image, est, dans une certaine mesure, indépendant (en termes de temps de calcul) de la complexité de la scène : seuls les calculs effectués par pixel comptent. Ils doivent donc être très rapides et simples, mais de telles méthodes permettent au final d'obtenir des résultats impressionnants tant en termes de rapidité que de qualité.

Déplacement de surface dépendant du point de vue Ainsi, Wang *et al.* [WWT+03] ont développé une technique permettant d'effectuer du déplacement de surface en délocalisant les calculs au niveau du pixel, c'est à dire dans l'espace écran. Un modèle géométrique paramétré de façon conventionnelle est affiché (figure 1.10 (a)), et la couleur de chaque pixel affiché de sa surface est calculée par l'algorithme. Afin de déterminer cette couleur, issue de la carte de déplacement, leur algorithme se base

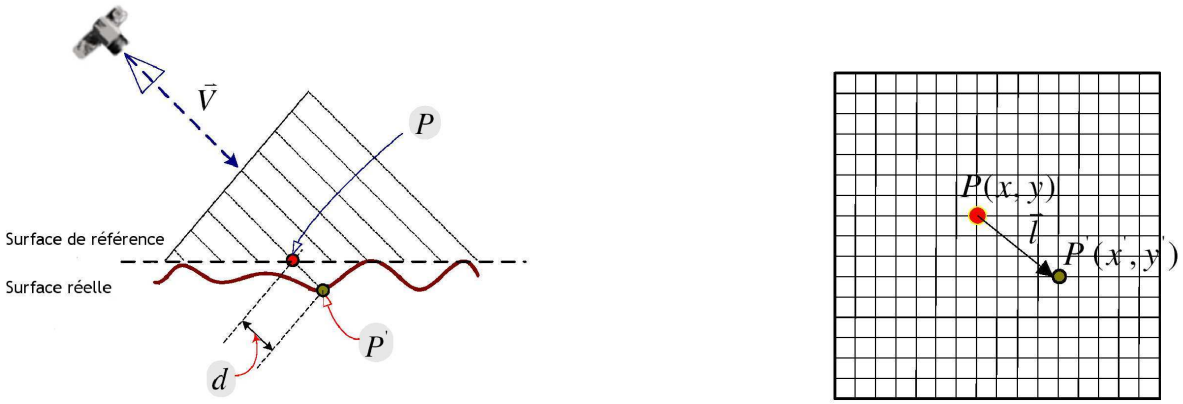
(a) Maillage englobant

(b) Déplacement de surface dépendant du point de vue

FIG. 1.10 – Déplacement de surface dépendant du point de vue

[21] Par *flexibilité* d'une carte graphique, nous entendons le nombre d'opérations autorisées par pixel, c'est à dire la taille du programme exécuté par le GPU appelé *shader* (Voir les annexes), ainsi que la nature des opérations possibles (opérations arithmétiques ou d'accés mémoire).

sur un ensemble de cartes de profondeurs de la surface réelle, précalculées par lancer de rayons et combinées dans une texture spéciale à cinq dimensions appelée VDM (*View-dependent Displacement Map*)[22].



(a) (b)

FIG. 1.11 – Définition d'une *VDM* ([WWT+03])

Cette VDM représente une fonction permettant, étant donnés un pixel P destiné à être affiché (cf. figure 1.11) et la direction d'observation $\overrightarrow{V}$ par rapport au plan tangeant de la surface, de trouver l'origine dans la carte de hauteurs de la partie déplacée à cet endroit, c'est à dire le point $P^{'}$.

Pour cela, cette texture stocke la distance d entre la surface englobante de référence et la surface réelle, pour chaque point $P(x, y)$ de la surface de référence, en fonction de la direction de vue $\overrightarrow{V}(\theta, \varphi)$ (en coordonnées sphériques). Cette texture étant destinée à être plaquée sur une surface quelconque (non plate), la courbure c de la surface réelle (au point P') est également stockée afin de prendre en compte les distorsions induites dans la carte de hauteurs. Ainsi, une VDM représente une fonction à cinq dimensions : $d_{VDM}(x, y, \theta, \varphi, c)$. Cette fonction permet de trouver, lors du rendu, pour une position d'observation donnée, le décalage $\overrightarrow{l}$ (figure 1.11(b)) apparent des points de la surface réelle, c'est à dire l'image de la surface réelle selon le point de vue d'affichage.

Une telle texture à cinq dimensions est discrétisée et compressée afin de pouvoir être stockée en mémoire : le nombre d'angles de vues de référence est fixé, ainsi que l'ensemble des valeurs de courbure. La taille d'une VDM de résolution de 128x128 pixels, 32x8 directions d'observation et 16 valeurs de courbure est de 64 Mo. La compression utilisée (une décomposition en vecteur propres des données) permet de réduire la taille de la structure à 4Mo. La décompression se fait au vol, lors du rendu.

Finalement cette technique permet de rendre tous les effets qui caractérisent la technique de déplacement de surface (figure 1.10 (b) page 26) c'est à dire les occlusions dans la texture dues au relief ainsi que la silhouette du modèle. De plus, l'auto-ombrage est géré en extrayant l'information d'ombrage de la VDM à partir de la position de la lumière. Les performances de la méthode sont impressionnantes :

[22]Une bonne description de l'algorithme, avec expérimentations peut être trouvée à l'adresse suivante : http ://www.cs.unc.edu/~nico/courses/comp236/vdm/

plus de 100 images/s en utilisant les dernières cartes graphiques. En revanche, l'espace mémoire nécessaire pour stocker l'ensemble des textures représentant la VDM restreint son utilisation a des motifs de faible résolution.

Déplacement de surface par pixel Hirche *et. al* [HEGD04] proposent également une technique permettant d'effectuer le rendu d'une surface déplacée à l'aide de *pixels shaders*.

La couleur de chaque pixel affiché correspond à l'intersection du rayon issu du centre de projection de la caméra avec la surface déplacée. Pour trouver cette intersection (figure 1.12 page 28), l'affichage d'un prisme contenant la surface déplacée de chaque triangle de la surface de base est effectué. Le prisme est créé en extrudant chaque triangle de la surface de base le long des normales à ses sommets. La face du prisme orientée vers l'observateur détermine le point d'entrée du rayon dans le prisme. La méthode consiste alors à échantillonner la partie du rayon à l'intérieur du prisme afin de trouver l'endroit ou il intersecte la carte de hauteurs. A cause des limitations dans la taille des *shaders*, quatre échantillons seulement sont répartis régulièrement le long du rayon. Le calcul de ces points (0,1,2 et 3 sur la figure 1.12 page 28) est effectué dans l'espace des coordonnées de texture et permet de comparer la hauteur du rayon à la hauteur stockée dans la carte. Cette méthode "telle qu'elle" ne fonctionne pas si la surface sur laquelle est plaquée la carte de déplacement est courbée, c'est à dire que le prisme peut être trop déformé (ne plus avoir ses côtés perpendiculaires au plan du triangle) et les interpolations linéaires faites sur les coordonnées de texture deviennent alors fausses. Pour traiter ce problème, le même principe est appliqué mais en se basant sur une décomposition du prisme en tétraèdres en fonction de la direction d'observation. Ceci nécessite un calcul par le CPU mais permet de traiter le cas général.

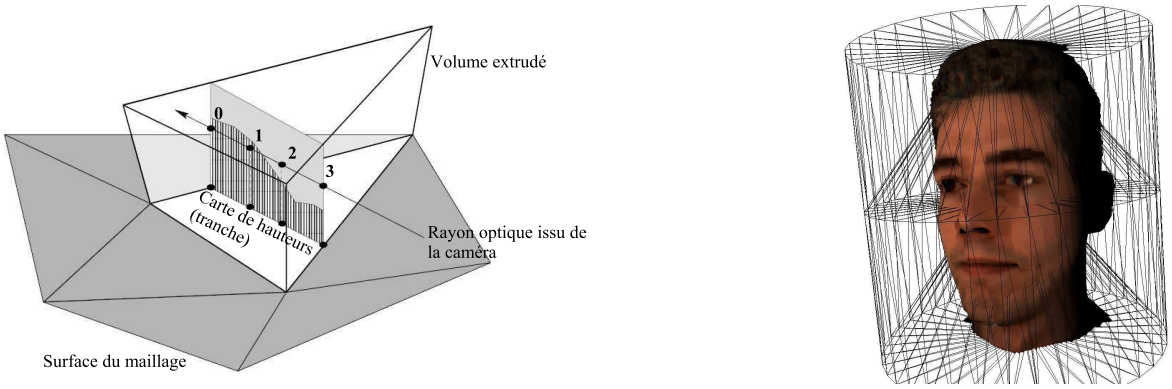
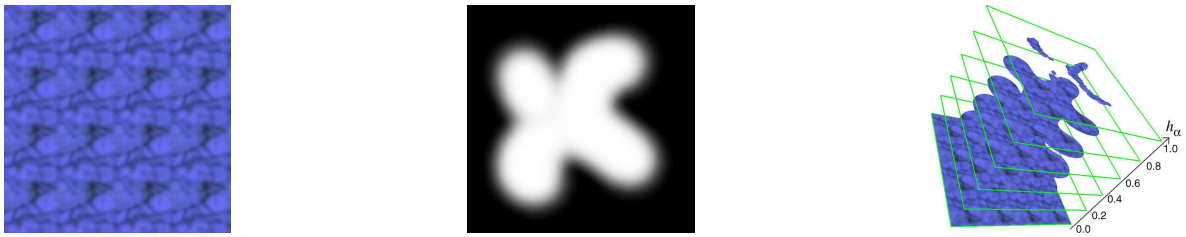


FIG. 1.12 – Affichage par pixel de surface déplacée ([HEGD04]).

Comparé à la méthode précédente, cette approche vise à échantillonner directement, au vol, le rayon issu du pixel à dessiner, afin de trouver son intersection avec la carte de hauteurs au lieu de la précalculer pour l'ensemble des directions d'observations. Ainsi, cette approche ne nécessite aucun espace de stockage supplémentaire par rapport à du déplacement de surface conventionnel. Néanmoins, le nombre restreint d'échantillons par rayon (fixé par la taille maximum d'un shader) limite la qualité du rendu. De plus, les performances, bien qu'interactives, ne suffisent pas dans le contexte d'une application classique d'environnement virtuel : 20 FPS pour une résolution de 500x500 pixels.

Déplacement de surface par superposition d'images Dans [KS01], Kautz implémente le déplacement de surface en affichant et combinant une série d'images issues de la découpe du *volume de déplacement*[23] (figure 1.13).



(a) Texture

(b) Champ de hauteurs

(c) Découpe du volume de déplacement

FIG. 1.13 – Déplacement de surface basé image ([KS01])

Le rendu s'effectue en dessinant toutes les tranches, leurs pixels n'étants affichés que si la hauteur dans la carte de déplacement est supérieure à la hauteur de la tranche à cet endroit. Plusieurs orientations des tranches sont créées afin de s'adapter au point de vue (tranches orthogonales au volume de déplacement, ou alignées par rapport à la direction de vision). La composante alpha des couleurs est utilisée pour stocker la hauteur et le *stencil buffer*[24] pour superposer les couches.

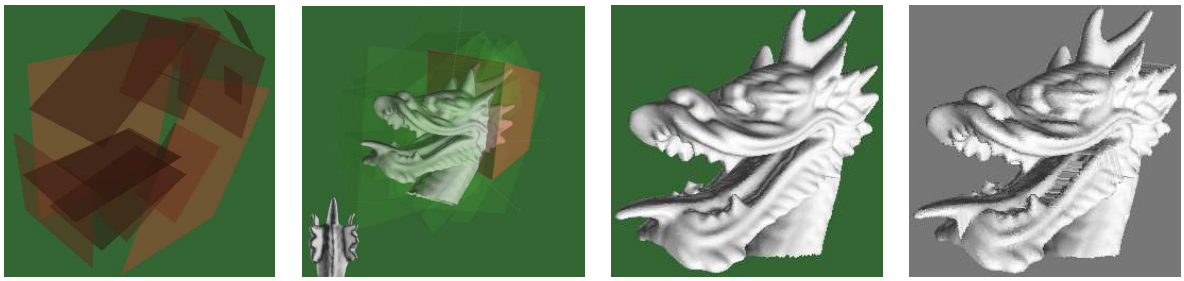
La méthode fonctionne en temps réel (entre 20 et 30 FPS sur une GeForce2) mais génère beaucoup de géométrie : 64 tranches pour une orientation de volume de déplacement (et il peut y avoir jusqu'à 6 orientations superposées) afin d'avoir une bonne qualité visuelle. De plus, cette méthode est très gourmande en fillrate, en espace mémoire et génère beaucoup de recouvrement (pixels dessinés plusieurs fois).

Déplacement de surface par déformation d'images Dans [SP99], Schaufler effectue l'affichage d'objets complexes en déformant plusieurs images de profondeurs issues du modèle avec, l'algorithme de déformation 3D de McMillan (section 1.4.3.2).

La figure 1.14 (a), montre les plans de projection des images de profondeurs de références utilisées pour afficher le modèle. Le figure (b) montre le résultat de l'affichage de ces images déformées, avec en bas à gauche, l'image contenue dans le volume de vision marron. Le problème principal est le positionnement de ces plans car ils doivent recouvrir le modèle de façon à ce qu'isoléments ils capturent uniquement un champ de hauteurs, c'est à dire qu'un rayon optique issu du centre de projection d'un volume de vision ne doit pas intersecter plus d'une fois la surface du modèle.

Les performances sont bonnes : pour le dragon, avec une résolution de 512x512 pixels pour les images de référence, le temps de rendu est de 2 secondes sur un processeur cadencé à 250 MHz, c'est à dire très ancien. Néanmoins, tout l'algorithme est effectué par le processeur central ce qui est très couteux. De plus, le positionnement des plans de référence doit être fait à la main et conditionne la qualité du

[23] Le *volume de déplacement* est le volume englobant le relief produit par la carte de hauteur sur la surface de base.
[24] Voir les annexe A.



(a) Plans des images de référence

(b) Reconstruction par déformation 3D

(c) Le modèle original

(d) Le modèle reconstruit

FIG. 1.14 – Déplacement de surface selon Schaufler ([SP99])

résultat. Comme le montre la figure 1.14 (d), des effets de "ruban de caoutchouc" (*rubber-sheet*) sont inévitables.

Placage de texture en relief La technique de placage de texture en relief a été présentée par Oliveira *et al.* [OBM00]. Elle permet d'effectuer du displacement mapping en utilisant le placage de texture conventionnel sur un modèle simplifié de l'objet a afficher. Pour cela, la carte d'élévation est tout d'abord utilisée afin de déformer l'image en fonction de la direction d'observation. Cette image déformée est ensuite plaquée de façon conventionnelle sur le modèle simplifié. La déformation est basée sur les équations de déformation 3D *(3D Warping)* de McMillan (cf. section 1.4.3.2) modifiées de façon à se qu'elles puissent s'appliquer à une carte d'élévation plutôt qu'une carte de profondeurs. Finalement, l'opération de placage de texture en relief s'opère en deux phases (cf. figure 1.15). Durant

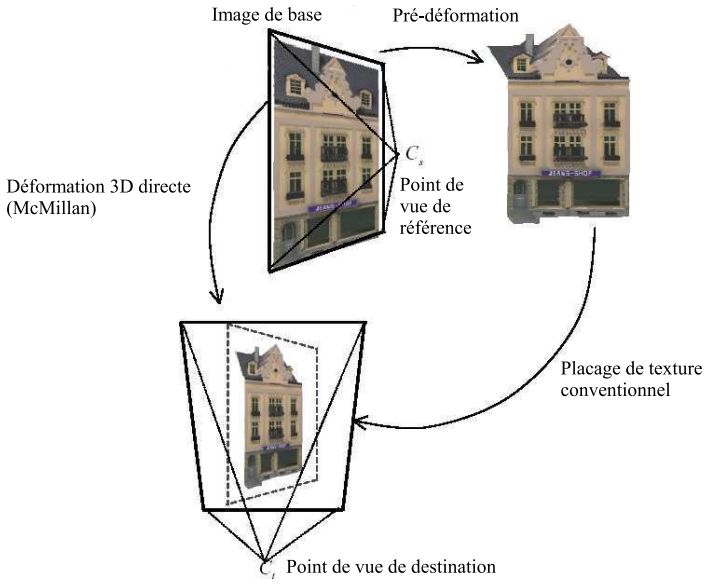


FIG. 1.15 – Placage de texture en relief ([OBM00])

la première phase de pré-déformation (*pre-warping*), la texture est déformée selon les équations modifiées de McMillan. Cette phase est effectuée par le CPU et malgré qu'une méthode permettant de tirer parti des cartes graphiques ait été présentée [MF02], correspond à la partie la plus lente de l'algorithme. La seconde passe de consistant qu'à plaquer la texture est très rapide. La méthode peut être appliquée à une boîte englobant la totalité de la scène.

1.4.1.5 *Parallax Mapping*

Dans [TKT01], Kaneko *et al.* introduisent le *parallax mapping*. Le principe est de changer les coordonnées de textures de chaque pixel affiché, en fonction de la hauteur stockée dans la carte de hauteurs et de l'angle d'observation, de façon à refléter les effets de parallaxe produits par le relief plaqué. On obtient ainsi le mouvement apparent du relief quand le point de vue change. Leur technique est très rapide (environ 80% de la vitesse du placage de texture conventionnel) et probante (figure 1.17 (b) page 32) mais fausse, comme le montre la figure 1.16, car ils approximent la hauteur du relief en supposant qu'elle varie selon une courbe basée sur la hauteur du point de référence (A sur la figure).

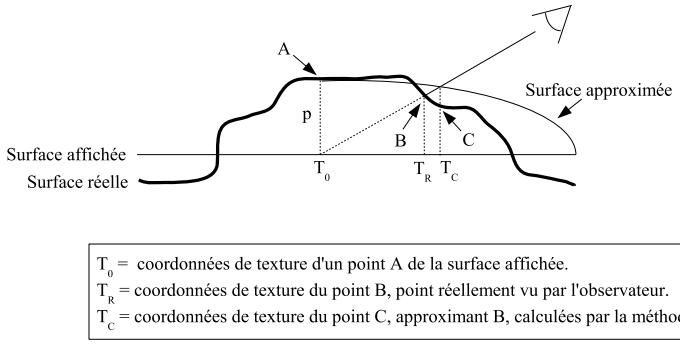


FIG. 1.16 – *Parallax Mapping*

Néanmoins, pour une applicaton ludique par exemple, l'erreur commise peut être ignorée car l'effet de mouvement du relief est saisissante. La rapidité de la méthode (un simple changement de coordonnées de texture implémenté à l'aide des fonctionnalités détournées du placage d'environnement des cartes graphiques) a assuré son succés.

Dans [Wel04], Welsh présente une amélioration de cette technique qui permet de prendre en compte, dans une certaine mesure, les effets d'occlusion. Le principe est de limiter le déplacement de la coordonnée de texture par une valeur arbitraire, basée sur ce que représente le relief. Par exemple, si le relief est un mur de brique, cette valeur sera basée sur la profondeur des interstices entre les briques, car ce sont eux qui génèrent de l'occlusion. Les résultats sont probants (figure 1.17 (c) page 32), mais très liés à la nature du relief (qui doit être le plus doux possible). Cette technique ne marche évidemment pas pour un relief quelconque, est géométriquement fausse, et nécessite de nombreux tests avant de trouver les valeurs donnant des résultats visuels probants, mais elle est rapide et simple à programmer, en particulier à l'aide des shaders.

(a) Placage de texture simple

(b) *Parallax Mapping*

(c) *Parallax mapping* avec décalage borné

FIG. 1.17 – *Parallax mapping* avec et sans décalage borné

1.4.1.6 Placage d'environnement

Bien que le placage d'environnement ne soit pas à proprement parler une technique de rendu en temps réel de scène complexe, il s'agit néanmoins d'une technique basée image permettant de faciliter la simulation d'éclairement complexe. Le placage d'environnement (*Environment Mapping)* a été introduit par Blinn [BN76]. Une carte de réflexion (une texture, cf. figure 1.18 page 32) représentant la lumière arrivant de toutes les directions en un point est utilisée pour modéliser les réflexions spéculaires entre les objets d'une scène. La carte de réflexion est associée à une sphère, un cylindre ou un

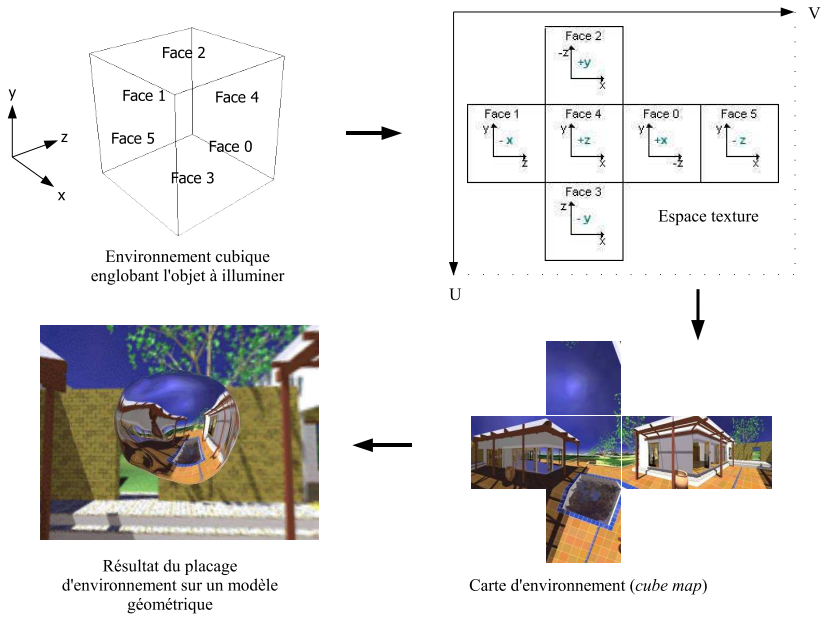


FIG. 1.18 – Placage d'environnement cubique

cube englobant centré sur l'objet dont on souhaite simuler l'éclairement. Par rapport au placage de texture standard, l'application de l'environnement à l'objet se fait non plus en utilisant la paramétrisation de l'objet (coordonnées de texture (u, v)), mais en utilisant la normale à la surface (cartes de réflexion

diffuse [MH84]), ou le rayon réfléchi issu de la caméra (cartes de réflexions spéculaires [BN76]). De nombreux travaux récents emploient le placage d'environnement pour permettre l'intégration d'objets virtuels dans un environnement réel : les objets virtuels sont éclairés par la lumière réelle de la scène capturée dans des photographies. Finalement, l'environnement mapping est une fonctionnalité intégrée dans toutes les cartes graphiques, étant donné qu'il s'agit d'un placage de texture. Le travail effectué par la carte graphique est simplement de calculer les coordonnées texture par sommet.

1.4.1.7 Placage de texture dépendant du point de vue

Le placage de texture dépendant du point de vue (*"View Dependent Texture Mapping"*) est une technique qui permet de modifier la couleur des pixels d'une texture de façon à ce qu'elle s'adapte à la position de la caméra. Le principe général est de pré-acquérir un certain nombre d'images de référence de la surface d'un modèle géométrique complexe selon différents angles d'observation, puis de plaquer et de mélanger ces images sur une version simplifiée du modèle de façon à obtenir un ensemble continu de textures intermédiaires de la surface, quelque soit la direction d'observation.

L'ensemble des techniques que nous décrivons fusionnent ces images en fonction de la proximité du point de vue virtuel avec les points de vue de référence. La fusion en elle même est très simple, il s'agit d'une combinaison linéaire des couleurs des texels des images de référence effectuée comme suit (figure 1.19 (a) page 34) :

Soient :

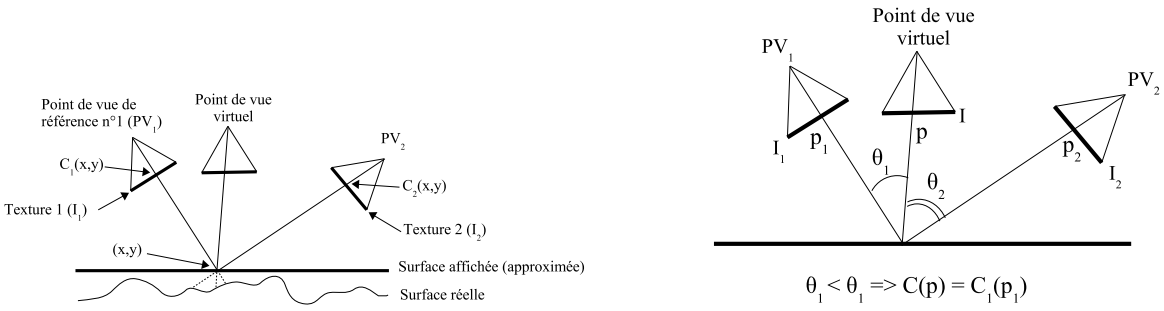
$C(x, y)$	la couleur (R, V, B) du texel de coordonnées (x, y) dans la texture à obtenir.
$C_i(x, y)$	la couleur du texel de même coordonnées dans la texture de référence n°i.
$w_i(x, y)$	un coefficient de pondération inversement proportionnel à la distance entre le texel (x, y) du point de vue virtuel et le point de vue de référence i.
n	le nombre de points de vues choisis.

On a :

$$C(x,y) = \frac{\sum_{i=1}^{n} w_i(x,y).C_i(x,y)}{\sum_{i=1}^{n} w_i(x,y)}$$

Debevec *et al.* Debevec *et al.* ont été les premiers à utiliser de telles textures dynamiques. Dans [DTM96], la texture à plaquer sur le modèle simplifié est construite comme suit. Chaque pixel p de la texture à générer reçoit la couleur $C_i(p_i)$ du pixel p_i de l'image de référence n°i considérée comme la mieux adaptée au rendu de ce pixel. Comme le montre la figure 1.19 (b) page 34, l'image la mieux adaptée au pixel p de la texture I à construire est l'image I_1 car l'angle θ_1 formé par le rayon issu du centre optique du point de vue 1 avec le rayon issu du centre optique de point de vue virtuel est inférieur à l'angle θ_2 formé avec le deuxième point de vue. Le pixel p de l'image I a afficher est donc coloré avec la couleur de p_1.

Etant donné qu'avec cette technique des pixels voisins de p peuvent être colorés avec des pixels issus de différents points de vue, des discontinuités apparaissent, dégradant l'image. Pour traiter ce

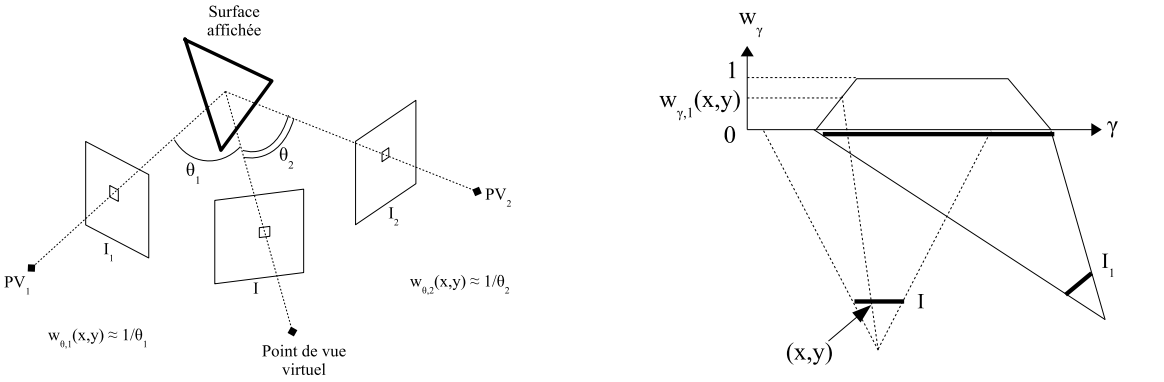


(a) Principe général

(b) Selon Debevec

FIG. 1.19 – Mélange d'images

problème, le pixel p n'est plus coloré avec un pixel issu d'un seul point de vue, mais en mélangeant tous les point de vue avec une fonction de pondération $w_{\theta,i}$ évaluée pour chaque pixel p et chaque point de vue de référence. Cette fonction est inversement proportionnelle à l'angle θ_i formé par les rayons issus des centres optiques de chaque point de vue de référence et du point de vue virtuel (cf. figure 1.20(a)).



(a) Pondération $w_{\theta,i}$

(b) Pondération $w_{\gamma,i}$

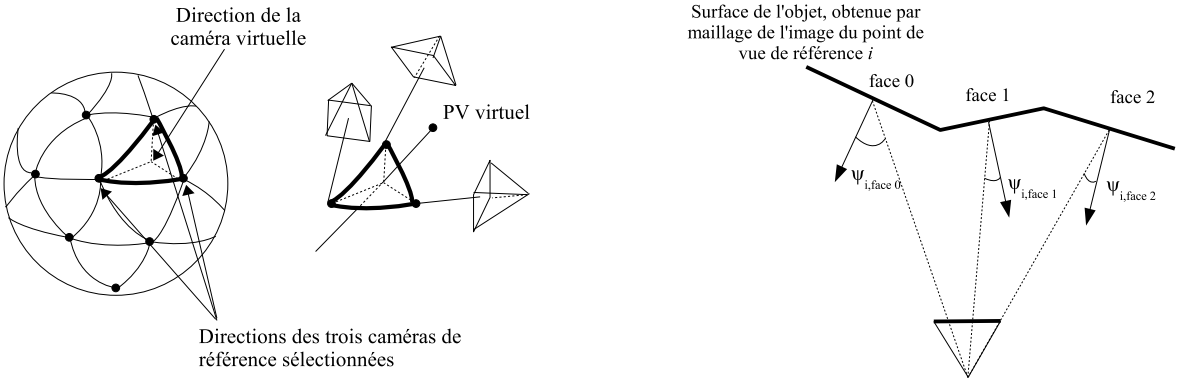
FIG. 1.20 – Pondérations utilisées dans [DTM96]

Une autre discontinuité apparaît également à la frontière entre une région texturée à partir d'une seule image de référence et une région texturée à partir de plusieurs images de référence. Une seconde fonction de pondération, $w_{\gamma,i}$ indépendante du point de vue, vient se combiner à la première. Cette pondération tend vers zéro sur les bordures de chaque image de référence et permet ainsi d'adoucir les transitions (figure 1.20(b)).

Bien entendu, ces calculs effectués par pixel sont lourds en ne peuvent s'effectuer en temps réel. Dans [DYB98], les pondérations sont rammenées à l'échelle de la face entière sans baisse notable de la qualité visuelle : lors du rendu pour chaque face sont déterminés les points de vue à fusionner

(les points de vue où cette face est visible) et la pondération $w_{\theta,i}$ n'est calculée qu'une seule fois, au barycentre de la face. Ainsi, le mélange s'effectue en temps réel. Néanmoins les problèmes principaux restent : les effets de flou, et de *ghosting*[25] bien qu'atténués, ne font illusion que si la surface sur laquelle est utilisée cette méthode est relativement éloignée de l'observateur. Cette méthode a été utilisée dans le film Matrix (©1999-2004 Warner Bros.) pour créer le décor de fond de certaines scènes impossibles à filmer "en réel". Etant donné que le mélange des couleurs ne tient pas compte de l'éclairement, les photographies de référence du décor devaient être prises durant un temps nuageux de façon à ce qu'aucun reflets spéculaires ne soit présents sur les photos.

Pulli *et al.* Dans [PCD+97], le placage de texture dépendant du point de vue est utilisé pour effectuer le rendu en temps réel de maillages issus de la triangulation d'images se superposant partiellement, obtenues à l'aide de scanners optiques. Le schéma de pondération est légèrement différent de celui utilisé par Debevec. Ils utilisent trois pondérations différentes, soit : $w_i = w_{\theta,i}.w_{\gamma,i}.w_{\psi,i}$.



(a) Pondération basée sur les coordonneés barycentriques du point de vue virtuel dans le triangle la sphère intersectant la direction de la caméra

(b) Pondération basée l'orientation des faces du maillage par rapport au point de vue de référence

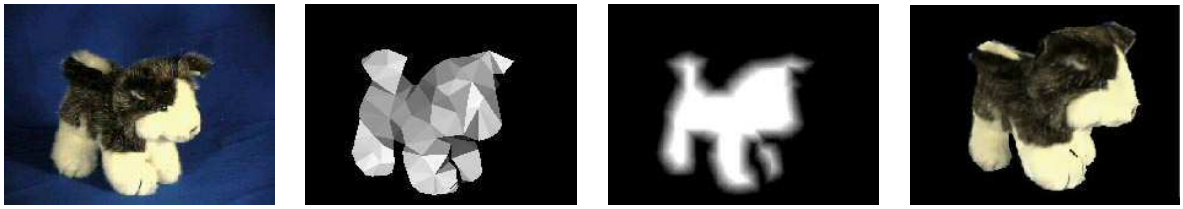
FIG. 1.21 – Schémas de pondération de Pulli *et al.*

$w_{\theta,i}$ est une pondération dépendante du point de vue basée sur trois points de vue, les trois plus proches du point de vue virtuel. La pondération ne s'effectue pas par pixel, mais est évaluée une seule fois et appliquée à tous les pixels. Etant donné que cette méthode ne fonctionne bien que si les points de vue de référence sont positionnés sur un plan autour de l'objet (l'angle θ_i s'évalue facilement) et ne peut pas s'appliquer à des points de vue répartis à des hauteurs différentes, Pulli *et al.* répartissent les points de vue de référence sur une sphère unité centrée sur la scène et la maillent avec une triangulation de Delaunay (figure 1.21(a)). Le point de vue virtuel est alors positionné sur cette sphère et les coordonnées barycentriques de son axe optique intersectant une maille déterminent alors directement $w_{\theta,1}$, $w_{\theta,2}$ et $w_{\theta,3}$.

$w_{\gamma,i}$ est une pondération indépendante du point de vue qui, comme chez Debevec, permet d'atténuer les discontinuités aux frontières du maillage (figure 1.22(c)).

[25] Apparition dans l'image de morceaux de la scène à moitié mélangés et à des positions erronées.

$w_{\psi,i}$ est également indépendante du point de vue et mesure l'adéquation d'une image de référence à une face. Les meilleurs points de vue de référence étant ceux qui ont leur orientation la plus proche de la normale à la face. C'est à dire les images pour lesquelles les déformations de perspective de la texture sont les plus faibles. (figure 1.21(b) et 1.22(b)).



(a) Image de référence (b) Pondération ψ_i (c) Pondération γ_i (d) Image reconstruite

FIG. 1.22 – *View-Based Rendering.*

Pighin *et al.* Dans [PHL+98], le placage de texture dépendant du point de vue est basé sur le mélange de seulement deux points de vue de référence appartenant à un même plan. Il s'agit de photographies d'un visage effectué par un système d'acquisition tournant autour du sujet. Le shéma de pondération est basé sur trois termes : $w_i = F^i.P^i.V^i$.

F^i est une carte de visibilité calculée pour chaque texture I_i, indépendante du point de vue. Ce terme est un masque qui indique les zones de l'image de référence qui appartiennent au modèle (valeur=1) des zones extérieures (le fond, valeur=0). Un léger dégradé est calculé pour adoucir la transition à la frontière.

P^i correspond à la pondération $w_{\gamma,1}$ de Pulli.

V^i correspond à la pondération $w_{\theta,i}$ de Pulli, modifiée de façon à ce qu'une seule texture soit sélectionnée quand le point de vue de référence et le point de vue virtuel correspondent.

1.4.2 Imposteurs

Toujours dans le but d'obtenir une grande complexité visuelle au moindre coût, il existe une classe de techniques regroupées sous le terme d'*imposteurs*. Toutes ces méthodes effectuent un pré-rendu hors-écran (*off-screen*) d'une partie de la scène. L'image créée est utilisée par la suite en temps que texture plaquée sur un rectangle positionné au sein de la scène. Ainsi, à la place de l'objet complexe est affiché son image : un imposteur [MS95].

1.4.2.1 Billboards et Sprites

Les plus simples des imposteurs sont les *billboards* (litt. "Panneau d'affichage") et les sprites (litt. "lutin"). Un *billboard* est un plan texturé placé dans la scène, représentant un objet complexe. Typiquement, les billboards sont employés pour représenter des objets tels que des nuages ou des arbres, très difficiles à modéliser explicitement. On distingue les billboards toujours parallèle à l'écran, ceux

toujours orientés vers l'observateur et ceux qui sont de simples plans fixes dans l'espace de la scène. Chacun de ces types de *billboards* est adapté au rendu de certains types d'entités. Ceux toujours parallèles à l'écran ou orientés vers l'observateur servent par exemple pour représenter des nuages ou de la fumée. Un arbre, par contre, est souvent approximé par deux billboards perpendiculaires positionnés dans la scène.

Les sprites se différencient du *billboard* par l'utilisation qui en est faite et leur mise en oeuvre : un sprite est généralement plus petit, en mouvement dans la scène et appliqué directement à l'écran. En résumé, un *billboard* est un plan 3D tandis qu'un sprite est un rectangle 2D toujours parallèle à l'écran quelle que soit l'orientation de la caméra.

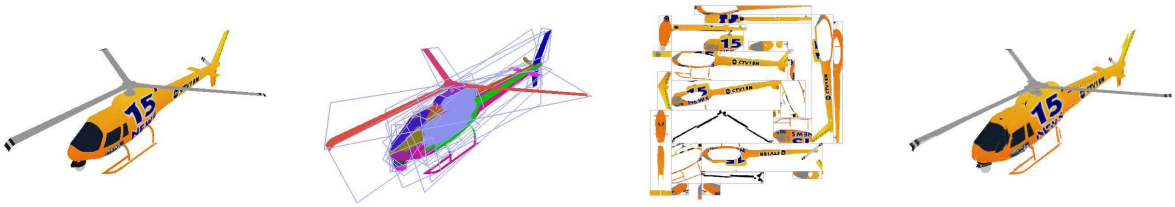
Dans [HAA97], Horry *et al.* utilisent des billboards extraits de photographies dans le cadre de leur méthode nommée "Excursion dans une image" (*Tour Into the Picture)*. Le principe est de déformer une image (voir la section 1.4.4 traitant du *morphing*) en fonction de son point de fuite. Le point de fuite sert de repère pour la création (par l'utilisateur) d'un maillage recouvrant la scène, et la séparant en différentes zones (arrière plan, sol, bords, ...) (figure 1.23 page 37). Les billboards sont utilisés pour modéliser les objets aux premiers plan de l'image et sont déplacés lors de l'animation afin d'accompagner le mouvement de pénétration dans l'image. Leur technique, très artistique, a été utilisée dans de nombreux vidéoclips : le déplacement des billboards, combiné à une légère déformation de l'image donne l'impression de pénétrer dans l'image.

(a) Image de référence (b) Maillage de l'image (c) Image obtenue

FIG. 1.23 – Horry *et al.* [HAA97]

Plus récemment, Décoret *et al.* [DDSD03] ont utilisé le principe des billboards pour effectuer l'affichage d'objets géométriques complexes. Le principe de leur méthode est de remplacer l'objet par la combinaison de plusieurs billboards représentant l'objet sous différents angles. Leur méthode permet de répartir automatiquement un certain nombre (fixé par l'utilisateur) de plans recouvrant l'objet de façon optimale. Cet ensemble de billboards (polygones à texture transparente) est simplement affiché lors du rendu, les plans se superposant et se combinant à l'aide de la composante de transparence des textures (figure 1.24 page 38).

Le modèle géométrique ainsi remplacé est approximé visuellement, ainsi la méthode est elle particulièrement adaptée à la simplification "grossière" d'objets. De plus, un grand nombre de textures sont nécessaires pour obtenir une qualité visuelle acceptable, et l'espace mémoire de la carte graphique



(a) Modèle original (b) Billboards l'approximant (c) Textures (d) Modèle reconstruit

FIG. 1.24 – Décoret *et al.* [DDSD03]

est une ressource critique. Néanmoins, l'affichage est extrêmement rapide car il se fait à l'aide du simple placage de texture. De plus, l'éclairement peut varier (utilisation d'une carte de normales) et la projection d'ombre est également possible.

Dans [YSK+02] Yamazaki *et al.* utilisent des billboards, comme Décoret *et al.*, pour effectuer le rendu en temps réel d'un objet très complexe, obtenu à l'aide d'un dispositif d'extraction de forme à partir de photographies effectuées autour du modèle (réel). L'objet, une peluche, est très difficile à modéliser explicitement, à cause de la fourrure. Pour chaque photographie, la coque englobante de l'objet, photographiée sur un "fond bleu", est extraite et sert à récupérer un version approchée de sa géométrie (figure 1.25(a)).

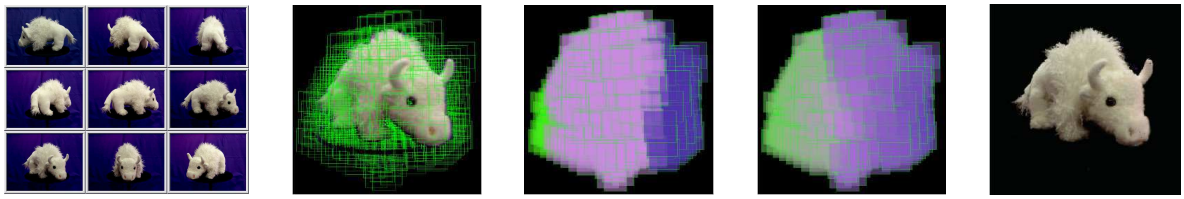
Ces photographies servent de texture pour le nuage de billboards qui sont positionnés à l'aide la géométrie approximée (figure 1.25(b)). Ainsi, une texture permet de générer tout un ensemble de *billboards*, au lieu d'un seul, d'où le terme de billboarding en microfacettes (*microfacet billboarding*). Chaque microfacette est toujours affichée perpendiculairement à la direction d'observation et est texturée avec du placage de texture dépendant du point de vue, de façon à mélanger les meilleures images de référence de l'objet (figures 1.25 (c) et (d)).

Le rendu final est très rapide. De plus, le modèle de base peut être un modèle réel, ce qui permet un grand réalisme. Le problème principal de cette méthode est que l'éclairement ne peut varier, du fait de l'emploi de placage de texture dépendant du point de vue.

1.4.2.2 Imposteurs statiques

Maciel et Shirley [MS95] ont introduit les imposteurs pour la visualisation interactive dans un environnement virtuel extrêmement complexe. Ils définissent un imposteur comme une entité qui est plus rapide à dessiner que l'objet réel, tout en gardant ses principales propriétés visuelles. Ainsi, ils remplacent des parties complexes de la scène par un modèle simplifié texturé englobant permettant d'obtenir un taux d'affichage élevé et approximativement constant. En utilisant en plus la technique des niveaux de détails (section 1.2), ces modèles simplifiés texturés sont appelés *Imposteurs*.

Dans leurs travaux, la scène complète est modélisée à l'aide d'un maillage géométrique conventionnel et les textures des imposteurs sont stockées dans une structure hiérarchique précalculée. Lors du rendu, la hiérarchie est traversée de haut en bas afin de trouver le meilleur compromis visuel pour le point



(a) Images de référence

(b) Microfacettes

(c) Choix des images de référence par microfacettes

(d) Interpolation dépendante du point de vue

(e) Image reconstruite

FIG. 1.25 – Yamazaki *et al.* [YSK+02]

de vue : au pire, quand il y a trop de modèles à afficher en temps réel, une simple boîte englobante texturée est dessinée. Ainsi, le problème principal de cette méthode est de savoir quel imposteur dessiner pour maximiser la qualité visuelle de l'image sans dépasser le temps d'affichage maximum fixé par l'utilisateur.

1.4.2.3 Imposteurs dynamiques

A la différence des imposteurs statiques, tous pré-générés, Shaufler [Sch95] a introduit les imposteurs dynamiques. Dans cette méthode, un imposteur est toujours représenté par une image de l'objet plaquée sur un polygone transparent (cf. figure 1.26 page 39). Ce qui a motivé ce travail est le fait que l'approche avec des imposteurs précalculés devient rapidement inutilisable dès que le nombre de d'objets augmente car l'espace mémoire nécessaire pour stocker les textures est limité. De plus, la définition des textures ne peut pas être déterminée précisément à l'avance ce qui oblige à toujours prévoir le pire des cas : un imposteur vu de près, nécessitant une haute définition, donc beaucoup de mémoire, alors qu'il est possible que cet objet apparaisse toujours au dernier plan de la scène. Pour finir, l'ensemble discrétisé des directions de vue pour lesquelles les textures de l'imposteur statique sont générées limite les points de vue pour lesquels l'imposteur est visuellement valide.

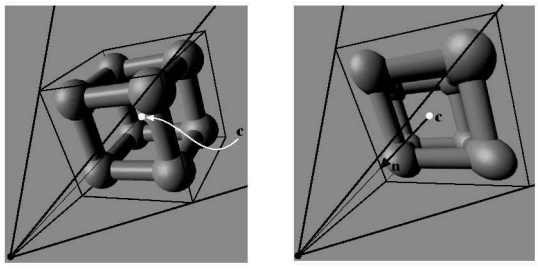


FIG. 1.26 – Imposteur. A gauche, l'objet original, à droite, l'imposteur.

Schaufler propose donc de générer un imposteur par objet, et ce, lors du rendu. La scène et ses imposteurs est stockée dans une structure hiérarchique appelée "*three dimensional image cache*". Durant le

rendu, les imposteurs sont utilisés pour remplacer les objets complexes. Pour déterminer la validité visuelle d'un imposteur, l'angle maximal en dessous duquel tous les points de la surface de l'objet sont aussi dans l'image est évalué. Tant que l'angle de vue de l'imposteur est inférieur à ce seuil, l'imposteur est considéré valide. Si le point de vue n'est pas translaté mais que seule la direction de vue change, l'imposteur reste valide. Ainsi les imposteurs peuvent être réutilisés d'une image à l'autre lors de l'animation, et ne sont régénérés que si ils deviennent invalides, accélérant ainsi le rendu.

Shade [SLS+96] a proposé une méthode similaire à celle de Schaufler. Durant une phase de précalculs, un arbre BSP qui partitionne hiérarchiquement les primitives géométriques de la scène est construit. Lors du rendu, les images de la géométrie stockée dans chaque noeud sont mises en cache pour être réutilisées par les images suivantes de l'animation. Un telle image mise en cache est réutilisée en la plaquant sur un simple rectangle qui remplace la géométrie du noeud.

1.4.2.4 Nailboards et imposteurs en couche

Les précédentes méthodes traitent toutes les images comme des polygones texturés. Une valeur de profondeur est assignée à chaque imposteur afin de déterminer ceux destinés à être en avant plan, des autres. Ce système en couche ne résoud pas les problèmes de visibilité en général étant donné qu'un imposteur ne peut être que soit devant, soit derrière un autre. L'utilisation de polygones transparents texturés provoque également des problèmes de visibilité car les polygones peuvent s'intersecter ou intersecter la géométrie de la scène. De plus, ces images planes ne prennent pas en compte les effets de parallaxe et d'occlusion quand elles sont vues d'un point de vue différent du point de vue de capture.

Nailboards Pour résoudre ces problèmes, Schaufler a proposé une nouvelle primitive graphique nommée *nailboard* [Sch97], qui étend les imposteurs dynamiques. Un nailboard est un polygone sur lequel est plaqué une texture de type $RVB\Delta$, représentant un objet complexe. Pour chaque texel, la composante Δ représente la distance entre le polygone de l'imposteur et l'objet réel. Quand cette texture est générée à partir de l'objet réel, la valeur de Δ est récupérée directement à partir du tampon de profondeur. Ensuite, lors du rendu, cette valeur est utilisée pour changer la profondeur du polygone de l'imposteur. Schaufler décrit les transformations nécessaires pour pouvoir intégrer ce *nailboard* au tampon de profondeur standard et ainsi obtenir une visibilité correcte. Avec cette méthode, un *nailboard* peut être mélangé au rendu conventionnel.

Imposteurs en couches Une autre extension des imposteurs dynamiques à été proposée par Schaufler dans [Sch98], appelée *imposteurs en couche (layered impostors).* Un imposteur dynamique remplace un objet complexe par un polygone transparent sur lequel est plaquée une image de l'objet. Un imposteur en couches est composé de plusieurs de ces polygones. Tous les pixels d'une couche sont à une distance identique de l'observateur, c'est à dire que les pixels d'une image sont distribués sur plusieurs calques en fonction de leur profondeur (cf. image 1.27 page 41). Comme les imposteurs en couche contiennent une profondeur (approximative), il est possible de résoudre correctement les problèmes de visibilité lors des changement de point de vue. Une mesure d'erreur visuelle identique aux imposteurs standard est utilisée pour savoir quand régénérer l'imposteur. Avec cette technique, un imposteur a une durée de vie supérieure aux imposteurs monocouches.

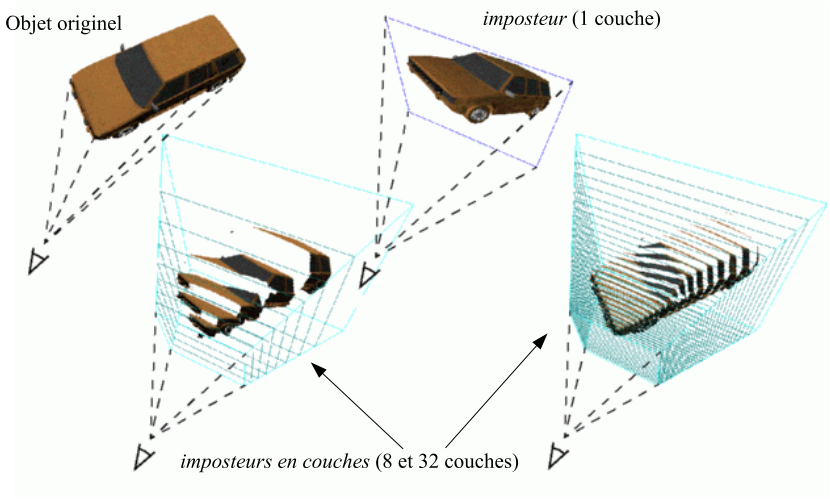


FIG. 1.27 – *Imposteurs en couches* ([Sch98])

1.4.2.5 Imposteurs maillés

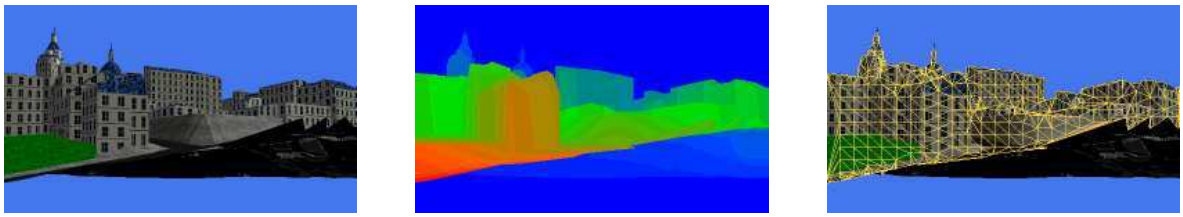
Jusqu'ici, les méthodes à base d'imposteurs plaquent une image sur un plan (quadrilatère ou polygone). Néanmoins, la durée de vie (le nombre d'images de l'animation pour lesquelles l'imposteur est valide) de ces imposteurs est courte à cause des effets de parallaxe. Afin d'augmenter ce temps de vie, un certain nombre d'algorithmes utilisent une géométrie plus complexe. Ils n'utilisent plus seulement une image pour remplacer la géométrie distante de l'observateur mais un maillage de triangles. Cette classe d'imposteurs est appelée *imposteurs maillés (meshed impostors)*. Le principe est de trianguler la scène en fonction de zones approximativement planes, durant une phase de précalculs. Lors du rendu, l'affichage du maillage sur lequel est plaqué l'image capturée est effectué.

Darsa *et al.* [DCV97] proposent une technique de rendu basée image pour la naviguation dans un environnement 3D virtuel complexe. Un ensemble de cartes d'environnement cubiques sont pré-affichées à partir d'un certain nombre de points de vues fixés dans l'environnement. Ensuite, un maillage polygonal est créé par triangulation des six cartes de profondeurs ainsi obtenues. Lors du rendu, une vue de la scène pour un point de vue arbitraire est créée en plaquant la texture d'environnement sur les maillages créés. Plusieurs cartes d'environnement avec leur maillage de triangles sont mélangés pour remplir les trous et éliminer les artéfacts[26] visuel (déformation, étirement des triangles sous des point de vues différents des points de vue de capture[27]) en combinant les images de profondeurs issues de la structure à l'aide d' α-*blending*, du *stencil buffer* ou de l'*accumulation buffer*[28].

Sillion *et al.* [SDB97] utilisent ce principe pour la visualisation en temps réel d'environnement urbains. A chaque image de l'animation, la ville est découpée en deux parties : la partie distante (arrière plan) et la partie proche de la géométrie (premier plan). La géométrie au premier plan est affichée normalement, tandis que les parties distantes sont remplacées par des imposteurs maillés. Une image de la partie distante est générée une seule fois pour un point de vue donné et est stockée dans une texture (la texture de l'imposteur). Le contour externe de la scène distante est construit à partir de

[26] Des effets indésirables.
[27] Aussi dénommés *ruban de caoutchouc (rubber-sheet)* ou *écorchure (skinning).*
[28] Voir les annexes.

l'image de profondeur (issue du *z-buffer*) : le maillage de triangles associé à la texture constitue alors l'imposteur (cf. figure 1.28 page 42). Un des problèmes avec cette approche est que la séparation imposteur/géométrie réelle est visible lorsqu'on s'en rapproche.



(a) Texture de l'imposteur (b) Tampon de profondeur (c) Imposteur maillé

FIG. 1.28 – *Imposteurs maillés* de Sillion ([SDB97]).

Chen *et al.* [CSKK99] ont proposé une méthode similaire pour accélérer le rendu de terrain : un image est générée à partir de toute la géométrie du terrain à chaque position clé. Cette image est réutilisée par placage de texture sur une version simplifiée de la géométrie durant l'intervalle entre deux positions clés.

1.4.3 Déformation d'images

Dans cette partie, nous décrivons les techniques de déformation d'images basées sur des images de profondeurs. La problématique est la suivante : étant donnée une image obtenue à partir d'une caméra placée dans une scène, comment utiliser uniquement cette image pour construire un nouveau point de vue (une nouvelle image) pour une position de caméra différente. Ici, nous connaissons la profondeur des pixels, c'est à dire que nous disposons d'une représentation surfacique discrète de la scène et nous connaissons aussi les caractéristiques de la caméra (position, direction, matrice de transformation).

1.4.3.1 Reprojection de pixels

La façon directe permettant de transformer ces points dans l'espace d'une autre caméra est la *reprojection de pixels*. Le principe est d'effectuer deux changements de repère successifs pour chaque point de l'image de départ. Ainsi, cf. figure 1.29, p_1 dans l'image I_1 s'exprime dans l'espace de la scène par la transformation représentée par la matrice de transformation inverse du point de vue 1 (P^{-1}), et donne le point p exprimé dans l'espace de la scène, soit $p = P_1^{-1}.p_1$. Reste alors à projeter de façon conventionnelle le point p sur le plan de projection de la caméra 2, soit : $p_2 = P_2.p$. Les matrices de transformation (linéaires) se concatènent et l'on obtient finalement : $p_2 = P_2.P_1^{-1}.p_1$.

Cette technique nécessite l'emploi d'un tampon de profondeurs pour l'image 2 car plusieurs points de l'image de départ peuvent se projeter sur le même pixel de l'image de destination. De plus, les problèmes de visibilité et d'échantillonnage ne sont pas traités : l'image finale contient des trous (cf. image I_2). Cette technique a été utilisée dans [DCV97, Gro98].

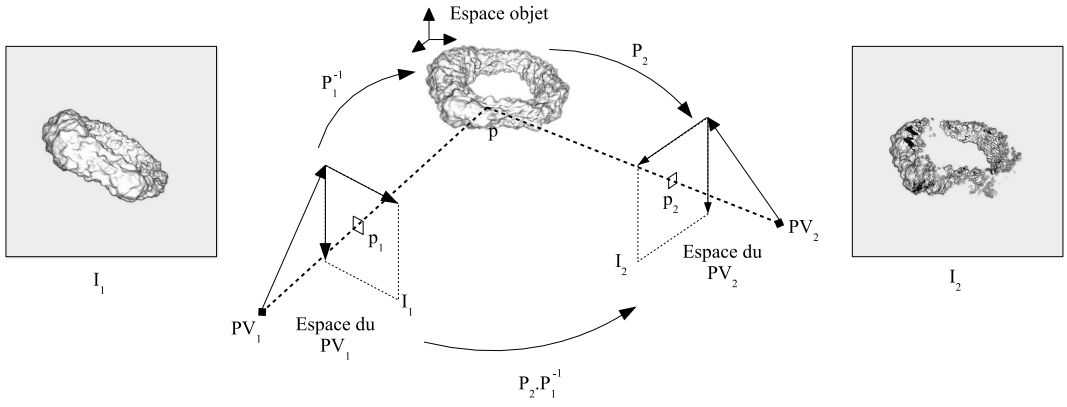


FIG. 1.29 – Principe de la rétroprojection de pixels

1.4.3.2 Déformation 3D d'images

Dans [McM97, McM95b], McMillan formalise les équations de reprojection. Il dérive de la profondeur classique la *profondeur projective (projective depth)* ou encore *disparité généralisée (generalized disparity)*, pour laquelle l'expression des équations de reprojection est plus simple. La profondeur projective d'un pixel correspond au rapport de la distance focale de la caméra par la profondeur du point (cf. figure 1.30 page 44). Ainsi, si l'on souhaite calculer le point $p_2(x_2, y_2)$ de I_2 à partir du point $p_1(x_1, y_1)$, de disparité généralisée $\delta(x_1, y_1)$, on a :

$$x_2 = \frac{\overrightarrow{a_1}.(\overrightarrow{b_2} \times \overrightarrow{c_2})x_1 + \overrightarrow{b_1}.(\overrightarrow{b_2} \times \overrightarrow{c_2})y_1 + \overrightarrow{c_1}.(\overrightarrow{b_2} \times \overrightarrow{c_2}) + \overrightarrow{C_2C_1}.(\overrightarrow{b_2} \times \overrightarrow{c_2})\delta(x_1, y_1)}{\overrightarrow{a_1}.(\overrightarrow{a_2} \times \overrightarrow{b_2})x_1 + \overrightarrow{b_1}.(\overrightarrow{a_2} \times \overrightarrow{b_2})y_1 + \overrightarrow{c_1}.(\overrightarrow{a_2} \times \overrightarrow{b_2}) + \overrightarrow{C_2C_1}.(\overrightarrow{a_2} \times \overrightarrow{b_2})\delta(x_1, y_1)}$$

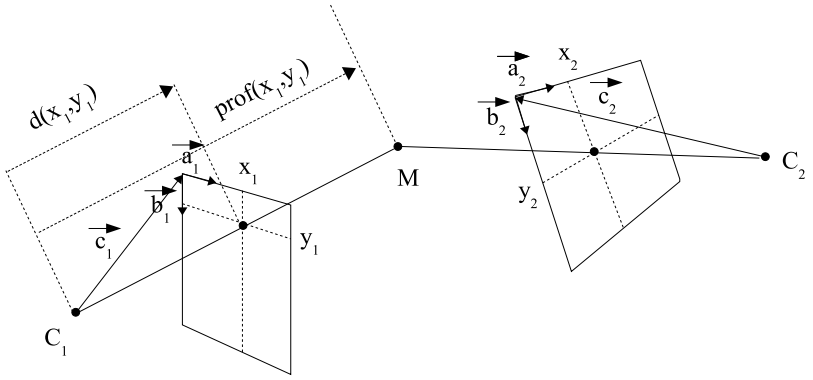
$$y_2 = \frac{\overrightarrow{a_1}.(\overrightarrow{c_2} \times \overrightarrow{a_2})x_1 + \overrightarrow{b_1}.(\overrightarrow{c_2} \times \overrightarrow{a_2})y_1 + \overrightarrow{c_1}.(\overrightarrow{c_2} \times \overrightarrow{a_2}) + \overrightarrow{C_2C_1}.(\overrightarrow{c_2} \times \overrightarrow{a_2})\delta(x_1, y_1)}{\overrightarrow{a_1}.(\overrightarrow{a_2} \times \overrightarrow{b_2})x_1 + \overrightarrow{b_1}.(\overrightarrow{a_2} \times \overrightarrow{b_2})y_1 + \overrightarrow{c_1}.(\overrightarrow{a_2} \times \overrightarrow{b_2}) + \overrightarrow{C_2C_1}.(\overrightarrow{a_2} \times \overrightarrow{b_2})\delta(x_1, y_1)}$$

De plus, McMillan proposa une méthode permettant de gérer correctement la visibilité lors du rendu, sans utiliser de tampon de profondeur[29] [McM95a], ainsi qu'une technique de reconstruction d'image incrémentale permettant de remplir l'image de destination par balayage de lignes direct. Cette technique fût appelée *3D Image Warping,* et a été utilisée dans de très nombreux travaux, dont [MMB97, PZvG00, SGwHS98, CBL99] ainsi que dans [SP99, OBM00] dans lesquels cette technique a été étendue au rendu de cartes de déplacement orthogonales.

Parmi les utilisations du *3D Warping*, les *sprites with depth* de Shade [SGwHS98], introduits en même temps que les LDI (section suivante) peuvent être vus comme une extension des *nailboards* de Schaufler. La profondeur du sprite est utilisée pour le déformer légèrement en fonction du point de vue, augmentant ainsi leur temps de vie. Bien sûr, la déformation ne peut dépasser un certain seuil à cause des problème de dé-occlusion (trous) qui se produisent lors de la déformation.

Une façon de combler les trous a été proposée dans [MMB97] par Mark *et al.* Ils accélèrent la visualisation de séquences vidéos en n'affichant que quelques images de la séquence et en interpolant

[29] Les pixels sont produits d'arrière en avant, du plus éloigné au plus proche et donc se superposent automatiquement.



Disparité généralisée (δ) = rapport de la profondeur du pixel avec la distance pixel-centre de projection.

$$\delta(x_1,y_1)=prof(x_1,y_1) / d(x_1,y_1)$$

FIG. 1.30 – *3D Image Warping*

par déformation 3D les images intermédiaires. Pour créer les images intermédiaires entre une image de départ I_1 et une image d'arrivée I_2 , ils déforment I_1 et I_2 selon un point de vue intermédiaire et superposent les deux images obtenues pour combler les trous (figure 1.31). Ce système va plus vite que l'extraction de la vidéo, surtout à de hautes résolutions d'image. Ils passent ainsi de 5Hz à 60Hz.

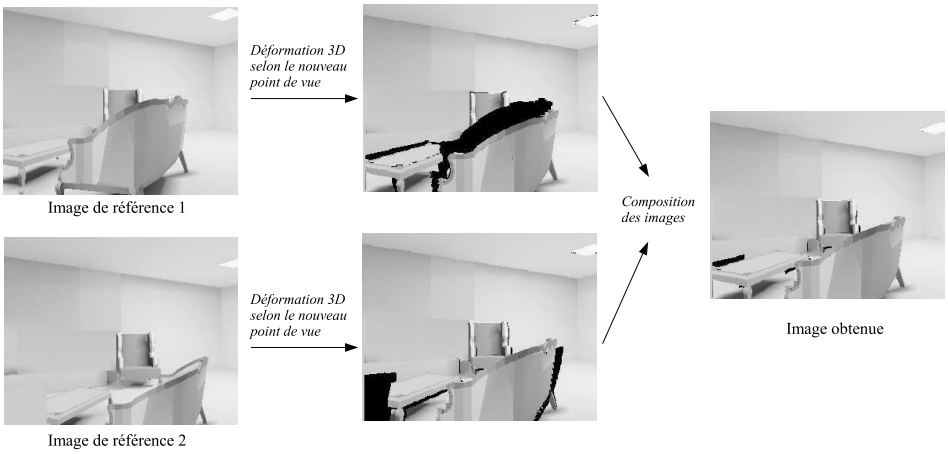


FIG. 1.31 – *Post-Rendering 3D Warping* [MMB97]. Les zones noires des images correpondent aux trous dûs aux effets de parallaxe.

1.4.3.3 Images à plans de profondeurs

A cause des problèmes de dé-occlusion, une seule image ne contient pas assez d'information pour générer un nouveau point de vue complet. Certaines parties de la scène peuvent être cachées dans l'image

de référence et être visibles à partir d'un autre point de vue. Les LDI ("*Layered Depth Images*", Images à Plans de Profondeurs) ont été spécifiquement conçues pour résoudre ce problème. Une LDI stocke plusieurs pixels par rayon optique, c'est à dire que chaque rayon intersecte plusieurs couches de la scène. Le principe des couches (Layers) est un concept très puissant en modélisation et en rendu. Max [MO95, Max96] a proposé d'utiliser un tampon de profondeur en couches (*layered z-buffer*) pour modéliser des arbres. Schaufler [Sch98] utilise des imposteurs en couche pour représenter des objets. Shade *et al.* [SGwHS98] ont généralisé le principe et proposé les LDI comme primitive de rendu, puis Chang *et al*. [CBL99] ont introduit les *LDI-tree* (LDI hiérarchiques) pour résoudre les problèmes d'échantillonnage inhérents aux LDI.

Shade *et al*. ont donc proposé les LDI pour résoudre les problèmes d'occlusion en stockant plusieurs points de la scène par pixel, tout en gardant la simplicité de la déformation d'image de McMillan. Une LDI peut être créée de plusieurs façons. Si on dispose d'un ensemble d'images de profondeurs de la scène, on peut les rétro-projeter sur la caméra de référence. On peut également effectuer un lancer de rayons et stocker les multiples intersections entre la scène et chaque rayon. Une fois la LDI construite, on dispose d'une liste de surfels par pixels. Le rendu s'effectue à l'aide de l'algorithme de déformation 3D d'images de McMillan, adapté de façon à ce que les trous soient comblés avec les informations stockées dans les couches de profondeurs des pixels. Les pixels étant dessinés d'arrière en avant, il est possible d'utiliser l'écrasement de point (*splatting*) directement afin d'améliorer la qualité de l'image en remplissant les trous qui peuvent néanmoins apparaître si la LDI n'est pas assez bien échantillonnée. Ceci permet un rééchantillonnage rapide des pixels de l'image. Dans leurs travaux, ils utilisent un *splat* de taille variable dépendant de la profondeur et de la normale à la surface. Une table d'indexation quantifiée peut être utilisée pour accélérer ce calcul et donc l'algorithme.

Cet algorithme comporte quand même quelques problèmes. Premièrement, les pixels subissent deux rééchantillonnages avant d'arriver à l'écran, ce qui dégrade la qualité de l'image. Deuxièmement, de l'information peut être perdue si une surface est mieux échantillonnée dans une image de base qu'à partir du point de vue de la LDI. Les performances sont moyennes : 8 à 10 FPS pour une résolution de 300x300 pixels sur un Pentium 300Mhz.

Chang *et al.* [CBL99] ont proposé les *LDI-tree* pour résoudre ces problèmes. Un *LDI-tree* est un octree comportant une LDI à chacun de ses noeuds, ainsi que sa boîte englobante. Une face de la boîte englobante est utilisée comme plan de projection (orthographique) pour les images de référence de la LDI. La LDI attachée à chaque noeud de l'octree contient uniquement les échantillons de points de l'objet qui sont contenus dans sa boîte englobante. Le LDI-tree est construit à partir d'images de référence en les rétroprojetant sur les LDI des cellules de l'octree, et en filtrant chaque pixel affecté d'une LDI dans les LDI des cellules parentes. Lors du rendu, l'octree est traversé de haut en bas. A chaque niveau un test est effectué pour déterminer si la LDI du noeud donne assez d'informations si elle est projetée sur le plan de la caméra, et si elle ne donne pas assez de détails, l'opération est effectuée sur les noeuds enfants. Une LDI est considérée comme donnant assez de détails si la taille d'un de ses pixels permet de recouvrir un pixel de l'image de destination. L'image de destination est implémentée sous la forme d'une LDI afin de gérer la visibilité. A la fin du rendu, les pixels projetés sur un même pixel sont triés en fonction de leur profondeur et mélangés à l'aide de leur composante de transparence afin d'obtenir la couleur finale du pixel.

1.4.4 Interpolation de points de vue

Les méthodes décrites dans cette section permettent de calculer des images intermédiaires entre deux points de vue de référence de la scène. La plus simple est le *morphing* [BN92, Smi90]. Le *morphing* est une technique qui permet de créer les images intermédiaires entre deux images par interpolation des couleurs et des formes. Pour cela, des correspondances, des éléments de contrôle, sont établies entre les deux images (segments de droite [BN92] ou des sommets de maillage [Smi90]), puis ces éléments de contrôle sont interpolés, généralement linéairement, entre les deux points de vue. Les autres points de la nouvelle image sont interpolés de façon plus complexe (bilinéaire, splines) entre les éléments de contrôle les plus proches.

Si les deux images représentent deux points de vue d'une scène, cette technique ne fonctionne que si les caméras de référence sont parallèle, sinon l'interpolation est peu réaliste, comme le montre la figure 1.32, et ne correspond à aucun point de vue réel.

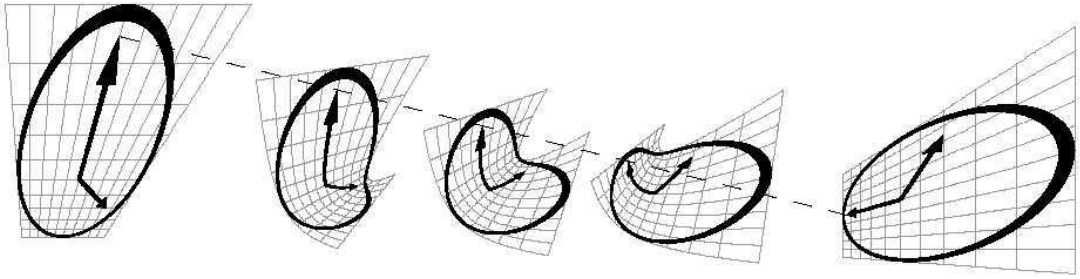


FIG. 1.32 – *Morphing* [SD96]

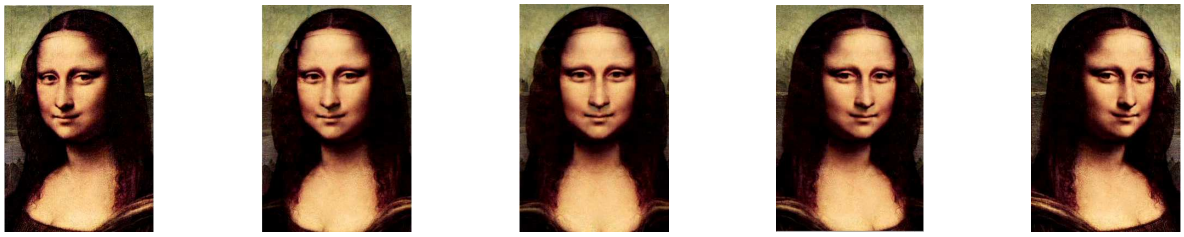
1.4.4.1 Avec connaissance implicite de la géométrie

Dans cette partie, nous survolons les techniques d'interpolation de point de vue issues du domaine de la Vision Stéréoscopique. Etant donné le très grand nombre de travaux dans ce domaine, nous ne nous attardons que sur les méthodes les plus en relation avec nos travaux.

Ces techniques sont dites de "prédiction algébrique de pixels" et ont été spécifiquement developpées pour synthétiser de nouvelles images d'une scène uniquement à partir de photographies. Pour cela, la "géométrie" de l'image est décrite sous forme de cartes de correspondances liant plusieurs vues de référence de cette scène. Une carte de correspondance est un tableau 2D appariant deux pixels issus de deux images de référence de la scène. Les pixels appariés correspondent approximativement au même point 3D dans l'espace de la scène. Grâce à cet appariemment et à la connaissance des caractéristiques des caméras, ont peut obtenir la profondeur des points. Cet appariement peut être créé de diverses façons, comme par exemple en établissant des correspondances entre deux images d'un *référentiel* positionné dans la scène, et extrait par des techniques de traitement d'images. La géométrie est donc dite *implicite* car il faut l'extraire des images. Ces méthodes sont également appelées "techniques de transfert" de pixels car elles consistent à "transférer" dans la nouvelle image chaque pixel de l'image de référence. A la différence de la reprojection de pixels vue précédemment, le problème est inversé : ces techniques visent à prédire la position dans l'image source d'un pixel donné de

l'image à construire. En fonction du type de caméra, de la position des points de vue (alignés ou non, par exemple), les équations algébriques de transfert sont plus ou moins complexe. On distingue les méthode de transfert *trifocal*, *épipolaire* et *linéaire*. Pour une description de ces méthodes, voir la thèse de Frédéric ABAD [Aba03].

Interpolation de point de vue Chen et Williams [CW93] ont ainsi présenté une méthode d'interpolation de point de vue (*View Interpolation)* basée sur le principe du transfert linéaire. Leur méthode synthétise de nouvelles vues à partir d'images de référence pour lesquelles ils établissent des correspondances permettant de déplacer les points d'une image à l'autre. Chaque pixel se déplace donc dans l'image de son emplacement d'origine à son emplacement d'arrivée en suivant une ligne droite. Ainsi, ils génèrent toutes les images intermédiaires entre deux images. D'une façon générale, les pixels en mouvement ne sont pas exactement au bon endroit (par rapport à la scène réelle), mais cette interpolation linéaire donne de bons résultats tant que les deux points de vue de référence ne sont pas trop éloignés. Par ailleurs, si les plans de projections des caméras des points de vue de référence et du point de vue à générer sont parallèles, leur interpolation est exacte. Etant donné que plusieurs pixels peuvent être déplacés au même endroit, un tampon de profondeur doit être utilisé. Les trous qui inévitablement apparaîssent sont remplis en interpolant les couleurs des pixels adjacents différents du fond.



(a) Image de départ

(b) Image d'arrivée

FIG. 1.33 – *View morphing* ([SD96])

Déformation de point de vue Seitz et Dyer [SD96] ont introduit le *déformation de point de vue* (*View Morphing)* qui peut être vues comme une extension des travaux de Chen et Williams. Leur méthode consiste à prédéformer les images de référence avant de les interpoler linéairement. La prédéformation permet d'obtenir des images parallèles pour lesquelles l'interpolation par transfert linéaire est exacte. Une fois l'interpolation effectuée et l'image intermédiaire aux points de vue calculée, l'image est redéformée pour donner l'image finale (*post-warping*). Voir la figure 1.34 page 48.

1.4.4.2 Avec connaissance explicite de la géométrie

D'autres méthodes permettant de créer les images intermédiaires entre des points de vues de référence basées sur une géométrie explicite ont été proposées.

Les images à *multiples centres de projection* [RB98] (*MCOP*, "Multiple Center of Projection"), par exemple, permettent de reconstuire l'image d'une scène pour un chemin 3D positionné par l'utilisateur autour de la scène. Une structure est créée en effectuant le rendu de la scène pour des positions

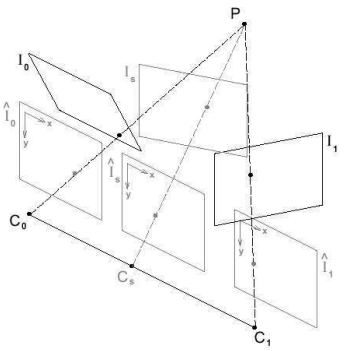
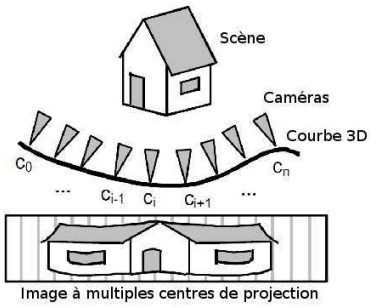
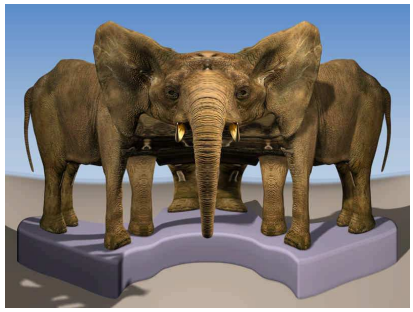


FIG. 1.34 – *View Morphing* de l'image I_0 vers l'image I_1. L'image intermédiaire I_s est créée par *post-warping* de l'image $\hat{I}_s$ elle même obtenue par transfert linéaire entre les images $\hat{I}_0$ et $\hat{I}_1$.

d'échantillonnage de la courbe 3D et en stockant une partie de l'image résultante dans une image spéciale stockant en plus des pixels la position de la caméra (cf. figure 1.35(a)). Cette image représente l'objet vu selon tous les angles de vues du chemin 3D (figure 1.35(b)). Elle est utilisée pour pouvoir effectuer le rendu de la scène pour tous les points de vue intermédiaires entre les positions d'échantillonnage.



(a) Acquisition



(b) Exemple d'image MCOP

FIG. 1.35 – *Images à multiples centres de projection* [RB98]

Dans [Hal98], Halle présente également une méthode permettant d'interpoler des vues d'une scène de synthèse en tirant parti de la cohérence de la variation de perspective d'un point de vue à un autre. Ici, les points de vue de référence sont régulièrement répartis sur un plan. La variation de perspective entre un point de vue et un autre, pour un déplacement horizontal (pour simplifier), sont représentées par une image créée à partir d'une tranche du volume *spatio-perspectif.* Le volume *spatio perspectif* est l'empilement des images obtenues par les points de vue d'une ligne (image 1.36 page 49). Cette tranche-image (une *EPI*, "Epipolar Plane Image") est alors représentée/décomposée en primitives géométriques 2D. Le rendu de la scène selon les points de vues intermédiaires aux points de vue de référence s'effectue en reconstruisant l'image ligne par ligne à partir des *EPI* elles même reconstruites par le rendu des primitives géométriques les constituant. La reconstruction des *EPI* est effectuée par

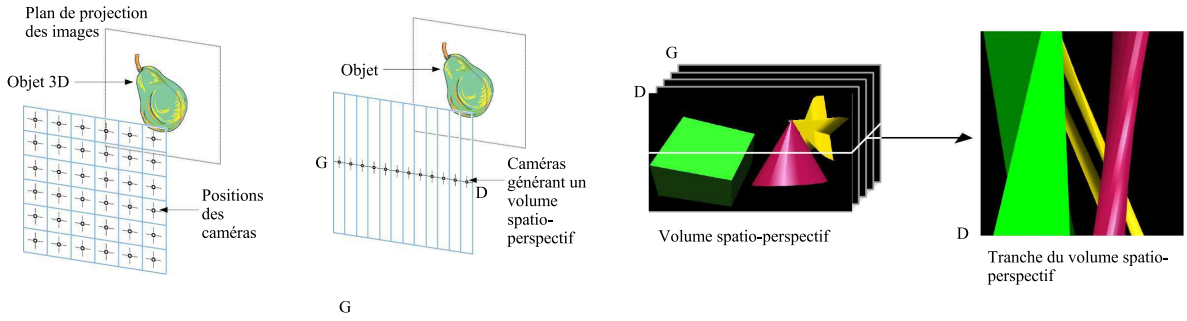


FIG. 1.36 – *Multiple Viewpoint Rendering* [Hal98]

la carte graphique.

1.4.5 Interpolation de rayons lumineux

Le principe de l'interpolation de rayons lumineux est de considérer l'espace à cinq dimensions de tous les rayons lumineux possibles traversant une scène. Cet espace est dénommé *lightfield* (champ lumineux)[Ger39]. Un rayon y est défini par sa position (3 coordonnées) et sa direction (2 angles). Une image est alors un plan 2D plongé dans cet espace et "interceptant" ces rayons lumineux. Les méthodes d'interpolation de rayons visent à acquérir ce champ lumineux (à partir d'images réelles), à le stocker, et à l'utiliser afin de produire des images. La géométrie de la scène n'intervient alors pas dans les calculs, seul le champ lumineux est reconstruit. C'est donc idéalement l'approche parfaite pour la synthèse d'images. Bien sûr, il est impossible d'acquérir intégralement ce champ lumineux, cela demanderait un espace de stockage (et un temps d'acquisition) infini. Ces méthodes se basent donc sur un échantillonnage de cette fonction à des points clés (positions de référence) et interpolent le champ pour les positions intermédiaires. Les méthodes d'*acquisition*, d'*échantillonnage* du champ lumineux, de *compression* des données et d'*interpolation* différencient ces méthodes.

1.4.5.1 Fonction plénoptique

La *fonction plénoptique* [AB91] définit en chaque point (V_x, V_y, V_z) de l'espace l'énergie radiante dans toutes les directions (θ, φ), à tout instant t et pour toute longueur d'onde λ, soit :

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

En ne tenant pas compte du temps (scène statique), ni de la longueur d'onde (éclairement constant), la fonction se réduit à 5 dimensions [MB95] :

$$p = P(V_x, V_y, V_z, \theta, \varphi)$$

Malgré ces simplifications, la quantité de données nécessaire pour décrire une telle fonction est tellement grande que les méthodes que nous allons décrire restreignent les positions (ou orientations de vue) pour lesquelles le champ lumineux pourra être reconstruit. Ainsi, si la scène peut être contenue

dans une boîte englobante, les *Light Fields*[LH96] et *Lumigraphes* [GGSC96] permettent de construire le flot lumineux *autour* de la scène. Si l'espace des points de vue possibles est contenu dans un cercle 2D, les *Mosaïques Concentriques* [SH99] peuvent simuler la fonction plénoptique. Si le point de vue est fixé, la fonction plénoptique peut alors être représentée par un *panorama 2D*.

1.4.5.2 Reconstruction globale

Les méthodes de reconstruction globales reconstruisent la fonction plénoptique afin de générer des images d'une scène statique complète. Avec ces méthodes il est donc possible d'effectuer le rendu de scènes complexes virtuelles ou de scène réelles.

Plenoptical Modeling Dans [MB95], McMillan et Bishop choisissent d'utiliser une projection cylindrique pour représenter un échantillonnage de la fonction plénoptique. Cette projection cylindrique est obtenue à partir d'une caméra vidéo à trépied permettant un mouvement vertical et de rotation. La reconstruction de la fonction plénoptique à partir d'un tel échantillonnage nécessite d'estimer le flot lumineux à chaque position de la caméra quand elle est déplacée. Pour cela, à partir de deux (ou plus) images de référence panoramiques projetées cylindriquement, le champ lumineux est calculé par disparité stéréoscopique. La projection cylindrique à un nouveau point de vue peut être obtenue par un placage de type cylindre↦cylindre, puis peut être reprojetée sur un plan permettant d'obtenir une image.

Light Fields et Lumigraphes Light Fields et Lumigraphes sont tous deux basés sur une paramétrisation à 4D de la fonction plénoptique, quand la scène peut être contenue dans une boîte englobante. Au lieu d'utiliser $p = P(V_x, V_y, V_z, \theta, \varphi)$ pour caractériser un rayon lumineux, ces deux méthodes utilisent une *tranche de lumière* (*light-slab*) à 4D (cf. image 1.37) : $p = P(u, v, s, t)$. Ceci est possible en supposant que le flot lumineux ne change pas le long du rayon, ce qui est le cas pour la lumière *entourant* la scène (et si l'environnement n'est pas un milieu participant).

FIG. 1.37 – Une tranche de lumière (*Light-Slab*) (à gauche) et ses deux plans positionnés dans l'espace de la scène (à droite).

Levoy et Hanrahan ont donc introduit une méthode de rendu de champ lumineux (*Light Field Rendering*). Ils proposent de paramétrer les rayons par deux plans dont le système de coordonnées est (u, v) pour le premier et (s, t) pour le second (par convention) (cf. figure 1.37). Le champ lumineux est alors généré en assemblant une collection d'images. Pour une scène réelle, un tableau 2D d'images est créé en plaçant le centre de projection de la caméra à chaque position (u, v) du premier plan. Dans leurs travaux, ils construisirent un dispositif photographique spécial contrôlé par ordinateur pour acquérir

le champ lumineux d'une scène réelle. La caméra est déplacée aux positions d'une grille 3D dans l'espace et des photographies de la scène sont prises. Chacune de ces images est ensuite projetée (par placage de texture) sur le second plan (s, t) (cf. figure 1.38). La structure peut être compressée avec un système préservant les données : une quantification vectorielle à taux fixé suivi d'un encodage entropique (gzip). Finalement, ils obtiennent un taux de compression de l'ordre de 120 :1, en moyenne. La décompression est faite en appliquant ces deux étapes dans l'ordre inverse. Bien que l'algorithme consiste essentiellement en des interpolations linéaires de couleurs, donc très rapide, la décompression au vol est très coûteuse : environ 25% du temps CPU. Ceci empêche l'utilisation de plusieurs modèles simultanés dans une même scène, limitant la méthode. De plus, aucun travaux n'ont jusqu'ici permi de représenter un environnement complet à l'aide de *Light Fields*, du fait de sa complexité de mise en oeuvre.

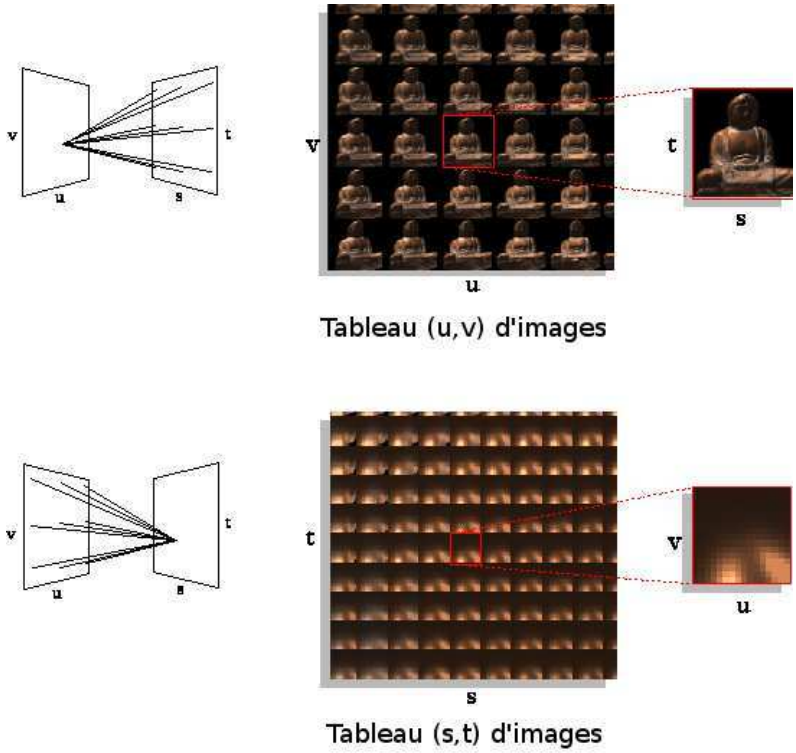


FIG. 1.38 – Acquisition des échantillons d'images de référence pour la construction d'un *light-slab.*

Gortler *et al.* [GGSC96] ont developpé leur système appelé *Lumigraphes* en même temps que les *Light Fields*. Dans leurs travaux, la fonction plénoptique est reconstuite sur la surface d'un cube englobant la scène, comme les *Light Fields*. La technique de paramétrisation d'un rayon de lumière est la même que pour les Light Fields. Dans leur méthode, les échantillons de la fonction plénoptique peuvent être acquis à l'aide d'une caméra tenue à la main. Le calibrage de la caméra, le temps de pose et une version approchée de la géométrie de la scène peuvent être obtenus à l'aide de techniques de Vision par Ordinateur. Etant donné qu'ils utilisent une caméra arbitrairement positionnée, les positions d'échantillonnage ne peuvent être ni spécifiées ni contrôlées ce qui garantit pas que ces échantillons de la fonction plénoptique soient régulièrement répartis. Ils proposent une technique de rééchantillonnage basée sur le modèle 3D approché de l'objet.

Mosaïques concentriques Plus on contraint les positions possibles de la caméra, plus la fonction plénoptique devient facile à reconstruire. Shum *et al.* [SH99] ont proposé les *mosaïques concentriques* qui permet de reconstruire la fonction plénoptique pour un point de vue positionné sur un cercle. Leur méthode compose entre elles différentes photographies prises à différentes positions sur ce cercle. Les rayons sont caractérisés par trois paramètres : rayon, angle de rotation et élévation verticale (3D). Les nouvelles vues sont créées en combinant rapidement les rayons appropriés lors du rendu. Aucun effet de parallaxe n'est capturé par cette technique car les positions de la caméra sont contenues dans un plan et qu'une seule image est prise à chaque point de vue. Lors du rendu néanmoins, des distorsions verticales peuvent apparaître dans l'image obtenue. Différentes corrections sont proposées pour résoudre ce problème. Comparés aux *Light Fields* et *Lumigraphes*, les mosaïques concentriques sont bien moins gourmandes en espace mémoire.

Panoramas Quand le point de vue de capture est réduit à un point, la structure de capture de la fonction plénoptique est dénommée *Panorama.* Ainsi, Chen [Che95] a construit des images panoramiques cylindriques à 360° (à champ de vision vertical limité). Les images panoramiques (figure 1.39) peuvent être créées par rendu conventionnel, à l'aide d'appareils photos panoramiques spéciaux ou en combinant des photographies entre elles. A chaque point de capture, une rotation et un zoom peuvent être effectués en utilisant une déformation de type cylindre↦plan. La naviguation en 3D se fait de façon discrète, d'un point à l'autre.

FIG. 1.39 – Images panoramiques.

1.4.5.3 Reconstruction locale

Les méthodes de reconstruction locale de la fonction plénoptiques permettent de reconstuire le champ lumineux à la *surface* d'un modèle virtuel. Un tel champ lumineux est caractérisé par une fonction appelée *Surface Light Field* (SLF) [WAA+00], ou *Lumisphère*, donnant l'intensité lumineuse émanant d'un point de la surface d'un objet 3D dans toutes les directions, soit $I(R, V, B) \leftarrow f(r, s, \theta, \varphi)$. (r, s) définissent, comme les coordonnées de texture, la position du point sur la surface et (θ, φ) la direction du rayon émis. Pour reconstuire cette fonction à partir de photographies, il est nécessaire

d'associer aux photographies le modèle virtuel 3D de l'objet, et donc de connaitre exactement la position des point de vue. Ceci nécessite du matériel d'acquisition spécialisé, exactement comme les *Light Fields.* L'avantage du SLF par rapport aux méthodes de reconstruction globale est que l'ensemble des points de vue de visualisation est plus grand : avec un *light slab,* par exemple, il est impossible de se rapprocher du modèle reconstruit et de plus, il faut combiner plusieurs *light slabs* pour pouvoir faire le rendu de l'objet selon tous les angles. De plus, avec le SLF, il est possible de positionner plusieurs modèles dans une même scène, ce qui est très difficile avec les *Light Fields*.

Dans [CBCG02], le SLF est implémenté à l'aide du matériel en compressant l'énorme base de données d'images (jusqu'à un rapport de 5500 :1) et en stockant les informations dans des textures. Ainsi, la décompression et le placage sont effectués par la carte graphique.

Le taux d'affichage d'un modèle 3D est de 8 FPS en moyenne pour les SLF et d'environ 50 à 70 FPS pour les SLF accélérés avec la carte graphique (pour une qualité de compression moyenne). Le réalisme des images résultantes est très impressionnant, néanmoins, le dispositif d'acquisition est très complexe, les temps de précalculs extrêmement longs et il est impossible d'utiliser un tel objet dans une scène où l'éclairement change (ou de le faire se déplacer). Finalement, le rendu simultané de plusieurs de ces objets rendus fait baisser drastiquement le taux d'affichage.

1.4.6 Autres techniques de reconstruction

Ces techniques peuvent être considérées comme les toutes premières méthodes de rendu à base d'image. Elles sont "brutales", extrêmement gourmandes en espace mémoire et très limitées en termes de degrés de liberté, mais permettent de reconstruire un point de vue "quelconque" d'une scène à partir d'images uniquement.

1.4.6.1 Movie-maps

Les *movie-maps* ont été introduites en 1980 par Lippmann [Lip80]. Le principe est de créér une base de données d'images de la scène (une ville) et de la stocker sur vidéodisque. Une méthode spéciale permet d'accéder à un de ces points de vue en fonction du point de vue virtuel (l'observateur) et d'afficher l'image la plus proche dans la base de données. L'utilisateur peut contrôler sa vitesse et son chemin parmi ceux autorisés. Dans le *musée virtuel* [Hof92], l'utilisateur peut se déplacer de point à point par une vidéo et de regarder à 360° aux points de vue d'intérêt.

1.5 Conclusion

Comme nous nous sommes attachés à le montrer dans cet état de l'art, le rendu de modèles géométriques complexes en temps réel a généré beaucoup de travaux et autant d'approches originales du problème. Les différentes classes de méthodes que nous avons présentées (simplification géométrique, rendu à base de points et rendu à base d'images) ont toutes pour but d'obtenir une grande complexité visuelle au moindre coût possible en terme de temps d'affichage et d'espace mémoire.

Dans ce cadre, la simplification de maillage peut être utilisée pour réduire le coût d'affichage d'un modèle complexe, quand celui-ci est vu de loin. Cette approche est très efficace mais un compromis doit être fait entre la qualité de la simplification, la complexité de l'algorithme et le type de maillage (tous les maillages ne sont pas simplifiables). De façon générale, même les algorithmes les plus performants demandent un temps de calcul qui dégrade les performances si on les applique à plusieurs modèles simultanément (ce qui est le cas général dans une application de simulation). De plus, par essence même, si le modèle est vu de près, il est affiché avec tous ses détails ce qui évidemment va a l'inverse du but recherché. Finalement, de part leur complexité même, ces algorithmes utilisent le processeur central, ce qui réduit la puissance de calcul disponible pour le reste de l'application.

Les techniques de rendu à base de points permettent d'afficher des modèles extrêmement complexes [GBP04], mais ne sont pas intégrables telles qu'elles dans un moteur de simulation d'environnements en 3D. En effet, les points sont par essence déconnectés, et donc difficiles à manipuler dans le cas ou l'on souhaite, par exemple, prendre en compte les collisions entre objets (ou tout autre interaction physique faisant appel à la surface de l'objet). Notamment, la projection de l'ombre du modèle sur son environnement semble très difficile à mettre en oeuvre.

Les méthodes de rendu à base d'images génèrent des images du modèle sur la base d'échantillons visuels (réels ou virtuels) de celui-ci.

Ces échantillons peuvent être utilisés pour réhausser l'apparence d'un modèle géométrique simple, comme dans le cas des techniques de *placage de texture*. Ils peuvent également complètement le remplacer au sein d'un scène virtuelle, comme dans le cas des *imposteurs*. Le problème général de ces méthodes provient du fait que les échantillons visuels ne sont valides que pour une seule position d'observation du modèle : la position d'échantillonnage. Dès que l'on s'écarte du point de vue de capture, les effets de parallaxes qui ne se produisent *pas* révèlent la nature de l'approximation visuelle qui a été faite lors de l'emploi de ces méthodes. Plus drastiquement encore, ces échantillons ne sont valides que pour une seule condition d'éclairement : celle de la capture. Ainsi, si l'éclairement de la scène change, cela peut révéler rapidement les approximations : les reflets spéculaire, par exemple, dépendent de la position de la surface et de l'éclairement ; les inter-réflexions ou les projections d'ombres, là aussi, ont un grand impact visuel sur l'image. Ces effets visuels, pour ne citer qu'eux, doivent donc d'être pris en compte par la technique de rendu.

Afin de gérer ces problèmes, ces échantillons visuels peuvent être réhaussés d'une information de relief liée à leur point de vue de capture. Ce relief peut être utilisé afin de traiter les effets de parallaxe, comme dans l'*interpolation de point de vue*, la *déformation 3D d'images* ou le *parallax mapping*. Il peut être utilisé également pour répercuter visuellement le changement d'éclairement, comme dans le cas du *bump mapping* ou des techniques d'*auto-ombrage* et d'*inter-réflexions*.

Cette représentation discrète de la surface de l'objet peut être obtenue (sous certaines conditions) à partir d'images réelles par des techniques d'analyse d'image ou de vision stéréoscopique. Dans le cas d'objets 3D virtuels, le tampon de profondeur (*z-buffer*) peut être directement utilisé. Le problème principal de ces relief exprimés par carte d'élévation ou de profondeurs est que les effets de parallaxe ne sont pas pris en compte. Ceci génère des zones sur- ou sous-échantillonnées quand le relief est observé selon un point de vue différent du point de vue d'échantillonnage. Les *images à plans de profondeurs,* entre autres, permettent de traiter ce problème mais sont beaucoup trop gourmandes en temps CPU pour être employées dans une application de simulation d'environnement 3D interactive.

Dans le cas de l'*interpolation de rayons lumineux,* le problème est la taille des données à traiter, et leur acquisition. Le matériel requis pour obtenir des échantillons lumineux permettant de reconstruire la fonction plénoptique est très spécifique et délicat à mettre en oeuvre. De plus, la taille des données (plusieurs dizaines de giga octets) rend très difficile leur manipulation. Par ailleurs, le champ lumineux n'est reconstuit que pour un éclairement fixe, et il n'existe pas encore d'algorithmes permettant de prendre en compte en éclairement variable de la scène en temps réel. Finalement, le méthodes les moins gourmandes en espace de stockage (*panoramas* et *mosaïques*) sont très limitées en termes de degré de liberté.

En conclusion, le problème de la complexité visuelle optimale au moindre coût (en termes d'espace mémoire et de temps de calcul) n'a evidemment pas de solution générale. Pour générer une image, les deux approches extrêmes seraient la capture du champ lumineux complet, nécessitant un espace de stockage infini, et la simulation complète du fonctionnement de la lumière, nécessitant un temps de calcul infini. Toutes les méthodes que nous avons présentées ici proposent un compromis entre ces deux extrêmes. Ce compromis est basé en particulier sur les spécificités du matériel disponible à un instant donné.

2

Extension au temps réel de l'interpolation d'images

2.1 Introduction

Dans ce chapitre, nous présentons les travaux que nous avons menés sur dans le domaine de l'interpolation d'images. D'un point de vue général l'interpolation d'images permet d'obtenir de nouveaux points de vue d'une scène à partir de points de vue de référence pré-acquis de celle-ci. Nous nous sommes donnés comme base les principes du *placage de texture dépendant du point de vue* (section 1.4.1.7) et de l'*interpolation de points de vue* (section 1.4.4), puis nous nous sommes positionnés dans le contexte du rendu en temps réel d'un maillage géométrique complexe.

Le placage de texture dépendant du point de vue nécessite des opérations complexes à effectuer par pixel, lors du rendu. Dans leur méthode de placage de texture dépendant du point de vue, Debevec *et al.* [DYB98], utilisent une pondération unique pour tous les pixels d'une face affichée, ce qui permet d'effectuer le mélange en temps réel. Le principal problème est que le modèle approché de la géométrie d'origine doit être créé à la main à partir des images de référence. Dans ce chapitre, nous avons remplacé un modèle 3D par un modèle géométriquement simplifié l'englobant et donc très facile à obtenir. Nous avons également tiré parti de la connaissance géométrique que nous avons de la scène originelle afin de prendre en compte un éclairement variable de la scène, ce qui n'a pas été fait dans les travaux antérieurs basés sur des photographies réelles.

L'interpolation de points de vue permet de produire les images intermédiaires entre deux images de référence, de façon à obtenir une animation continue entre deux images. Ces images peuvent être issues de points de vue d'une scène réelle, auquel cas il est nécessaire d'extraire les informations de profondeur par des techniques d'analyse d'images afin de pouvoir générer les images intermédiaires. De façon générale, l'interpolation de points de vue ne fonctionne bien que pour des points de vue peu éloignés les uns des autres, et les techniques d'appariement stéréoscopique permettant de retrouver la profondeur des points d'une image sont assez approximatives. Dans nos travaux, nous avons appliqué cette technique à l'interpolation à deux points de vue de référence d'une scène virtuelle. Ceci nous permet d'obtenir l'appariement de façon précise à partir des images de profondeurs des points de

vue de référence, et nous pouvons intégrer les images générées directement dans un moteur de rendu conventionnel, tout en prenant en compte l'éclairement de la scène.

Interpolation de points de vues par blocs de pixels L'interpolation de points de vue opère par déplacement de pixels à l'écran de façon à adapter les images de référence à des points de vue intermédiaires. Partant de la constatation que beaucoup de pixels se déplacent dans la même direction et de la même distance, nous les avons regroupé afin de les traiter par blocs plutôt que séparément. Comme nous le verrons, le déplacement des points dépend de leur profondeur, ce qui nous a finalement amené à la création d'un maillage basé sur les tampons de profondeur des images de référence, à la façon des imposteurs maillés (section 1.4.2.5 page 41).

De façon générale, ces méthodes permettent de créer de nouveaux points de vue d'une scène à partir d'images de référence. Comme les calculs (mélange de couleurs, déplacement de points) s'effectuent par pixel, le temps de rendu ne dépend plus de la complexité de la scène, accélérant d'autant la vitesse de rendu. Etant donné que nous avons employé un objet 3D virtuel pour générer les images de référence, nous avons accès directement aux informations de profondeurs nécessaires. Finalement, nous pu tout aussi facilement prendre en compte l'éclairement de la scène.

2.2 Interpolation d'images de référence en fonction du point de vue

2.2.1 Principe

Dans cette section, nous présentons nos travaux portant sur l'interpolation d'images de référence en fonction du point de vue. Ces travaux sont en rapport avec le *placage de texture dépendant du point de vue* (*VDTM*, "View-Dependent Texture Mapping") (voir section 1.4.1.7) car nous effectuons un mélange pondéré des couleurs des images de référence en fonction de la position d'observation, et que nous l'appliquons à un maillage approchant la géométrie du modèle complexe de base.

Mais la différence principale avec le VDTM est que nous nous basons sur un modèle 3D au lieu d'images réelles pour acquérir les images de références. Nous avons tiré parti de cette connaissance explicite de la géométrie originelle pour appliquer le mélange d'images à un maillage simplifié automatiquement généré : par rapport au VDTM nous n'avons pas à traiter les images de référence pour en extraire un maillage approché. De plus, le maillage géométrique remplaçant le modèle complexe est automatiquement paramétré : les coordonnées de textures sont générées lors du rendu, ce qui évite cette manipulation fastidieuse.

2.2.2 Points de vue de référence

2.2.2.1 Répartition et acquisition

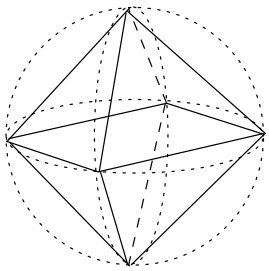
La disposition des points de vues de référence que nous avons choisie est similaire à celle utilisée dans les travaux de Pulli *et al.* [PCD+97] : ils sont répartis sur une sphère englobant le modèle. Néanmoins, nous avons choisi de répartir les points par subdivision régulière d'un octaèdre de base plutôt que créer le maillage à partir de points et d'une triangulation de Delaunay, et ce pour des raisons pratiques exposées en section 2.2.2.2.

La création de l'ensemble des points de vue se fait donc à partir d'un octaèdre unité (figure 2.1(a)) que nous subdivisons un nombre fixé de fois jusqu'à atteindre le nombre de points de vue de référence que nous souhaitons. La figure 2.1(b) illustre le principe de la subdivision : chaque étape de récursion crée le nouveau maillage raffiné de la sphère en subdivisant en quatre les triangles du maillage de base et en projetant les points ajoutés sur la surface de la sphère unité. Dans l'exemple, nous obtenons un maillage composé de 32 faces et de 18 sommets à partir des 8 faces et des 6 sommets de l'octaèdre de base. La figure 2.1(c) montre un maillage sphérique de 128 points obtenu avec notre méthode.

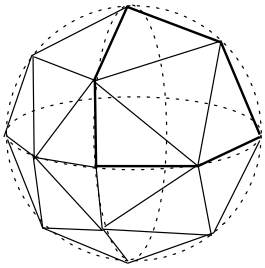
A partir des directions données par les sommets du maillage de la sphère, nous générons autant de caméras de référénce dont les axes optiques passent par le centre de la scène et dont les plans de projection sont tangeants à la sphère englobante de l'objet. La direction verticale de la caméra est quelconque. Finalement, nous effectuons le rendu conventionnel de l'objet 3D selon chaque point de vue de référence et nous stockons les images obtenues dans la carte graphique, sous forme de textures.

2.2.2.2 Recherche des meilleurs points de vue

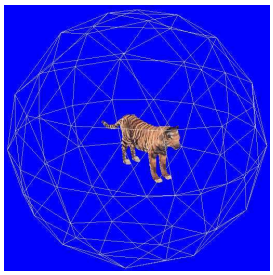
Lors du rendu, le choix des images de référence à mélanger est très simple : il s'agit des images issues des trois points de vue de référence dont les axes optiques sont les plus proches (angulairement) de celui du point de vue virtuel. Pour cela, la caméra virtuelle associée au point de vue à reconstruire



(a) Au départ, la sphère englobant l'objet est approximée par un octaèdre.



(b) A chaque pas de récursion, les faces sont subdivisées en 4.



(c) Maillage de 128 points de vue de référence obtenu avec cette méthode

FIG. 2.1 – Répartition des points de vue sur la sphère.

doit, comme les points de vue de référence, être dirigée vers le centre de la scène. Pour trouver ses trois plus proches points de vue, il suffit alors de trouver l'unique face du maillage de la sphère qui intersecte l'axe optique de la caméra virtuelle.

k	**nb. de faces**	**nb. de tests**
0	8	8
1	32	12
2	128	16
3	512	20
4	2048	24
k	2.4^{k+1}	$4.k + 8$

TAB. 2.1 – Recherche en $\log(n)$ des trois meilleurs points de vue de référence. k = niveau de récursion (profondeur de l'arbre)

Au lieu de tester l'intersection de l'axe optique avec toutes les faces du maillage de la sphère, nous effectuons le calcul d'intersection en choisissant les faces stockées dans les noeuds de l'arbre de récursion créé lors de la subdivision de l'octaèdre, et ce, jusqu'à arriver sur une face terminale (une feuille de l'arbre). Ainsi, la détermination des trois plus proches point de vues s'effectue avec un nombre de tests de l'ordre de $\log(n)$ au lieu de n, avec n le nombre de faces de la sphère (cf. figure 2.1).

2.2.2.3 Schéma de pondération

La composition des images de référence s'effectue avec une pondération identique sur tous les pixels. Pour calculer la contribution de chaque point de vue à l'image finale, nous nous basons sur les coordonnées barycentriques de l'intersection entre la direction de vision et la face du maillage de la sphère, calculées durant l'étape précédente (cf. figure 2.2).

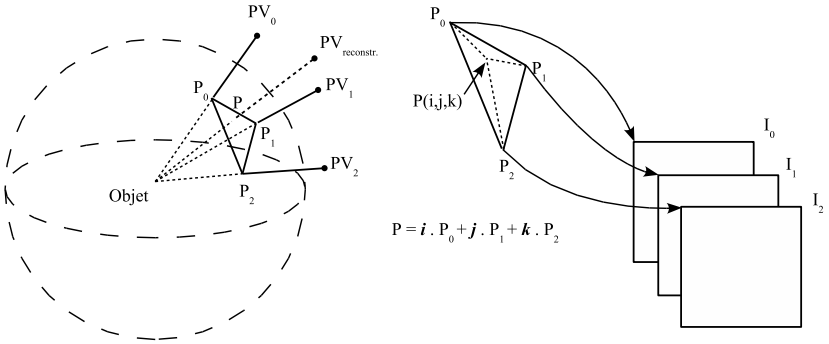


FIG. 2.2 – Calcul de la contribution des points de vue de référence

Ainsi, pour chaque pixel c de l'image, on a : $c = i.c_0 + j.c_1 + k.c_2$, avec (i, j, k) les coordonnées barycentriques de l'intersection et c_0 , c_1 et c_2 les pixels issus respectivement des images de référence I_0 , I_1 et I_2.

2.2.2.4 Référentiel de mélange

Afin de mélanger les pixels des trois images de référence, nous devons disposer d'un référentiel 2D au sein des images car celles-ci ont été obtenues avec une caméra orientée aléatoirement (cf. figure 2.3).

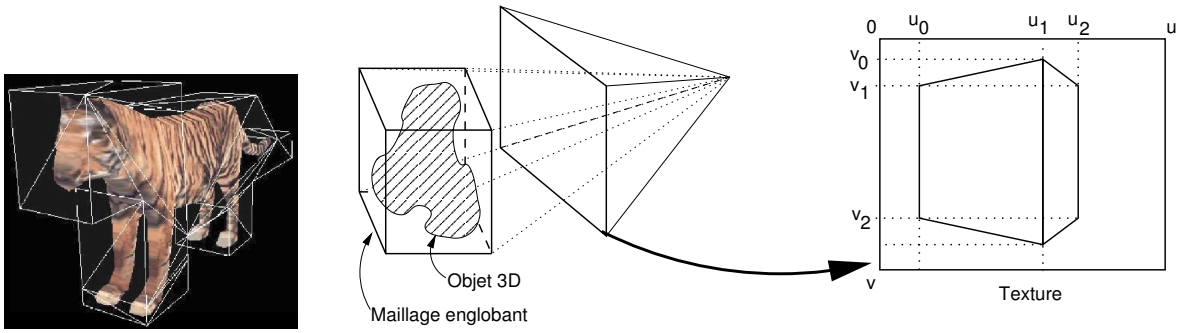
FIG. 2.3 – Exemples d'images de référence obtenues avec notre méthode

Ce référentiel est obtenu directement à partir du maillage englobant de la scène (figure 2.4(a)) pour lequel on associe à chaque image de référence ses coordonnées 2D projetées sur chaque point de vue de référence (figure 2.4(b)).

En pratique, plutôt que de stocker ces coordonnées de texture pour chaque point de vue, nous les recalculons lors du rendu (section suivante). Ainsi, les seules informations que nous stockons en chaque point de vue de référence sont sa direction, l'image de la scène (stockée dans une texture) et la matrice de transformation de la caméra.

2.2.3 Rendu

Le rendu s'effectue comme suit :



(a) Maillage utilisé comme référentiel pour le mélange

(b) Calcul du référentiel par projection du maillage simplifié sur le point de vue de référence.

FIG. 2.4 – Référentiel des images de référence

1. *CPU :* Calcul des points de vue référence les plus proches. Cette partie est effectuée par le processeur central et permet d'obtenir les trois points de vue de référence PV_0, PV_1 et PV_2, ainsi que les coordonnées barycentriques (i, j, k) du point de vue virtuel (figure 2.2).
2. *GPU* **:** Affichage du maillage simplifié selon le point de vue virtuel. Partie effectuée par la carte graphique.
   (a) *Vertex Shader* : Chaque sommet 3D p du maillage est projeté sur les plans de projection des caméras des trois points de vue de référence. On obtient donc 3 points 2D : p_{r0}, p_{r1} et p_{r2}. Chacun de ces points est envoyé dans un canal de coordonnées de texture distinct : $p_{ri} \Rightarrow uv_i$. De cette façon, chaque triangle affiché du maillage simplifié est automatiquement balayé par placage de texture projectif dans la texture de chaque point de vue de référence.
   (b) *Pixel Shader* : L'affichage final est effectué par mélange de chaque couleur issue de chaque texture de référence I_i (voir figure 2.2) aux coordonnées de texture uv_i. Ainsi la couleur c d'un pixel affiché est obtenue par : $c = i.I_0(uv_0) + j.I_1(uv_1) + k.I_2(uv_2)$.

2.2.4 Résultats

La figure 2.5 montre le résultat de l'interpolation des trois points de vues. A gauche des images sont affichées les textures des points de vue de référence sélectionnés. Au centre, le triangle de couleur représente les coordonnées barycentriques de l'intersection entre le point de vue virtuel et la face de la sphère des points de vue. Dans cet exemple, 128 points de vue de référence ont été utilisés.

Comme le montre l'image de gauche, quand le point de vue virtuel est proche d'un point de vue de référence (par rapport aux deux autres), le résultat visuel est acceptable. Par contre dans le cas le plus défavorable (image de droite) où le point de vue virtuel est au centre d'une face (coordonnées barycentriques du point de vue virtuel égales à 1/3), le mélange donne les plus mauvais résultats. Ceci est dû, comme nous le montrons dans le paragraphe suivant, au trop grand écart entre le maillage simplifié et le maillage d'origine.

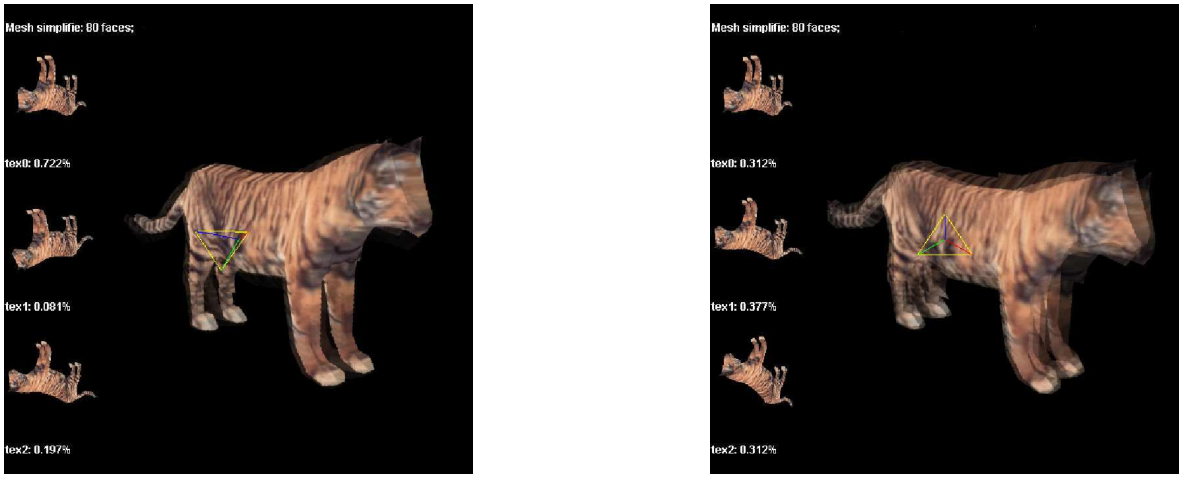


FIG. 2.5 – Mélange pondéré de couleurs de trois points de vue de référence.

Dans cet exemple, le modèle du tigre fait approximativement 300 faces, et le modèle simplifié 80. La pré-acquisition des 128 points de vue de référence a pris environ 1 mn (carte graphique ATI Radeon 9700 Pro). La taille d'une texture, stockée dans la carte graphique, compressée au format DXT1 pour une définition de 512x512 pixels est de 128 Ko. Le rendu s'effectue extrêmement rapidement, car le calcul par pixel, une simple opération de mélange, est très peu coûteux. Dans cet exemple, nous obtenons plus de 300 FPS.

Dans la figure 2.6 nous avons utilisé un maillage simplifié (sous-figure (c)) dont la surface est proche de celle du modèle original (sous-figure (a)). Le résultat est meilleur en ce qui concerne les effets de flou à l'extérieur de l'image de l'objet, mais ils subsistent à l'intérieur de l'image (sous-figure (b)).

Ce modèle est constitué de 1,1 million de faces, et s'affiche à un taux d'environ 20 FPS de façon conventionnelle (NVIDIA GeForceFx 5600). Avec l'emploi du placage de texture, nous obtenons un taux d'affichage supérieur à 100 FPS. Dans cet exemple, nous avons utilisé trois points de vue de référence très éloignés entre eux (angle entre les axes optiques > 45°).

2.2.4.1 Artefacts visuels

Afin d'isoler les problèmes, nous avons appliqué le mélange de couleurs en utilisant uniquement deux images de référence, et en interpolant les couleurs issues de chaque point de vue en fonction d'un paramètre t représentant l'angle de la caméra virtuelle entre les deux points de vue. Soit : $I(x, y) = (1 - t).I_0(u_0, v_0) + t.I_1(u_1, v_1)$, avec $t \in [0, ..., 1]$. Cette manipulation correspond à considérer l'évolution du point de vue virtuel le long d'une arête d'un triangle du maillage de la sphère : seuls deux points de vues interviennent dans le calcul de l'image résultante.

Comme le montre la figure 2.7, les artefacts visuels qui apparaîssent lors de l'affichage du maillage simplifié sont d'une part du *ghosting*, qui correspond au fait que l'on voit les silhouettes floues des images de départ et d'arrivée lors de l'interpolation et d'autre part du *blurring,* c'est à dire des effets de flou dus à la superposition des deux images.

Les artefacts visuels sont dus aux effets de parallaxe, eux même dus à l'écart entre la surface réelle et la surface approximée. Quand cette distance est trop grande, ce qui est le cas dans nos tests avec le tigre, les effets de *ghosting* dégradent complètement l'image. La figure 2.8 illustre ces problèmes.

(a) Modèle original (approx. 1 million de triangles)

(b) Modèle simplifié avec placage de texture dépendant du point de vue

(c) Vue en fil de fer du modèle simplifié (6000 triangles)

FIG. 2.6 – Placage de texture dépendant du point de vue. Les trois points de vue de références utilisés sont éloignés les uns des autres d'un angle supérieur à 45°. La figure du centre montre les effets de *ghosting* qui apparaissent (principalement autour du visage).

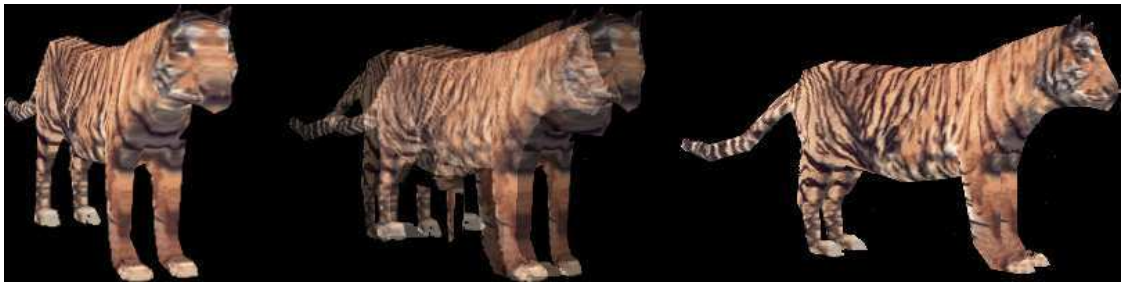


FIG. 2.7 – Interpolation de deux points de vue. A gauche, l'image de départ ($t = 0$). A droite, l'image d'arrivée $(t = 1)$. A centre, le résultat de l'interpolation à $t = 0.5$.

La figure 2.8(a) montre la provenance du problème de flou (*blurring*). Celui-ci est dû au fait que, étant donné un point p à dessiner, la couleur résultante sera un mélange des couleurs de p_0 et de p_1 , au lieu de la couleur du point réel P_R de la surface originelle. On constate également que l'erreur (la distance entre p_R et p_1 ou p_0) augmente en fonction de l'écartement de la surface approchée par rapport à la surface réelle ainsi qu'en fonction de l'écart entre les points de vue de référence.

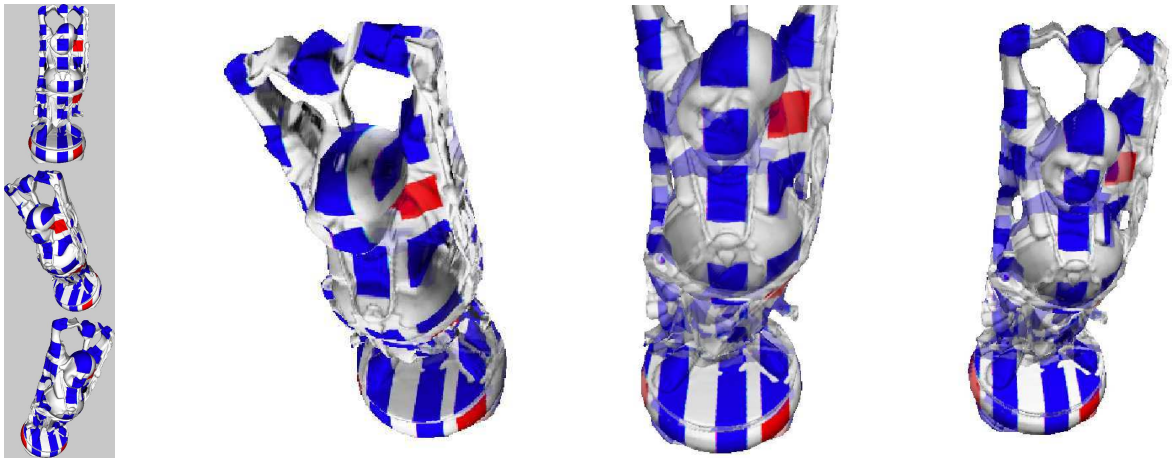
La figure 2.8(b) illustre le problème de *ghosting*. L'origine du problème est la même que pour le flou, mais la différence est que dans ce cas, p est coloré uniquement avec la couleur de p_0 c'est à dire avec l'image du point de vue 0. C'est l'artefact le plus remarquable des deux. Afin d'éviter cet effet en bordure de l'image, la solution classique ([PCD+97, DTM96, PHL+98]) consiste à stocker un coefficient de pondération (en plus de la texture de couleur) en chaque pixel de chaque image de référence. Ce coefficient vaut 1 au centre de l'image et décroît vers 0 en bordure de l'image de l'objet (voir la figure 1.22), ce qui permet "d'éliminer" (dans ce cas) le point de vue 1 de la pondération.

(a) *Blurring* (flou) (b) *Ghosting*

FIG. 2.8 – Artefacts visuels lors du mélange d'images

Néanmoins, cette solution n'est pas satisfaisante dans tous les cas car elle ne supprime que le "ghosting extérieur" à l'image.

Le "ghosting intérieur" est dû au fait que que certaines parties de l'objet sont cachées à partir d'un point de vue, mais que ce point de vue est quand même utilisé dans la pondération des couleurs. Dans la figure 2.9 nous avons mis en évidence le problème en colorant la surface du modèle de base avec une texture à motifs francs qui permet de faire ressortir les effets de ghosting.



(a) (b) (c) (d)

FIG. 2.9 – *Ghosting*. (a) : les trois points de vue de référence. (b), (c) et (d) : résultat de l'interpolation de textures selon différents angles, avec éclairage fixé.

Les artefacts de flou peuvent être réduits en utilisant des points de vue de référence plus proches les uns des autres mais le pendant est que l'on augmente la taille mémoire de la structure.

2.2.4.2 Interpolation des normales

Une autre piste que nous avons suivie pour augmenter la qualité visuelle de l'interpolation a été de prendre en compte la variation de l'éclairement de la scène. En effet, l'ombrage du modèle augmente le réalisme du rendu et la variation de cet ombrage intervient pour beaucoup dans la perception qu'a l'observateur de la cohérence de l'image.

Pour pouvoir faire varier l'éclairement de l'objet le principe est de pondérer l'éclairement de la surface en fonction de son orientation par rapport à la lumière. Pour cela, nous avons employé le modèle de Phong dans lequel nous avons ignoré les reflets spéculaires liés à la position de l'observateur. L'orientation de la surface est représentée par sa normale qui peut être issue directement du maillage simplifié ou précalculée à partir du modèle complet et stockée dans une texture.

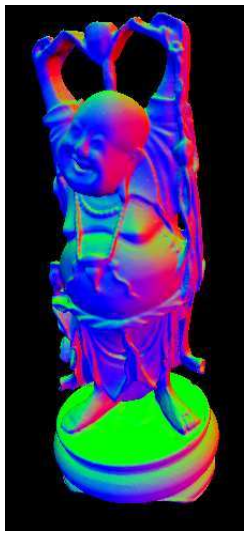


FIG. 2.10 – Texture de normales

Dans le premier cas, on peut utiliser une normale par face (modèle d'éclairement de Lambert) ou interpoler les normales des sommets au sein de chaque triangle (modèle de Phong). Dans le second cas, la normale utilisée est issue de la *carte de normales.*

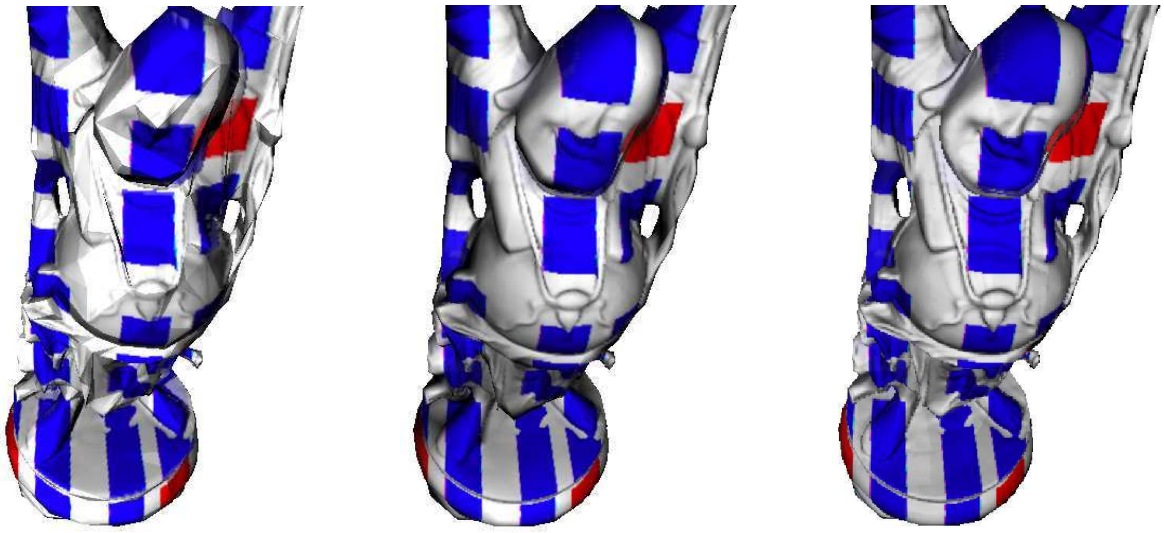
Dans notre cas, la carte de normales est stockée dans une texture associée à chaque point de vue de référence. Pour l'acquérir, nous effectuons le rendu de l'objet complexe en remplaçant la sortie des composantes (r, g, b) du pixel par les cooordonnées (dx, dy, dz) de la normale à la surface interpolée au sein des faces. L'image obtenue représente alors en chaque pixel la normale à la surface (image 2.10). Les composantes de la normale étant des valeurs signées dans l'intervalle [-1.0,...,1.0], nous devons les transformer pour qu'elles puissent être stockées en lieu et place d'une texture de couleurs dont les valeurs sont représentées sur 8 bits, dans l'intervalle [0,...,255]. La perte de précision s'avère visuellement indiscernable.

La figure 2.11 montre le résultat du placage de texture dépendant du point de vue avec un éclairement variable issu d'une unique source de lumière. Dans la figure (a), nous utilisons une seule normale par face du maillage simplifié (*flat shading*). Dans la figure (b) les normales sont interpolées au sein des faces à partir des sommets (*Phong Shading*). Dans la figure (c), nous utilisons les normales issues des textures de normales des points de vue de référence.

Les normales sont mélangées comme les couleurs, en fonction des coordonnées barycentriques du point de vue virtuel. Comme le montre cette dernière image, les effets de flou sont atténués et le relief plus apparent. Le modèle simplifié est ici composé de 3000 faces, contre 1,1 million pour le modèle complet. Le taux d'affichage est de 70 FPS dans les deux premiers cas, et de 60 FPS dans le dernier, pour une définition de 512x512 pixels sur une GeForce 5950. Bien que l'emploi de textures de normales double le coût mémoire, celui-ci reste acceptable, au vu de l'amélioration du taux d'affichage et de la qualité visuelle obtenue.

2.2.5 Discussion

L'interpolation d'images de références en fonction du point de vue appliqué à une scène virtuelle vue de près donne de bons résultats visuels quand les effets de parallaxe sont faibles, c'est à dire soit :



(a) Modèle de Lambert : une normale par face.

(b) Normales du maillage simplifié interpolées au sein des triangles (Modèle de Phong)

(c) Normales interpolées issues des trois textures de normales des points de vue de référence.

FIG. 2.11 – Placage de texture avec interpolation des normales pour le calcul de l'éclairement. Le maillage simplifié utilisé est composé de 3000 faces.

- dans le cas où l'on dispose d'un grand nombre de points de vue, proches les uns des autres, et d'un maillage simplifié également proche de celui d'origine.
- ou dans le cas où le relief de la scène originale est faible.

Originellement, cette technique a été appliquée sur des modèles réels pour lesquels les points de vue étaient des photographies, où les effets de flou sont moins apparents (car les photographies sont plus bruitées que nos images de synthèse). Ceci nécessitait des conditions d'éclairement particulières pour éviter les reflets spéculaires dans les images. De plus, le maillage simplifié de la scène devait être reconstruit à la main. Bien que nous n'ayons plus ces restrictions, la taille mémoire nécessaire au stockage des points de vues est bien trop importante pour être utilisable en l'état : si l'on a peu de points de vue de référence, ceux-ci doivent être éloignés pour couvrir l'intégralité de la scène, et les effets de *ghosting* et de flou dégradent énormément le résultat. Bien sûr, plus on éloigne l'objet de l'observateur, moins ces effets deviennent visibles, mais moins la méthode a d'intérêt. Les artéfacts de *ghosting*, directement dérivés des effets de parallaxe, sont impossible a résoudre de cette façon, c'est à dire par une interpolation brute des images de référence. Pour autant, le principal avantage de cette technique est d'accélérer énormément la vitesse de rendu, qui n'est quasiment plus liée à la complexité de la scène. De plus, il est toujours possible d'utiliser des modèles au relief peu prononcé afin de minimiser les effets de parallaxe.

2.3 Interpolation de points de vue en temps réel

Dans cette partie, nous avons expérimenté l'interpolation de points de vue dont le principe est présenté en section 1.4.4. Nous l'avons tout d'abord implémentée de façon directe, par migration de points, puis nous l'avons accélérée et améliorée à l'aide du placage de texture. Ensuite, nous avons développé une méthode utilisant intégralement la carte graphique, ce qui nous a amené à poser et résoudre le problème d'une façon originale.

2.3.1 Principe

Le principe de l'interpolation de points de vues est de générer les images intermédiaires entre deux points de vues de référence afin d'obtenir une animation continue. Pour cela, nous disposons deux caméra autour de l'objet et enregistrons dans deux textures les images obtenues. Dans notre implémentation, nous avons positionné les centres de projection des deux points de vue de référence de façon à ce que leurs axes optiques passent par le centre de l'objet, lui même centré en $(0, 0, 0)$ dans le repère de la scène. Il ne s'agit aucunement d'une restriction au principe de l'interpolation : cela nous permet simplement de positionner facilement les caméras intermédiaires.

Considérant une seule image de référence, le principe de l'interpolation est de déplacer linéairement en 2D tous ses pixels jusqu'à la position qu'ils occupent selon le deuxième point de vue (figure 2.12). Pour calculer la position d'arrivée d'un pixel donné, il suffit de le reprojeter sur le plan de projection de la caméra du point de vue d'arrivée (cf. section 1.4.3.1). Nous avons donc besoin de la profondeur des pixels de l'image. Cette profondeur est obtenue directement en stockant le tampon de profondeur utilisé pour la capture du point de vue de référence.

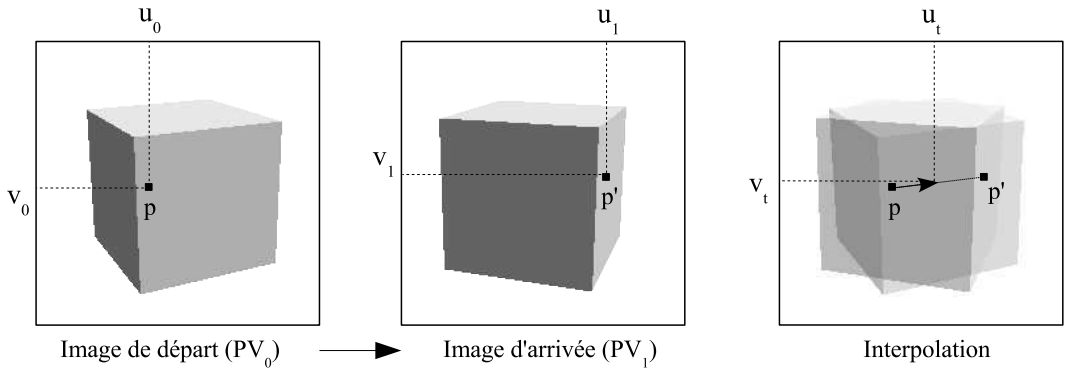


FIG. 2.12 – Principe de l'interpolation de points de vue : déplacement linéaire à l'écran des points de l'image $p(u_{0,}v_0)$ de l'image de départ, à la position (u_1, v_1) qu'il occupe selon le point de vue d'arrivée.

2.3.2 Interpolation avec précalcul du déplacement

Nous avons tout d'abord implémenté directement le déplacement de points. Le principe est de faire évoluer les points de l'image 1 vers leur position selon le point de vue 2 et *en même temps* de faire

évoluer les point des l'image 2 vers leur position selon le point de vue 1. Nous devons donc tout d'abord précalculer le déplacement de chaque pixel des images de références .

2.3.2.1 Réalisation

Soit $p(u, v, z)$ un pixel à la position (u, v) et de profondeur z de l'image de départ, M_d la matrice de transformation de la caméra du point de vue, M_a la matrice de transformation du point de vue d'arrivée, on obtient la position d'arrivée p_a par $p_a = M_a.M_d^{-1}.p$ (reprojection). Le déplacement $\overrightarrow{v_p}$ du point $p(u, v)$ vaut donc $p_a - p$, et est stocké dans une texture de définition égale à la résolution que l'image.

Une fois ce précalcul effectué, le rendu s'effectue en déplaçant chaque pixel de l'image de départ selon un paramètre $t \in [0, 1]$ représentant la position de la caméra virtuelle entre les deux point de vue. Etant donné que plusieurs pixels peuvent se retrouver au même endroit, nous devons utiliser un tampon de profondeur pour gérer le problème de visibilité des pixels. La figure 2.13 montre le résultat de l'interpolation pour chaque point de vue séparément.

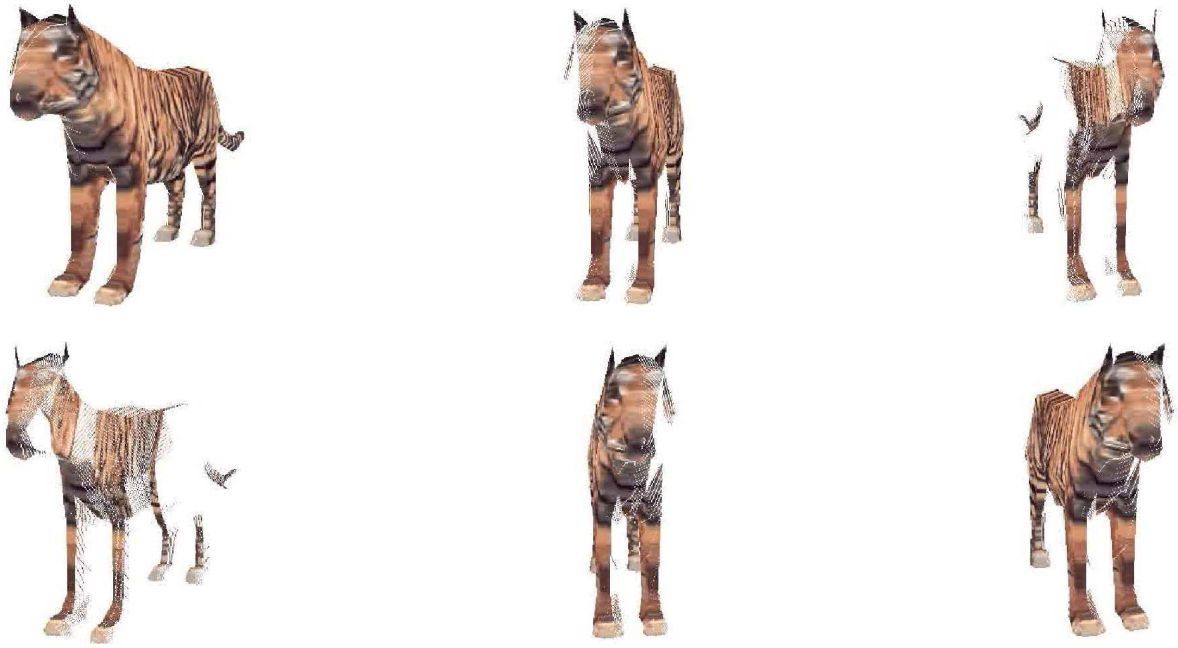


FIG. 2.13 – Interpolation avec précalcul du déplacement. Ligne supérieure : évolution des points du premier point de vue. Ligne inférieure : évolution des points du second point de vue. Colonne de gauche (pt. vue de départ) : $t \approx 0$, colonne du milieu : $t = 0.5$, colonne de droite (pt. vue d'arrivée) : $t \approx 1$.

Pour superposer les deux images, nous devons prendre en compte la profondeur des points déplacés. Etant donné un point $p(u_d, v_d, z_d)$ (dans l'espace de la caméra) d'une image de départ, transformé selon le point de vue d'arrivée en $p_a(u_a, v_a, z_a)$, nous créons le vecteur de déplacement $\overrightarrow{v_p}(\Delta u, \Delta v, \Delta z) = (u_a - u_d, v_a - v_d, z_a - z_d)$. Pour un instant $t \in [0, 1]$ de l'interpolation, le pixel est linéairement interpolé aux coordonnées $p_t = (u_d + t.\Delta u, v_d + t.\Delta v, z_d + t.\Delta z)$. Nous ne le dessinons que si sa profondeur interpolée z_t est inférieure à la profondeur stockée dans le tampon de profondeur.

FIG. 2.14 – Interpolation avec précalcul du déplacement. En haut, sans filtrage bilinéaire. En bas, avec filtrage bilinéaire de l'image.

2.3.2.2 Résultats

La ligne supérieure de la figure 2.14 montre le résultat de cette superposition.

Avec cette méthode, les informations suivantes sont stockées par pixel (pour chaque point de vue de référence) :

1. La couleur du pixel (R,V,B) ainsi que sa profondeur = 3+2 octets (la profondeur est exprimée sur 16 bits, ce qui c'est avéré suffisant dans notre cas).
2. Le vecteur de déplacement du point selon le point de vue opposé : $(\Delta u, \Delta v, \Delta z)$ = 3x2 octets.

De plus, tous les calculs sont effectués par le processeur central. Pour chaque image de l'animation nous devons :

1. Calculer la nouvelle position des points avec un tampon de profondeur pour chaque image de référence.
2. Comparer en chaque pixel de l'écran la profondeur interpolée de chaque point de vue de référence afin de déterminer le plus proche des deux.

Tout ce traitement s'avère coûteux, tant en temps de calcul qu'en espace mémoire. Pour une définition de 512x512 pixels, la mémoire nécessaire est de 2.75 Mo par point de vue. Néanmoins, les résultats visuels sont bons, même avec des points de vues très éloignés. Le taux d'affichage dépend entièrement

de la puissance du processeur central ; mais, étant donné que les calculs à effectuer sont uniquement des interpolations linéaires, le résultat est assez satisfaisant : >15FPS avec un processeur AMD Athlon cadencé à 1.4Ghz.

2.3.3 Méthode accélérée

2.3.3.1 Principe

Afin d'augmenter la rapidité de l'affichage et d'améliorer la qualité visuelle, nous avons tiré parti des capacités de calcul et de filtrage de texture de la carte graphique.

Pour cela, nous affichons un polygone occupant tout l'écran pour lequel la couleur de chaque pixel est calculée par une version modifiée de l'algorithme. En activant l'interpolation bilinéaire de texture, nous lissons ainsi l'image obtenue. De plus, nous pouvons tirer parti du tampon de profondeur de la carte graphique.

2.3.3.2 Problématique

Néanmoins, l'algorithme doit être modifié. En effet, le problème est inversé en ce sens que nous devons, étant donné un pixel de coordonnées (x, y) affiché, trouver son pixel d'origine dans les images de référence. Autrement dit, nous devons, pour un instant t donné, connaître le pixel de chaque image de référence se projetant en (x, y). Cette contrainte provient du fait que le programme exécuté par la carte graphique lors de l'affichage d'un pixel (*pixel shader*) ne peut pas afficher un pixel à une position quelconque de l'écran : le *shader* est un programme qui ne fait que calculer la couleur (et la profondeur) d'un pixel en cours d'affichage. Le pixel calculé est affiché à l'écran à la position actuelle de rastérisation, sans possibilité de choix.

2.3.3.3 Réalisation

Pour pouvoir connaître ces deux pixels d'origine nous associons à chaque point de vue de référence une nouvelle texture qui stocke en chaque texel les coordonnées 2D du point de l'image de référence qui s'y projete à un instant t donné, soit $D(x, y) = (u, v)$, $x, y, u, v \in [0, R]$, R = définition de la texture (carrée). Ce précalcul est effectué par le processeur central, avec un tampon de profondeur logiciel.

Le rendu s'effectue en affichant deux fois le polygone occupant tout l'écran : une passe par point de vue. Pour un passage, l'algorithme, codé dans le *Pixel Shader* de la carte, est le suivant :

1. lors de l'affichage d'un pixel (x, y) on obtient le pixel (u, v) de l'image de référence I s'y projetant à l'aide de la texture de déplacement, soit : $(u, v) = D(x, y)$.
2. La profondeur de ce pixel est stockée dans la texture de profondeur du point de vue de référence, soit : $z = P(u, v)$.
3. Reste à exprimer cette profondeur dans l'espace du point de vue virtuel, soit : $z_v \hat{=} M_v.M^{-1}.(u, v, z)$ (reprojection).
4. Finalement, nous affichons le pixel avec la couleur $I(u, v)$ et la profondeur z_v calculée.

Nous effectuons les mêmes opérations pour le second point de vue. Si les pixels générés par celui-ci sont plus éloignés de l'observateur, ils sont automatiquement éliminés par le test du tampon de profondeur de la carte.

2.3.3.4 Résultats

Le résultat de cette méthode est présenté en figure 2.14, ligne du bas. Comme on peut le voir, certains trous dus aux parties sous échantillonnnées de l'image sont efficacement comblés.

La taille de la structure est de 1,25Mo par point de vue de référence pour une définition de 512x512 pixels. Les opérations effectuées par le processeur central sont la génération des textures de déplacement nécessitant la gestion d'un tampon de profondeur logiciel. Tout le reste est effectué par la carte graphique. Malgré une meilleure qualité visuelle, la réduction de l'espace mémoire nécessaire (de plus de moitié) et un taux d'affichage supérieur (>40 FPS avec une GeForceFx et un Athlon 1.4Ghz), la partie CPU de l'algorithme correspond au goulot d'étranglement.

2.3.4 Interpolation avec calcul au vol du déplacement

Pour nous affranchir de cette dépendance vis à vis du processeur central, nous avons alors essayé de n'avoir aucune étape de calcul effectuée par le CPU, c'est à dire de délocaliser tous les calculs dans le *pixel shader* de la carte graphique.

2.3.4.1 Problématique

Le problème est le même que celui exposé dans la section précédente : le *pixel shader* ne nous permet pas d'écrire où l'on veut à l'écran. La problématique, du point de vue du shader, peut s'exprimer ainsi : "Trouver le meilleur texel à retourner parmi les deux textures de référence, sachant que la position actuelle de rastérisation à l'écran est p(x,y)."

2.3.4.2 Réalisation

La figure 2.15 détaille le problème : il faut déterminer la couleur du pixel p du point de vue virtuel. Au début de l'interpolation ($t = 0$), la couleur de p est issue du point p_0 de la surface de la scène. A la fin de l'interpolation, ($t = 1$), la couleur de p est issue de p_1. p_0 et p_1 sont obtenus à partir du tampon de profondeur de chaque point de vue de référence par projection inverse. Pour un instant intermédiaire de l'interpolation, la couleur de p est issue de l'intersection de son rayon optique avec la surface de la scène, c'est à dire du point p_R, sur la partie droite de la figure 2.15.

Dans notre algorithme, nous considérons que l'intersection entre l'axe optique de p et la surface de la scène décrit un segment 3D dans l'espace de la scène, segment qui débute en p_0 et se termine en p_1. Sur la partie droite de la figure, nous considérons donc que le point d'intersection est p_V au lieu de p_R .

Notre algorithme consiste alors à :

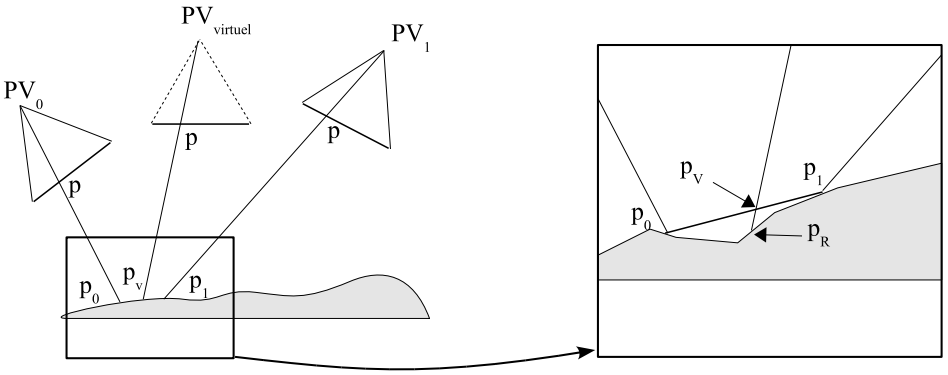


FIG. 2.15 – Interpolation avec calcul au vol du déplacement des points.

1. interpoler linéairement p_V entre p_0 et p_1 mais dans l'espace de chaque image de référence au lieu de la scène (pour des raisons de complexité du calcul, la taille d'un shader étant limitée). Pour cela, comme l'illustre la figure 2.16, il faut d'abord projeter p_1 sur le plan de projection du point de vue 0 (PV_0) et p_0 sur PV_1. On obtient alors les points 2D $p_{1,0}$ et $p_{0,1}$ dans l'espace des l'images des PV_0 et PV_1 (respectivement).

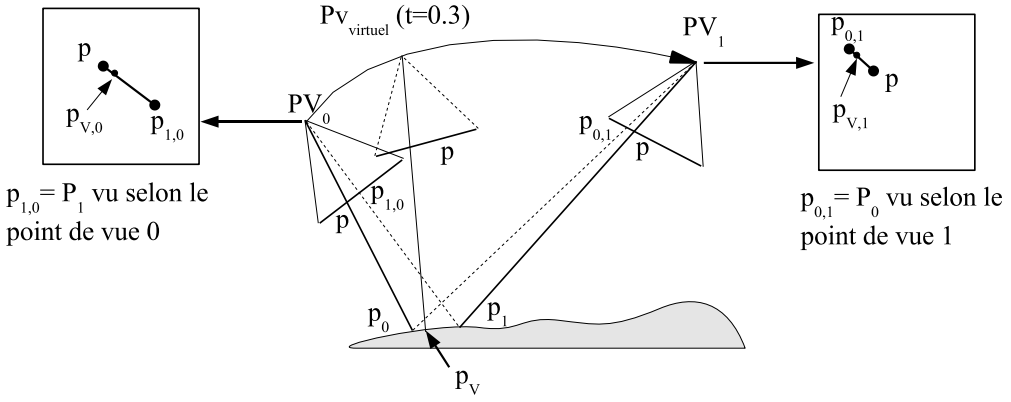
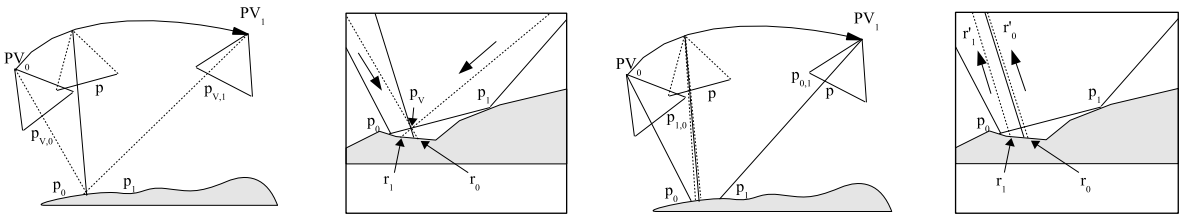


FIG. 2.16 – Principe de l'interpolation.

2. Ensuite, nous calculons les points $p_{V,0}$ et $p_{V,1}$ par interpolation linéaire en 2D sur les segments $[p, p_{1,0}]$ et $[p_{0,1}, p]$, soit : $p_{V,0} = p + t.(p_{1,0} - p)$ et $p_{V,1} = p_{0,1} + t.(p - p_{0,1})$.
3. Finalement, pour déterminer le meilleur point entre $p_{V,0}$ et $p_{V,1}$, nous procédons comme suit :
   (a) Nous reprojetons $p_{V,0}$ et $p_{V,1}$ dans l'espace de la scène (figure 2.17(a)) : on obtient les points r_0 et r_1.
   (b) Le meilleur point de vue est celui dont le point r_i est le plus proche de p_V. En pratique, pour calculer le plus proche, nous projetons r_0 et r_1 sur le plan de la caméra virtuelle (figure 2.17(b)) : on obtient les points $r_0^{'}$ et $r_1^{'}$ et nous comparons en 2D la distance $(p, r_0^{'})$ avec $(p, r_1^{'})$.



(a) Reprojection des points 2D interpolés dans l'espace de la scène.

(b) Projection des points 3D sur le plan de la caméra virtuelle.

FIG. 2.17 – Détermination du meilleur point de vue.

2.3.4.3 Résultats

Les premiers résultats montrèrent des problèmes au niveau des bordures de l'objet, mais une interpolation correcte dans les parties de l'image se superposant entre le point de vue 0 et le point de vue 1. La figure 2.18, montre le résultat de l'interpolation pour les points valides de l'image.

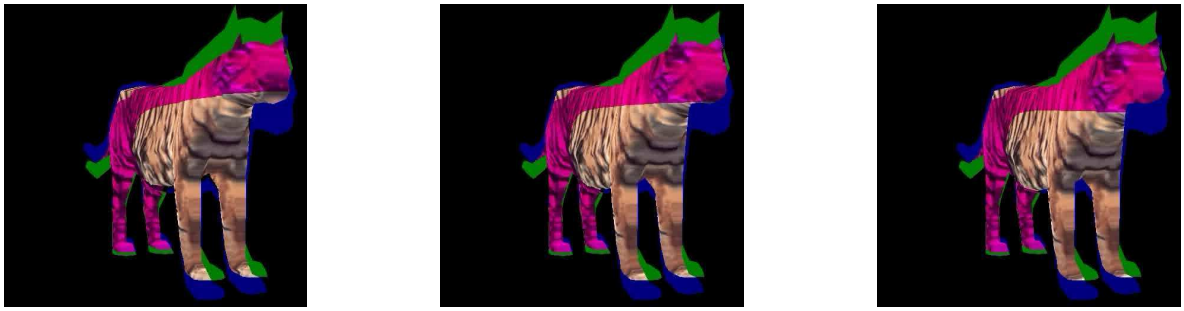


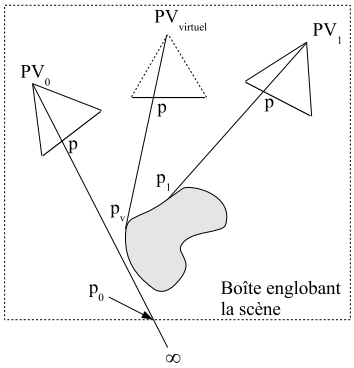
FIG. 2.18 – Interpolation avec calcul "au vol" du déplacement dans les parties de superposant des images de référence. De gauche à droite : $t = 0.0$, $t = 0.5$ et $t = 1.0$.

Comme le montre la figure 2.19, le problème des pixels en bordure de l'image est dû au fait que, pour ces pixels, il n'existe pas de point p_0 ou p_1.

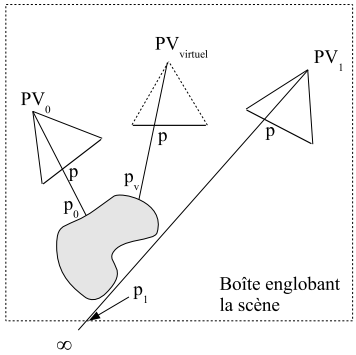
Pour résoudre ce problème, nous disposons une boîte englobante autour de la scène. Celle-ci permet de générer un point dans tous les cas. Néanmoins, nous devons prendre quelques précautions : dans le cas (a) de la figure 2.19, nous interpolons p_V entre p_0 et p_1 dans la texture du PV_0 mais nous ignorons le PV_1 car il ne peut pas donner de points pertinents dans ce cas. Nous procédons symétriquement pour le second cas, figure (b).

La figure 2.20 montre le résultat de l'interpolation avec cette amélioration. Les zones vertes correspondent au cas (b) de la figure 2.19, et les zones bleues au cas (a). Comme on peut le distinguer sur la queue et les pattes avant du tigre sur la figure, à $t = 0.5$, les pixels pour lesquels on ne dispose ni de p_0 ni de p_1 (les pixels noirs) ne sont pas interpolés, ce qui donne des trous.

Ainsi, pour que cette méthode fonctionne il faut que les points de vue de départ et d'arrivée ne soient pas trop éloignés afin d'augmenter les zones de recouvrement. L'avantage est que le rendu est très rapide (>90 FPS avec une Radeon 9700), totalement indépendant de la complexité de la scène et



(a) Problème des pixels contenus dans l'image d'arrivée mais pas dans l'image de départ.



(b) Problème des pixels contenus dans l'image de départ mais pas dans l'image de d'arrivée.

FIG. 2.19 – Utilisation d'une boîte englobante pour résoudre le problème des points extérieurs à l'objet.

FIG. 2.20 – Interpolation avec calcul "au vol" du déplacement. De gauche à droite : $t = 0.0$, $t = 0.5$ et $t = 1.0$.

demande peu de mémoire : une texture de couleurs par point de vue, car le tampon de profondeurs nécessaire en chaque point de vue de référence est stocké dans la composante alpha de la texture. Ainsi, un point de vue occupe environ 700Ko (avec compression DXT1) en mémoire vidéo.

Nous avons également testé la méthode avec un modèle de 871000 faces. Nous avons observé des déformations au sein des parties de l'image recouvertes par les deux points de vues. Ces déformations sont dûes au fait que la surface entre les points p_0 et p_1 est supposée plate, ce qui n'est que rarement le cas. La ligne supérieure de la figure 2.21 montre les points réellement balayés par le rayon optique issu d'un pixel de la caméra virtuelle, et la ligne du bas montre les points que nous balayons avec notre méthode. L'approximation est bonne dans certains cas : la premiere colonne d'images par exemple. Mais dans le cas général (les autre images), notre approximation linéaire est fausse.

Ceci nous empêche d'utiliser cette méthode à grande échelle, c'est à dire pour un relief quelconque et des points de vue de référence éloignés.

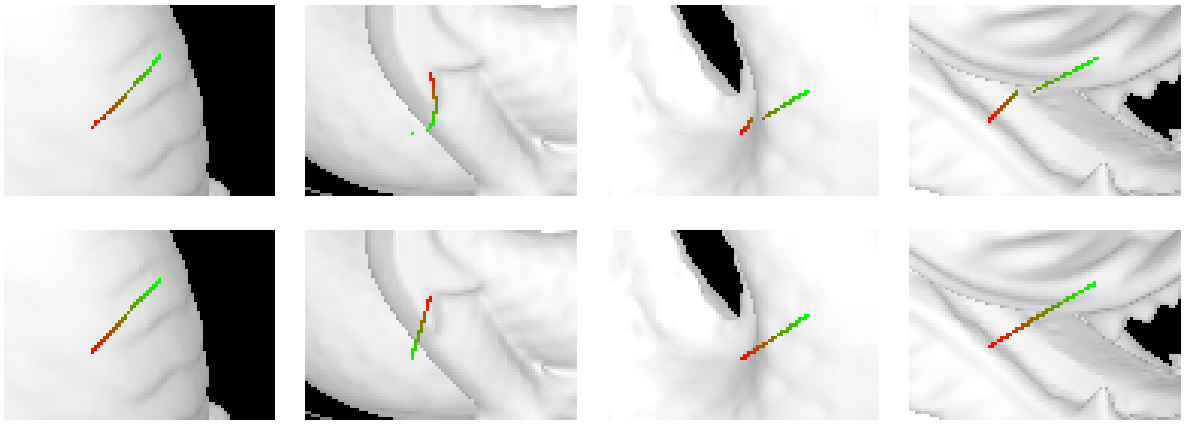


FIG. 2.21 – Problème dus à l'interpolation linéaire : dans les cas complexes, l'approximation par un segment de l'évolution des points est impossible. En haut, les points de la scène balayés par le rayon optique issu d'un pixel de la caméra virtuelle ; en bas les points calculés. Les points rouges correspondent à $t = 0$ et les points verts à $t = 1$. Le modèle utilisé est le Dragon numérisé du Laboratoire d'Informatique Graphique de Stanford.

2.3.4.4 Discussion

L'interpolation de points de vue par pixel s'avère difficile à employer dans le cadre d'une application complète de réalité virtuelle. Dans l'implémentation la plus convaincante visuellement (section 2.3.2), le problème est le trop grand temps de calcul nécessaire. Dans l'implémentation accélérée que nous avons développée (section précédente), nous n'avons pas réussi à prendre en compte l'intégralité des pixels si les points de vue de références sont trop éloignés (voir la figure 2.20). Dans ce cas, la restriction la plus importante que nous avons est que les points de vues de départ et d'arrivée doivent être très proches pour que la méthode fonctionne correctement. Malgré tout, le principe du calcul par pixel s'avère extrêmement rapide, en dépit des lourdes opérations de calcul matriciel effectuées pour chaque pixel affiché.

2.3.5 Interpolation par blocs de pixels

2.3.5.1 Principe

Le déplacement à l'écran des pixels dépend de leur profondeur. Partant du constat que beaucoup de pixels suivent approximativement la même trajectoire, l'idée développée dans cette partie est de les regrouper afin de les traiter par blocs plutôt que séparément. Ceci permet de n'appliquer les calculs de déplacement qu'aux blocs et de réduire ainsi le temps de calcul, l'affichage des blocs étant confié à la carte graphique.

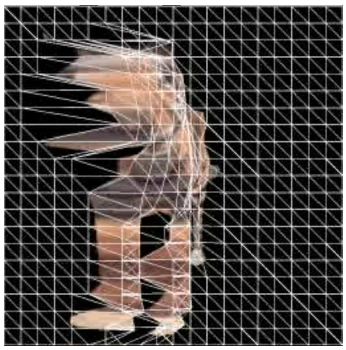
2.3.5.2 Maillage régulier

Tout d'abord, nous avons regroupé les points par une subdivision régulière des images de référence. Pour cela, pour une image, nous créons un maillage 3D basé sur son tampon de profondeur. Les sommets du maillage sont régulièrement positionnés en 2D par rapport à l'image puis leur troisième

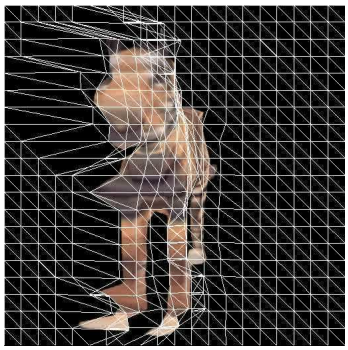
composante est extraite du tampon de profondeur. Nous obtenons ainsi (figure 2.22) un maillage de triangles

FIG. 2.22 – Maillage régulier.

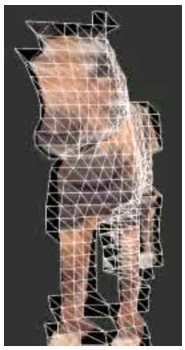
basé sur un point de vue de référence et dont chaque sommet est défini dans l'espace de la caméra par ses coordonnées (u, v, z). Le principe est alors de déplacer les sommets de ce maillage vers la position qu'ils occupent à l'écran selon le second point de vue, de façon exactement similaire au déplacement de points de la partie précédente. Les triangles du maillage sont texturés avec l'image de référence ; leurs coordonnées de texture sont directement issues de leur position dans l'image, à un changement de repère près : les coordonnées de texture (u, v) sont définies entre 0 et 1, tandis que les coordonnées (x, y) des sommets sont basées sur le repère du plan de projection de la caméra dont l'origine est au milieu de l'image. La figure 2.23(a) montre le résultat de l'interpolation avec cette méthode : les sommets extérieurs à l'objet ne se déplaçant pas, les triangles du maillage formés de tels points se voient étirés (effet de *skinning* ou de *rubber sheet*). Pour les triangles disposés totalement à l'intérieur de l'objet, par contre, le résultat est satisfaisant.



(a) Artefacts visuels (*Skinning*) apparaissant sur les triangles en bordure du modèle



(b) Correction partielle du problème en affectant une profondeur virtuelle aux sommets extérieurs



(c) Suppression des triangles vides

FIG. 2.23 – Déplacement des sommets du maillage de l'image de départ vers la position qu'ils occupent à l'écran selon le second point de vue.

Pour améliorer le rendu et tenter de résoudre ce problème, nous avons attribué une profondeur virtuelle à tous ces points extérieurs de façon à ce qu'ils se déplacent aussi et évitent ces étirements. La profondeur que nous attribuons aux sommets extérieurs d'un triangle est la moyenne des profondeurs des autres sommets (voir figure 2.24). Si il y a deux sommets extérieurs (cas le plus défavorable), nous leurs affectons la profondeur du seul sommet intérieur.

Le résultat est visible sur les sous-figures (b) et (c) : les effets d'étirement sont réduits, mais pas éliminés. En particulier, comme on peut le voir entre les pattes du tigre, les triangles de deux parties distinctes (de profondeur "très" différente) de l'objet partageant un ou plusieurs sommets se voient liés et étirés.

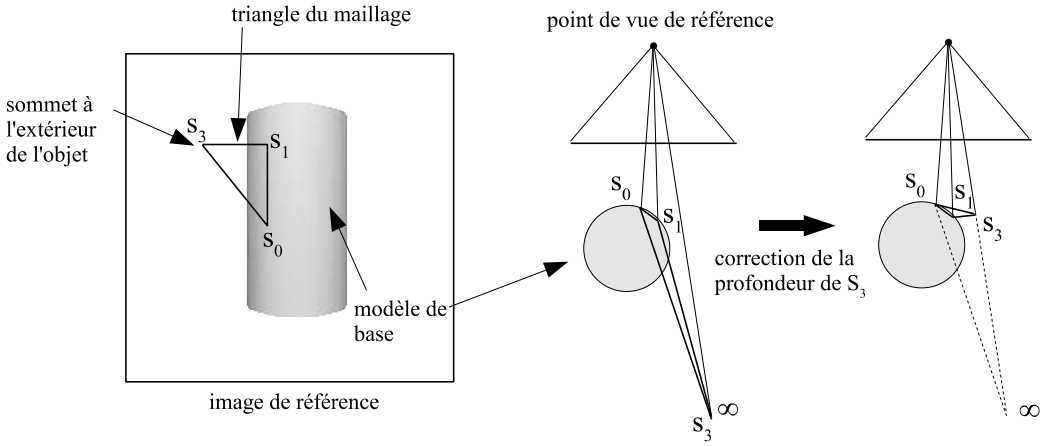


FIG. 2.24 – Attribution d'une profondeur virtuelle aux points extérieurs du maillage.

Ceci est dû à la façon dont nous avons créé le maillage : un triangle est défini par trois références aux trois sommets le constituant. Il faudrait donc, dans ce cas, séparer les sommets qui posent problème.

2.3.5.3 Maillage adaptatif

Les bordures de l'objet étant le problème central pour la création du maillage, nous avons expérimenté une autre méthode permettant d'éviter de créer des sommets extérieurs à l'objet.

Comme le montre la figure 2.25, nous segmentons tout d'abord l'image de profondeurs en un nombre fixé de couches, puis nous subdivisons les triangles du maillage régulier aux bordures de ces couches. La nombre de couches de profondeurs n'est pas choisi au hasard, il est choisi de façon à créer des zones partitionnant les niveaux de profondeurs générant les artéfacts d'étirement. Par exemple, sur la figure 2.25, si l'on créait un triangle couvrant une partie de la queue et une partie de la cuisse du tigre, ce triangle génèrerait du skinning car il y a une disparité de profondeur entre ces deux parties. Notre segmentation est donc choisie de façon à isoler ces parties de l'image. Bien sûr, cette segmentation arbitraire subdivise inutilement certaines parties de l'image, mais dans un premier temps cela permet de vérifier la validité de la méthode.

La subdivision des triangles se fait en parcourant pixel par pixel chacune des trois arêtes des triangles avec l'algorithme de tracé de droites de Bresenham jusqu'à passer d'un niveau de profondeur à un autre. Si nous ne trouvons aucune transition, nous ne subdivisons pas le triangle. Dans le cas contraire, nous obtenons principalement le cas exposé en figure 2.25 où deux arêtes sont coupées par une bordure de zone. Dans ce cas, nous relions les points de transition trouvés et nous subdivisons le quadrilatère Q en trois triangles T_0, T_1 et T_2 sur la base d'un nouveau point p positionné au milieu de l'arête sans transition. Les critères d'arrêt de la subdivision sont un niveau de récursion maximum et une taille minimale pour les arêtes subdivisées. La figure 2.25 montre le maillage obtenu avec cette méthode sur l'image segmentée sur 6 niveaux.

Nous avons constaté que cet algorithme génère des trous, invisibles quand on subdivise l'image, mais apparents dès qu'on affiche le maillage sous un autre angle du vue. La figure 2.26 illustre cet artéfact : les trous sont dus à la différence de profondeur entre le point de subdivision (p_1) et le point de même

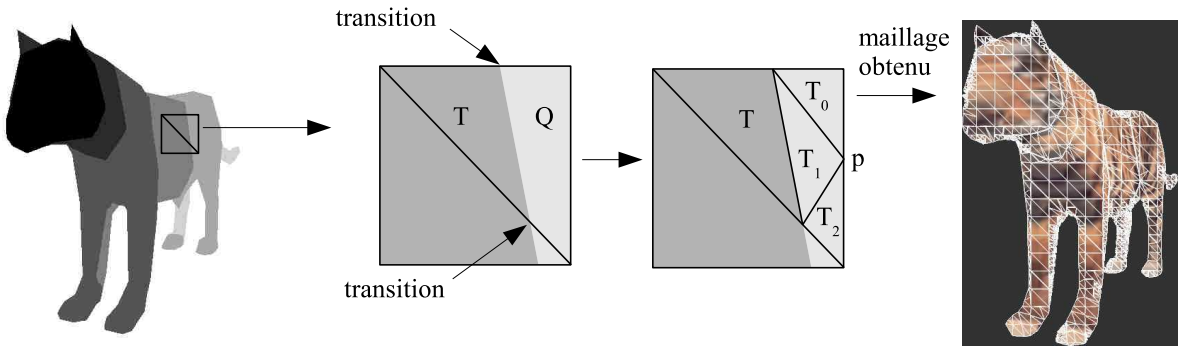


FIG. 2.25 – Partitionnement de l'image en un nombre fixé de niveaux de profondeurs servant à la subdivision du maillage.

coordonnées de texture, mais positionné sur l'arête du triangle adjacent (p'_1). Ce problème est courant dans les techniques utilisant la subdivision de triangles au sein de maillages : le point p_1 est un *T-vertex* (litt. sommet de type T) [AMH02].

Pour corriger ce problème, nous devons subdiviser le triangle adjacent. Nous avons choisi d'effectuer cette subdivision de façon à minimiser le nombre de triangles créés.

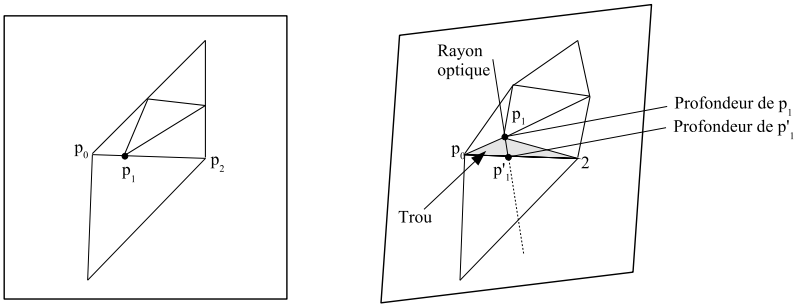
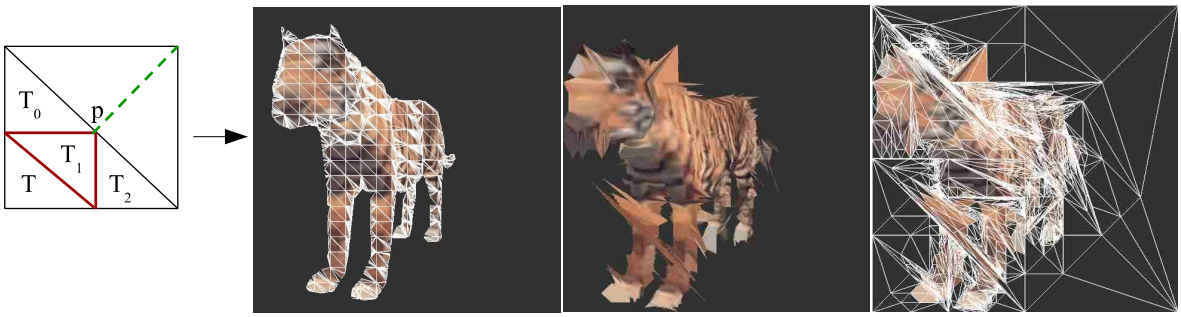


FIG. 2.26 – Problème de la subdivision d'arête.

Comme le montre la figure 2.27, nous séparons en deux le triangle adjacent par une arête joignant le point de transition au sommet opposé. Malheureusement, cela génère des sommets hors de l'objet, et nous retrouvons les problèmes d'étirement.

Sur la figure 2.27(b), nous avons retiré du maillage ces triangles n'étant pas intégralement contenu dans l'image de l'objet. Ceci crée des trous, visibles par exemple sur les pattes arrières (voir également l'image de droite de la figure 2.28).

Les figures 2.27(c) et (d) montrent le résultat de cette méthode appliquée à l'ensemble de l'image. C'est à dire que nous ne partons pas du maillage régulier mais d'un seul carré composé de deux triangles occupant toute l'image et appliquons l'algorithme.



(a) (b) (c) (d)

FIG. 2.27 – Application de la subdivision d'arêtes avec prise en compte des *T-vertices* (image (a)). Les images (c) et (d) correspondent à la subdivision appliquée à l'ensemble de l'image plutôt qu'au maillage régulier (b).

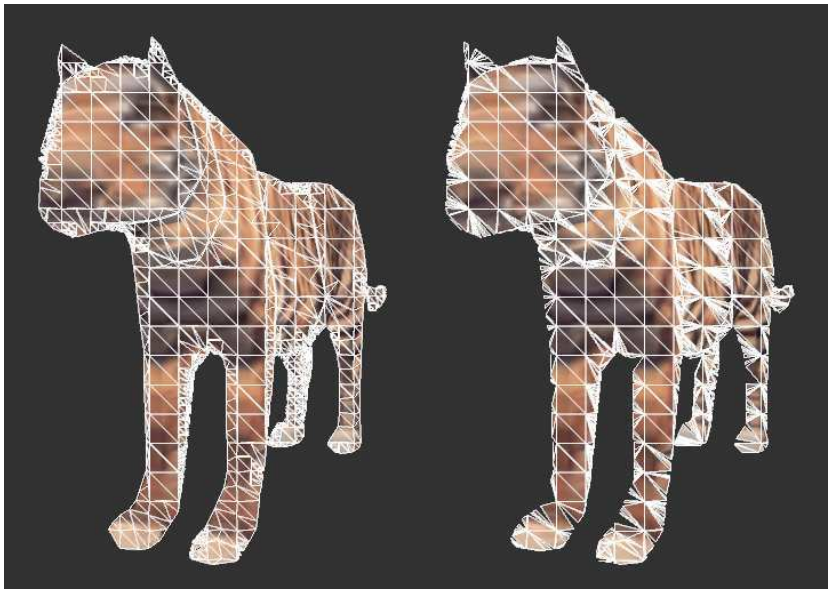


FIG. 2.28 – Comparaison des deux algorithmes de subdivision. A gauche, la première version générant des *T-vertices.* A droite, la version corrigée.

2.3.5.4 Discussion

La construction d'un maillage basé sur une image de profondeurs s'avère être une opération compliquée. Dans les travaux de Sillion *et al.* [SDB97][DSSD99], un maillage de l'image de fond d'une scène urbaine est utilisé pour remplacer la géométrie distante de l'observateur. Le maillage régulier est coupé aux endroits générant de trop grand effets d'étirement. Néanmoins leur méthode est adaptée à une scène distante de l'observateur, pour laquelle les effets d'étirement subsistants ne sont pas visibles.

Dans les travaux récents de Pajarola *et al.* [PSM04], le problème de la création d'un maillage polygonal approximant une image de profondeur est analysé en détails et ils proposent une méthode

(DMesh) permettant de détecter les disparités de profondeurs de l'image, de créer un maillage polygonal adapté et de combiner en temps réel plusieurs de ces maillages pour couvrir tous les angles de vue. Le maillage de l'image de profondeur peut également s'effectuer en temps réel et adapté au point de vue, bien qu'au prix d'une baisse de performances (150 FPS $\Longrightarrow$ 6 FPS sur un Pentium4 1.5Ghz et une carte graphique GeForce4 Ti4600).

2.4 Conclusion

Dans ce chapitre, nous avons présenté différentes méthodes d'interpolation d'images en temps réel.

Concernant l'interpolation d'images de références en fonction du point de vue, les principales améliorations que nous avons apportées par rapport au *placage de texture dépendant du point de vue* (VDTM) sont :

- la prise en compte d'un éclairement variable de l'image, entrainant une qualité visuelle accrue. Le VDTM est basé à l'origine sur des photographies et donne de mauvais résultats quand on l'applique à des images de synthèse, moins bruitées de que des images réelles. Notre interpolation de cartes de normales améliore la qualité globale du résultat.
- l'automatisation complète du processus d'acquisition : points de vue de référence et maillage simplifié sont obtenus automatiquement. Le VDTM nécessite un lourd travail de construction de maillage à partir des images de référence.
- la simplicité de l'implémentation, notamment, aucune paramétrisation du maillage simplifié remplaçant le modèle complexe n'est nécessaire, ce qui n'est pas le cas du VDTM.

Néanmoins, nous avons mis en évidence les problèmes inhérents à cette approche basée sur un mélange pondéré de couleurs : les effets de parallaxe (*ghosting, blurring*) dégradent l'image, et il est impossible de les traiter avec de simples images de couleurs. Malgré de très bons taux d'affichage, cette approche nécessite un trop grand nombre de points de vue de référence pour donner une qualité visuelle acceptable.

Concernant l'*interpolation de point de vue,* nous l'avons implémentée en temps réel, ce qui n'a pas été fait à notre connaissance. Notre méthode la plus aboutie, l'interpolation avec calcul au vol du déplacement, s'effectue intégralement à l'aide du GPU, ce qui est impossible à effectuer avec l'interpolation de points de vue conventionnelle : la déformation 3D d'image se prête très mal à ce type d'adaptation. La principale contribution de notre méthode est de proposer une solution partielle, mais originale, à la déformation d'image.

Les travaux que nous avons effectué sur le déplacement par blocs de points et la création de maillage géométrique basé sur une image de profondeur n'ont pas donné de résultats probants. Les méthodes de création de maillage que nous avons implémentées génèrent trop de polygones pour la méthode garde un intérêt. La création d'un maillage sur la base d'une image de profondeur est une opération complexe qui nécessiterait une thèse à elle seule.

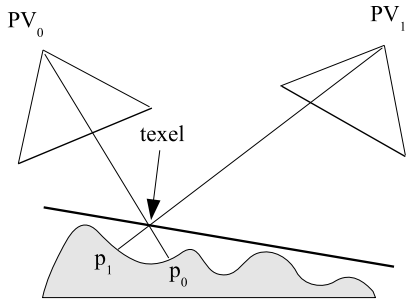
3

Textures directionnelles 4D

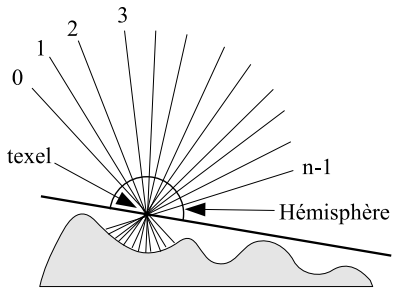
3.1 Introduction

Le principe du *placage de texture dépendant du point de vue* (section 1.4.1.7) est de mélanger et plaquer des images de référence d'une scène complexe sur un modèle géométrique approché de celle-ci. Comme nous l'avons montré en section 2.2, les effets de parallaxe produisent des artéfacts de flou dans la texture générée.

Dans ce chapitre, nous avons développé une approche brute pour résoudre ces problèmes. Plutôt que de se baser sur quelques images de référence pour créer une texture intermédiaire, nous créons une texture dont chaque texel stocke la visibilité de la scène originelle selon tous les angles d'observation possibles (figure 3.1). Ainsi, chaque élément de la texture est référencé par ses coordonnées de texture (u, v) et un angle d'observation (θ, φ), soit quatre dimensions.



(a) *Placage de texture dépendant du point de vue*
=> 2 couleurs de référence par texel (dans ce cas), correspondant aux deux points de vue de référence.



(b) *Texture dépendante du point de vue*
=> *n* couleurs de référence précalculées par texel, pour *n* orientations différentes réparties sur un hémisphère centré sur le texel.

FIG. 3.1 – Comparaison ne notre méthode avec le placage de texture dépendant du point de vue.

La méthode s'apparente au *lancer de rayons virtuels* des *BTF* (Bi-directional Texture Functions) de Dischler [Dis98] ainsi qu'au *déplacement de surface dépendant du point de vue* [WWT+03], car des informations géométriques sur le modèle de base sont stockées en chaque texel.

Motivations La principale motivation de ces travaux est d'obtenir une image du modèle originel selon un angle de vue quelconque en un temps ne dépendant pas de sa complexité géométrique mais uniquement du temps de génération des textures (de définition fixe), et ceci afin d'obtenir un rendu en temps réel.

Le second objectif a été la prise en compte de modèles d'illumination locaux coûteux en temps de calculs, comme les BRDF acquises physiquement ou l'auto-ombrage. Ainsi, nos textures directionnelles varient en fonction de la position d'observation mais aussi en fonction de l'éclairement.

Un autre objectif qui a motivé ces travaux est la possibilité d'utiliser l'interpolation bilinéaire de texels des cartes graphiques pour corriger les imperfections visuelles sur les images résultantes. Comme nous l'avons montré en section 2.3.3, cette technique permet de "boucher" une partie des trous apparaissant dans les textures générées texel par texel, et d'augmenter ainsi la qualité du rendu avec un impact négligeable sur les performances.

Principe D'un point de vue général, notre méthode consiste à remplacer une partie de la scène par son image vue à travers un polygone texturé positionné au sein de la scène (cf. figure 3.2), de façon à accélérer le rendu des parties complexes de la scène qu'elle remplace, et est en cela similaire aux travaux d'Aliaga [Ali96], et différente des *BTF*. La différence principale avec la méthode d'Aliaga est que notre texture s'adapte au point de vue de façon à prendre en compte les effets de parallaxe.

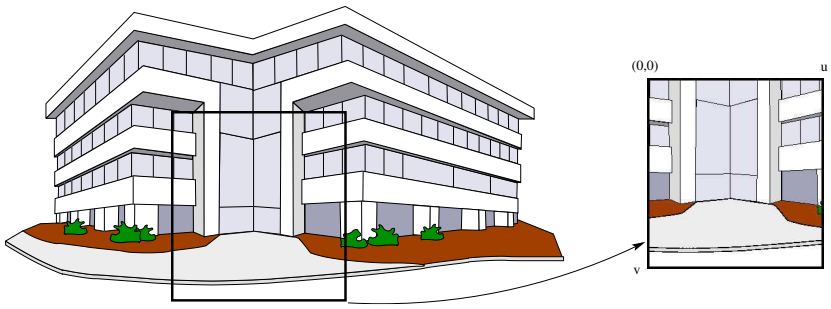


FIG. 3.2 – Remplacement d'une partie de la scène par un polygone texturé (un imposteur).

Une texture directionnelle est donc plaquée sur un polygone positionné autour du modèle d'origine. Dans nos travaux, nous avons utilisé une boîte englobant le modèle complexe de façon à le remplacer en totalité. Le modèle complexe est utilisé lors de la phase de précalculs pour construire les textures directionnelles, puis il est supprimé et remplacé pour le rendu par l'affichage de la boîte englobante texturée de la scène.

Le principe de la méthode est donc d'instancier et de plaquer une texture par face visible, l'ensemble prenant l'apparence du modèle d'origine.

Le rendu se décompose en deux étapes principales : le calcul de visibilité des texels et le calcul d'éclairement des texels.

▷ Le calcul de visibilité consiste à déterminer le relief de la scène pour une direction d'observation donnée. Ce relief est stocké en chaque texel, pour un ensemble de directions d'observations fixé.

▷ Le calcul d'éclairement consiste à attribuer une couleur à chaque texel de la texture en fonction de la position des sources de lumière ponctuelles et de l'observateur.

3.2 Textures directionnelles brutes

Nous avons tout d'abord implémenté la méthode de façon brute : le précalcul de chaque texture directionnelle appliquée à une face de la boîte englobante se fait en lançant n rayons par texel, n correspondant au nombre de directions d'observation que nous souhaitons.

3.2.1 Création d'une texture

Considérant une face f de la boîte englobante (figure 3.3), la phase de précalcul de la méthode consiste à stocker en chaque point p de f l'ensemble des points d'intersection des rayons passant par p, orientés dans toutes les directions, avec le modèle de base.

Pour cela, nous discrétisons la face avec une résolution arbitraire, correspondant à la résolution de la texture que nous plaquerons par la suite. Nous discrétisons aussi l'ensemble des directions qui sont réparties sur un hémisphère orienté selon la normale de la face.

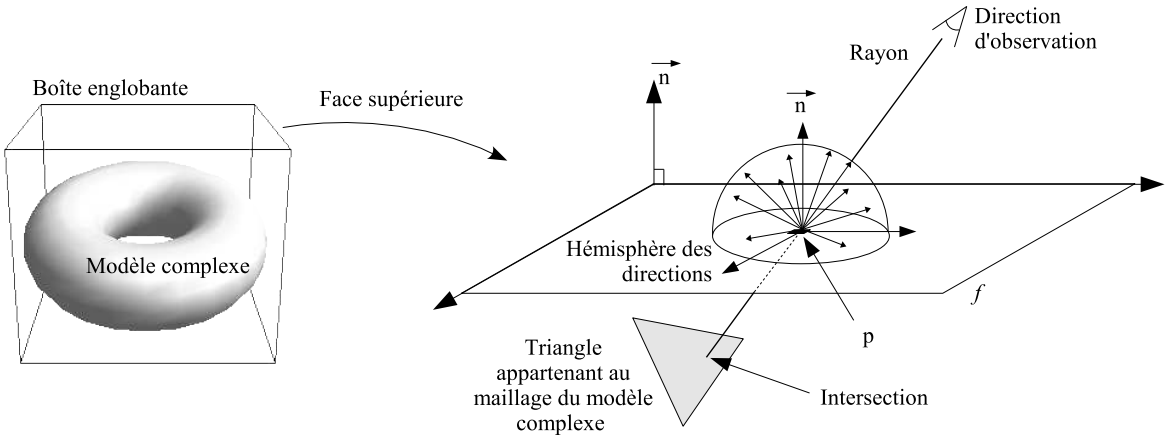


FIG. 3.3 – Précalcul : acquisition des informations géométriques pour chaque point des faces de la boîte englobante par lancer de rayons.

La phase de précalculs consiste à effecter un lancer de rayons pour chaque direction de chaque texel de chaque face (figure 3.4). Pour chaque texel, nous obtenons :

- la distance h entre le point p et le point d'intersection i.
- la couleur de la face intersectée.
- la normale $\overrightarrow{n}$ au point d'intersection

Ces données sont ensuite quantifiées de façon à occuper moins de place en mémoire.

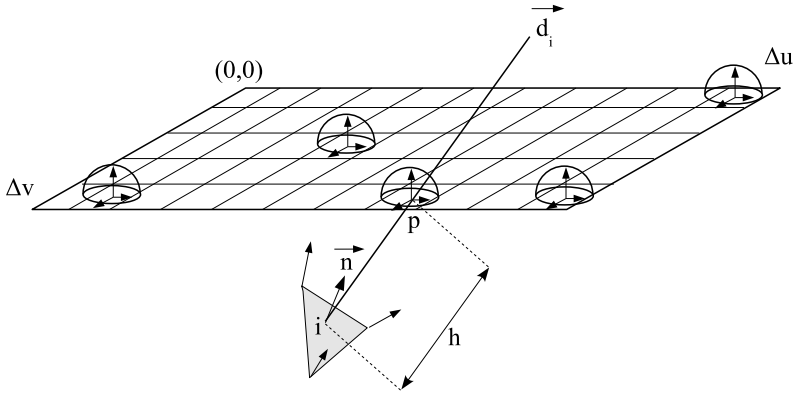


FIG. 3.4 – Précalcul : lancer d'un rayon par point, pour chaque direction d_i de l'hémisphère.

3.2.2 Structure des données

La structure de données représentant une texture directionnelle doit être choisie avec soin car nous avons deux contraintes : d'une part une texture directionnelle doit s'instancier très rapidement et d'autre part, elle doit occuper le moins d'espace mémoire possible.

Dans le premier cas, le point critique de l'instanciation est le calcul de localisation effectué en chaque texel lors du rendu (voir section 3.2.3). La structure, qui doit permettre des calculs rapides, est décrite en section 3.2.2.1.

Concernant la taille des données, nous décrivons en section 3.2.2.2 la méthode de quantifications des données que nous avons utilisée.

3.2.2.1 Discrétisation de l'hémisphère des directions

L'ensemble des directions utilisées pour le lancer de rayons est discrétisé régulièrement en coordonnées sphériques. Cette discrétisation donne un tableau de n vecteurs distribués sur l'hémisphère. Les vecteurs sont exprimés en coordonnées sphériques et répartis en n_φ tranches et n_θ secteurs. Sur la figure 3.5, $n_\varphi = 3$ et $n_\theta = 3$; ce qui donne au total $n_\theta * (n_\varphi - 1) + 1 = 7$ directions (la direction normale au texel n'est stockée qu'une seule fois). Il y a donc n_θ azimuths, définis dans l'intervalle $[0, ..., 2\pi[$ et n_φ angles polaires, définis dans l'intervalle $[0, ..., \frac{\pi}{2}[$. Cette décomposition permet de retrouver directement le secteur d'azimuth et d'angle polaire auquel appartient une direction quelconque donnée.[30]

[30]Le secteur s_a auquel appartient l'azimuth d'un vecteur $\overrightarrow{v}$ est obtenu par $s_a = \left\lceil \frac{\theta_{\overrightarrow{v}}}{\delta\theta} \right\rceil$, avec $\delta\theta = \frac{2\pi}{n_\theta}$. De la même façon, le secteur s_p de l'angle polaire de $\overrightarrow{v}$ est obtenu par $s_p = \left\lceil \frac{\varphi_{\overrightarrow{v}}}{\delta\varphi} \right\rceil$, avec $\delta\varphi = \frac{\pi/2}{n_\varphi}$.

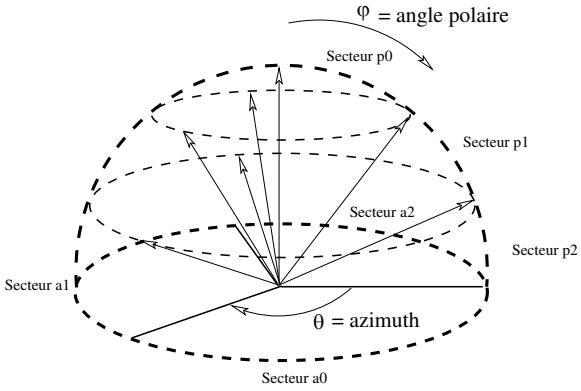


FIG. 3.5 – Discrétisation de l'hémisphère des directions.

3.2.2.2 Quantifications

Afin de limiter la taille de la structure, l'ensemble des couleurs et l'ensemble des normales obtenues lors du lancer de rayons sont quantifiés de façon à être indexés. C'est à dire que pour chaque texel, dans une direction donnée, on ne stocke qu'un index vers la table des couleurs quantifiées et un index vers la table des normales quantifiées, ce qui réduit ainsi la taille de la structure (figure 3.6).

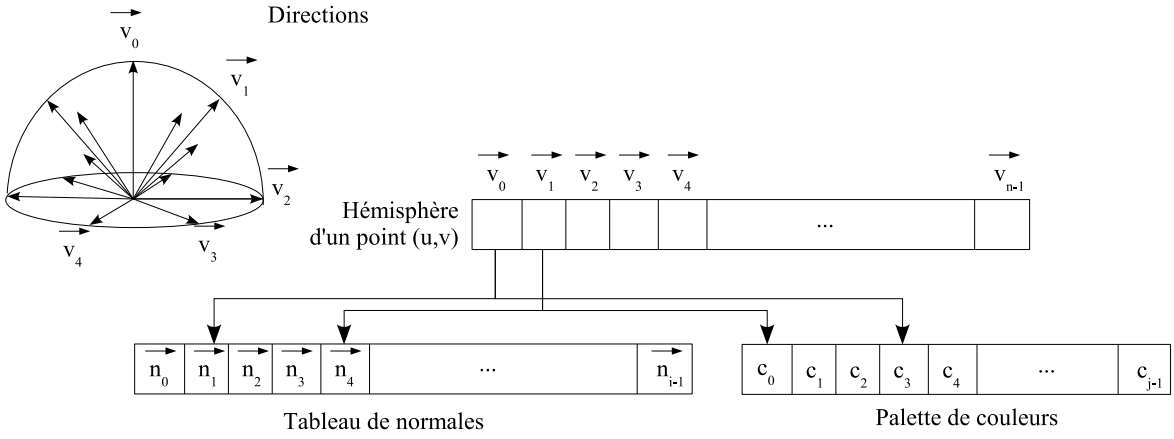


FIG. 3.6 – Quantification des normales et des couleurs.

Couleurs L'ensemble E_c des couleurs à quantifier est obtenu lors du lancer de rayons. Cet ensemble de couleurs est réduit à j couleurs distinctes selon l'algorithme suivant :

- le tableau E_q (de taille j) des couleurs quantifiées est initialisé avec les j couleurs les plus représentées parmi E_c.

- pour les couleurs les moins représentées, on recherche la couleur la plus proche (dans l'espace RVB) dans E_q et on moyenne les deux.

Cet algorithme n'est pas optimal, et même très mauvais dans le cas général, néanmoins si la scène comporte peu de couleurs différentes, ce qui est le cas dans nos tests, cet algorithme appliqué à plus de 256 couleurs quantifiées (index sur 1 octet) suffit largement. De nombreuses méthodes, entre autres [Hec82, Cla96, Her96], permettent de réduire le nombre de couleurs en créant des palettes optimales.

Normales L'ensemble des normales est quantifié selon une méthode très simple : l'ensemble des directions de la sphère unité est discrétisé comme précédemment mais sur une sphère au lieu d'un hémisphère, puis on recherche pour chacune des normales la direction quantifiée la plus proche.

Distance La distance h (figure 3.4) entre le point de la surface et le point d'intersection est utilisée lors du calcul de l'éclairement. Cette valeur est exprimée par rapport à la taille de la plus grande diagonale de la boîte englobante, qui correspond à la valeur maximale possible. h est donc exprimée comme proportion de cette diagonale, et stockée sur un octet.

Finalement, nous obtenons pour chaque texel un tableau (d'orientations) de taille n stockant en chaque élément (figure 3.6) :

$$\begin{cases} \text{l'indice de la plus proche couleur quantifiée} & \text{1 octet} \\ \text{l'indice de la plus proche normale quantifiée} & \text{2 octets} \\ \text{la distance de l'intersection} & \text{1 octet} \end{cases}$$

Par exemple, pour une texture de définition 128x128 et un hémisphère composé de 500 directions, la structure occupe 31 Mo en mémoire, soit 128 Mo pour la totalité de la boîte.

3.2.3 Rendu

Le principe est de construire la texture en fonction de la position de l'observateur, texel par texel. Pour l'ensemble de la boîte, l'opération est effectuée 3 fois[31]. La couleur d'un texel est déterminée en trouvant la direction discrétisée approximant le mieux la direction de l'observateur par rapport au texel[32]. Quand on dispose de cette direction, on dispose de la couleur, de la normale et de la profondeur stockées lors de la phase de précalculs, ce qui nous permet de calculer la couleur du texel avec le modèle d'illumination de Phong.

Pour chaque texel de la texture plaquée sur une face de la boîte les opérations à effectuer sont (figure 3.7) :

1. Exprimer dans le repère local du texel le rayon optique $\overrightarrow{R}$ issu de la caméra passant par le centre du texel.
2. Trouver le rayon le plus proche parmi les directions précalculées. Nous obtenons un tableau D à deux dimensions stockant l'indice de la meilleure direction en chaque élément $D_{i,j}$.

[31] Pour un cube, il ne peut y avoir que 3 faces visibles à un instant donné.
[32] Cette méthode peut être vue comme un lancer de rayons inversé précalculé.

3. Calculer la couleur du texel à l'aide des informations stockées dans la direction trouvée.

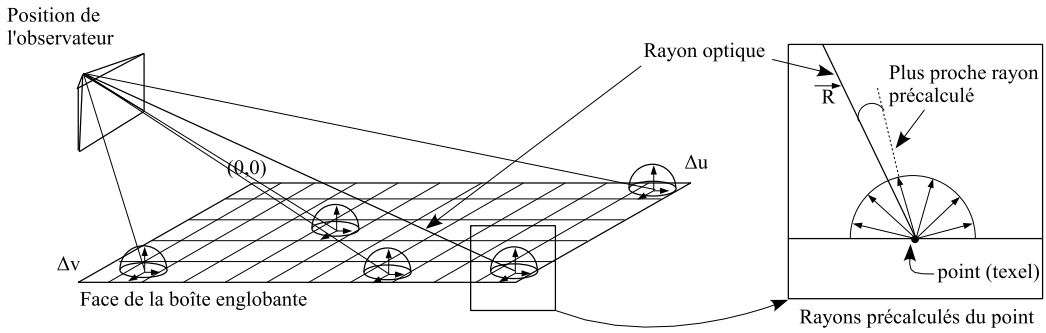


FIG. 3.7 – Rendu : reconstruction de la texture point par point en fonction des hémisphères précalculés stockant les intersections des rayons.

Durant la seconde phase, on applique le modèle d'éclairement de Phong en chacun des texels. On distingue deux cas :

- La position de l'observateur a changé.
- Les positions des sources de lumière ont varié mais pas la position de l'observateur.

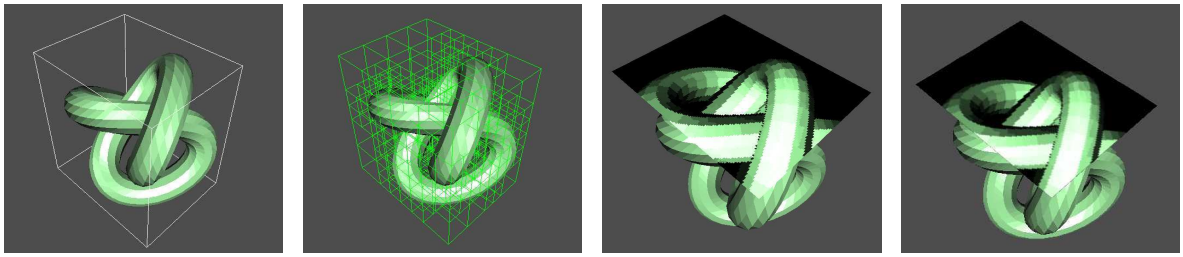
Dans le premier cas, on doit tout recalculer, c'est à dire qu'il faut effectuer les phases 1 à 3. Dans le second cas, seul l'éclairement de la scène est recalculé à partir de D.

3.2.4 Résultats

Concernant la phase de précalculs, étant donné que nous lançons beaucoup de rayons par texel (un rayon par direction de l'hémisphère), nos premiers essais nous obligèrent à accélérer cette partie : pour une résolution de 250 directions pour l'hémisphère et une définition de 64x64 texels pour les texture, le temps de calcul dépassait une heure pour une seule texture (Athlon 1Ghz). Nous avons alors utilisé un partitionnement de la scène de base en arbre octal (figure 3.8(b)). Bien que le temps de précalcul du lancer de rayon dépende fortement de la complexité de la scène, nous avons noté une accélération de l'ordre d'un facteur 20.

La figure 3.8 montre le résultat pour une seule texture plaquée sur la face supérieure de la boîte englobante. La définition de la texture est 128x128 texels ; la résolution de l'hémisphère est 256 directions.

Comme on peut le voir, des discontinuités apparaissent aux bordures de la texture, par rapport au modèle de base. Ces discontinuités sont dues à la faible résolution de l'hémisphère des directions.



(a) Le modèle de base et sa boîte englobante.

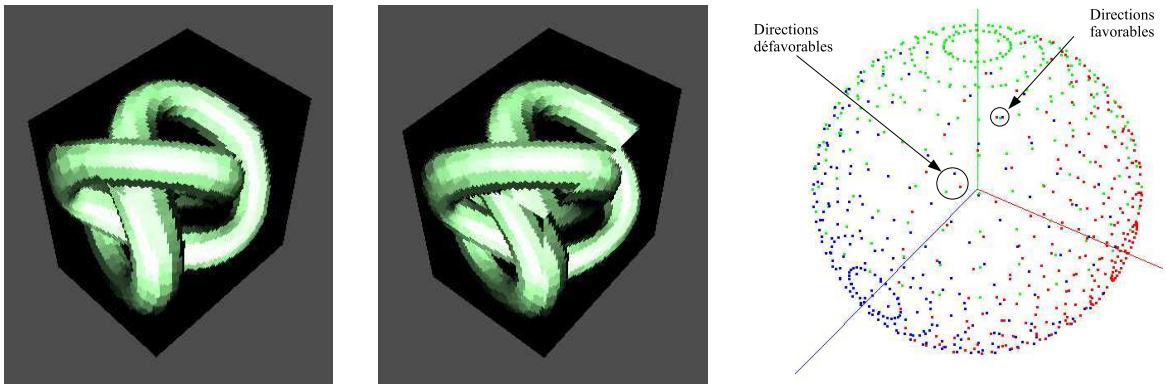
(b) L'octree utilisé durant la phase de précalculs.

(c) Rendu de la face superieure de la boîte, texturée avec notre méthode.

(d) = (c) + interpolation bilinéaire des couleurs des texels.

FIG. 3.8 – Résultat de la méthode appliquée à un modèle de tore, en rendu à ombrage plat. La définition de la texture (figure (c) et (d)) est de 128x128 texels, l'hémisphère est constitué de 256 directions quantifiées. Temps de précalcul de 8 mn environ. Rendu à 15 FPS.

Un autre problème se produit lorsque l'on affiche la boîte englobante complète : certaines directions donnent des discontinuités aux bordures entre les faces (figure 3.9(b)), tandis que d'autres directions d'observation donnent de meilleurs résultats (figure 3.9(a)).



(a) Angle d'observation favorable.

(b) Angle d'observation défavorable.

(c) Différences entre les directions d'échantillonnage des hémisphères des faces.

FIG. 3.9 – Résultat de la méthode avec une boîte englobante complète. La définition des texture est 64x64 texels. La résolution de l'hémisphère est 250 directions.

Ceci est dû au fait qu'avec cet algorithme, les texels adjacents d'une face à l'autre de la boîte peuvent être calculés avec des directions très différentes. La figure 3.9(c) illustre ce point. Les directions des hémisphères des trois faces affichées sont représentées avec des couleurs différentes. Quand la direction d'observation est favorable, les directions les plus proches pour les trois hémisphères sont

peu différentes. A l'inverse, quand les trois directions les plus proches sont trop différentes, les rayons utilisés pour reconstuire des texels voisins donnent des points de la scène très éloignés.

Concernant les résultats obtenus pour une boîte complète, l'espace mémoire est très grand (187 Mo pour une définition de 128x128 texels pour les textures et 500 directions pour l'hémisphère). Néanmoins, le temps de rendu est indépendant de la complexité géométrique. Il dépend uniquement de la résolution des textures. Sur un AMD Athlon 1GHz, nous obtenons 15 FPS pour des textures de définition 128x128 et plus de 25 FPS pour des textures de définition 64x64. Si l'éclairement seul change, nous ne recalculons que la couleur des texels sur la base des directions les plus proches déjà calculées (stockées dans $D_{i,j}$). Dans ce cas, le taux d'affichage est supérieur à 80 FPS pour une définition de 64x64 texels, et supérieur à 60 FPS pour une définition de 128x128.

3.2.5 Discussion

Le fait que les calculs s'effectuent par texel permet d'obtenir tous les effets de parallaxe (désocclusion et masquage) au sein de la texture générée. Néanmoins, l'espace mémoire nécessaire est beaucoup trop important pour être intéressant dans le contexte d'une application de simulation d'environnement 3D.

Concernant la phase de précalculs, le temps nécessaire pour l'accomplir est très grand car le lancer de rayons est effectué n fois par texel (n = résolution de l'hémisphère). Malgré l'utilisation d'un octree, le temps de calcul reste prohibitif si l'on souhaite avoir de bons résultats graphiques ; en fonction de la scène et de la résolution de l'hémisphère, les calculs peuvent nécessiter plusieurs heures.

Ainsi, nous sommes obligés de restreindre le nombre de directions et la définition des textures pour pouvoir obtenir un temps de précalculs et un encombrement mémoire "acceptables". Ceci resteint l'intérêt de la méthode.

S'ajoutent à tous ces inconvénients les problèmes d'aliassage que nous avons communément avec le lancer de rayons. En particulier, les petits objets peuvent être perdus entre les rayons et apparaître ou disparaître brusquement lors de l'animation.

3.3 Textures directionnelles de surfels quantifiés

Dans les travaux exposés dans cette partie, nous avons amélioré la méthode prédédente sous plusieurs aspects en employant une représentation en nuage de points du modèle complexe de base.

- Ceci nous permet de réduire l'espace mémoire car un texel stocke maintenant simplement une référence à un point, pour chaque direction de son hémisphère, au lieu des informations géométriques et de la couleur.
- Le temps de précalcul est grandement réduit car, bien que la conversion du modèle de base en nuage de points prenne du temps, la projection de ce nuage sur une face de la boîte englobante est extrêmement rapide comparé au lancer de rayons. Cette projection est à la base du précalcul de visibilité de la texture, comme nous le détaillons dans la section suivante (section 3.3.1).
- La qualité visuelle de l'affichage est accrue car nous réduisons les problèmes dus au lancer de rayons (petits objets) avec l'emploi du nuage de points pour les précalculs. D'autre part, nous utilisons une répartition mieux adaptée des directions (section 3.3.1.3).
- Finalement, nous pouvons appliquer des modèles d'éclairement locaux complexes sur les points, comme l'auto-ombrage, qui accroissent notablement la qualité du rendu (section 3.3.2).

Toutes ces améliorations reposent sur une représentation du modèle de base sous forme de surfels quantifiés que nous avons appelés QLS ("Quantized Light Surfels"). Un QLS est un surfel ("élément de surface", voir section 1.3.2) stockant des informations de visibilité (ombrage), des informations géométriques (position, orientation) et la couleur d'un point de la surface du modèle complexe. Toutes ces informations sont quantifiées de manière à être représentées de façon compacte en mémoire.

3.3.1 Acquisition d'une texture directionnelle

La différence principale avec la méthode précédente est que nous ne nous basons plus sur un lancer de rayon pour précalculer l'hémisphère des directions en chaque texel. A la place, nous projetons le nuage de points issu du modèle complexe sur chaque plan de la boîte.

Tout d'abord, nous devons transformer le maillage de l'objet de base en un nuage dense de points.

3.3.1.1 Décomposition en surfels du modèle complexe

Ce nuage de points est obtenu à partir du maillage de triangles du modèle complexe. L'algorithme consiste à subdiviser récursivement chaque triangle comme suit.

Étant donné un triangle t, nous le subdivisons en 4 en scindant ses arêtes en deux, comme le montre la figure 3.10. Ensuite, nous appliquons la subdivision aux sous-triangles créés. Le critère d'arrêt est le suivant : nous ne subdivisons pas les triangles constitués d'une arête plus petite qu'une distance fixée. Cette technique n'est pas optimale dans le cas de triangles très étirés, car la subdivision se termine en fonction de la plus petite de ses arêtes. Néanmoins, dans le cas où le maillage est constitué de triangles "raisonnablement" proportionnés, la méthode fonctionne bien.

L'ensemble de points que nous obtenons avec cet algorithme est défini par tous les sommets des triangles subdivisés (figure 3.10(d)). Chaque point est défini par ses coordonnées 3D dans le repère de la scène ainsi que par sa couleur et sa normale. Ces données sont interpolées lors de la subdivision à partir des sommets du maillage de base.

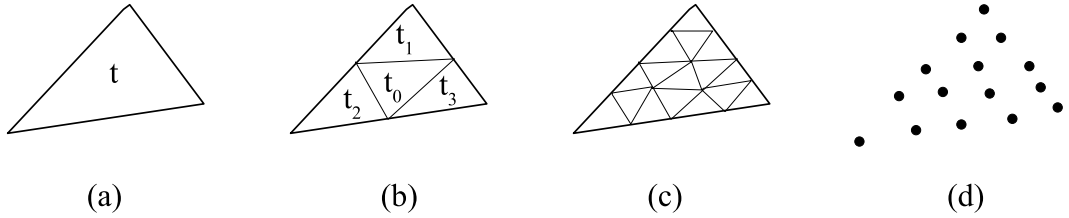


FIG. 3.10 – Création du nuage de points par subdivision récursive des triangles du maillage complexe.

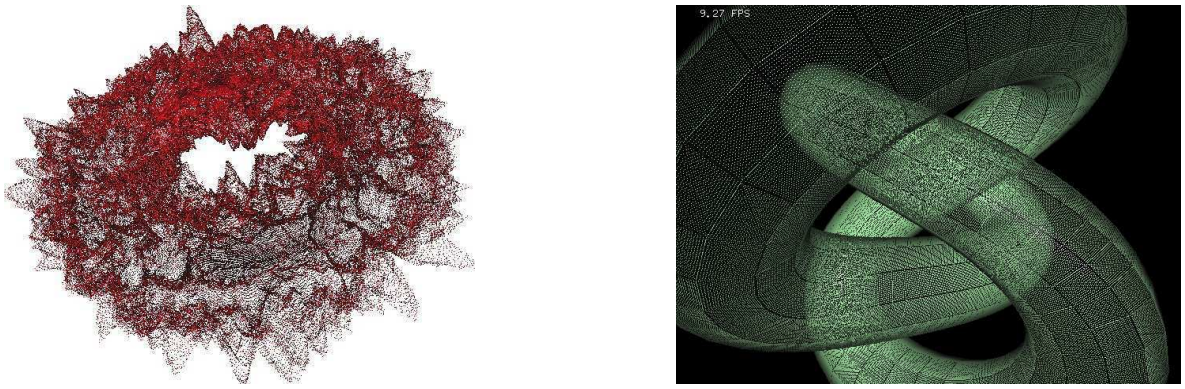


FIG. 3.11 – Résultat de notre méthode de discrétisation appliquée à un tore à surface perturbée avec une carte d'élévation (par *déplacement de surface*) et à un maillage représentant un noeud torique.

La figure 3.11 montre le résultat de cet algorithme appliqué à deux modèles géométriques.

Une fois ce nuage de points créé, nous calculons les textures directionnelles appliquées à la boîte englobante.

3.3.1.2 Calcul de la texture

Comme pour la méthode précédente, une texture directionnelle stocke en chaque texel les points du modèle complexe s'y projetant pour toutes les directions d'un hémisphère. La différence est que nous n'effectuons pas de lancer de rayons pour obtenir ces points.

Pour chaque direction d de l'hémisphère d'une face f_i de la boîte, l'algorithme permettant d'obtenir une texture directionnelle est le suivant (figure 3.12) :

Pour chaque direction d de l'hémisphère :

1. Créer un tampon de profondeurs T à deux dimensions, de résolution égale à la définition de la texture, positionné sur f_i, stockant en chaque élément $T(u, v)$ la profondeur z et un l'indice k d'un point dans le nuage.
2. Projeter orthographiquement tous les points du nuage sur f_i en ne gardant en chaque $T(u, v)$ que le point le plus proche parmi ceux qui s'y projetent.
3. Stocker l'indice k pour la direction d de l'hémisphère de chaque texel (u, v).

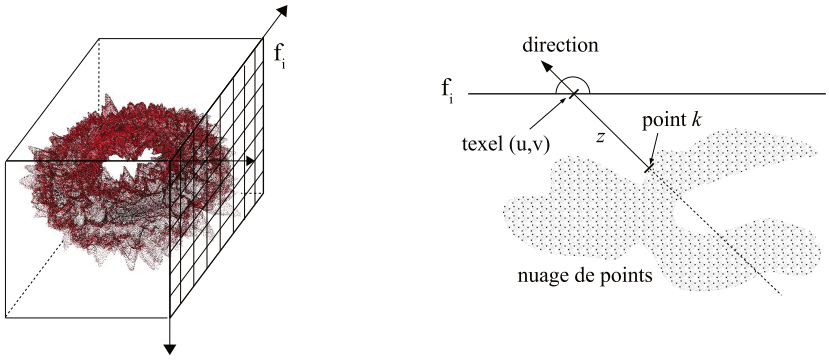


FIG. 3.12 – Calcul d'une texture directionnelle

Nous obtenons finalement une texture stockant en chaque élément un tableau d'indices de points (un point par direction).

3.3.1.3 Hémisphère des directions

Par rapport à la méthode précédente, nous avons amélioré la répartition des directions. La problématique de la discrétisation de l'hémisphère est que la répartition doit :

1. permettre une localisation d'un vecteur quelconque très rapidement,

2. distribuer le mieux possible un nombre fixé de directions de façon à couvrir uniformément la surface de la scène.

Concernant le premier point, la décomposition en secteurs des coordonnées sphériques d'une direction donne les meilleurs résultats en termes de rapidité de temps de calcul. En particulier la localisation d'une direction ne dépend pas du nombre de directions dicrétisées. Nous avons donc gardé cette décomposition.

Concernant la distribution des directions, la discrétisation régulière des directions en coordonnées sphériques donne de mauvais résultats en termes de recouvrement de la scène : comme le montre la figure 3.13, les angles rasants donnent des points trés éloignés d'une direction à l'autre, ce qui se traduit, lors du rendu, par de fortes discontinuités entre deux directions voisines.

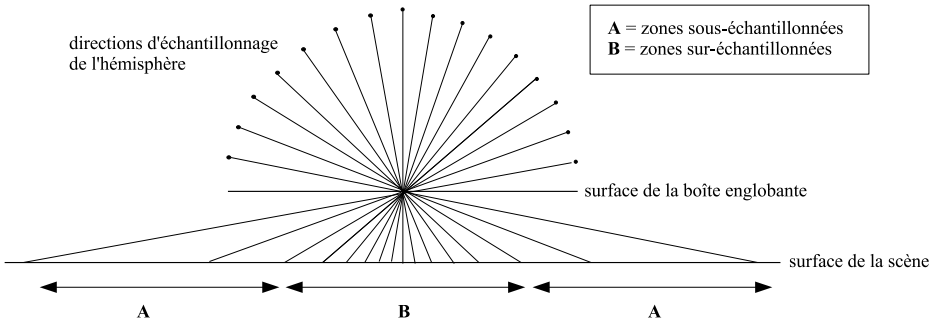


FIG. 3.13 – Variation de l'échantillonnage de la surface de la scène, avec des rayons issus de l'hémisphère des directions d'un texel de la boîte englobante.

Pour améliorer cette répartition, nous avons augmenté le nombre de directions sur les angles rasants. C'est à dire qu'au lieu de subdiviser régulièrement l'angle d'élévation φ, nous utilisons des intervalles se réduisant petit à petit selon une courbe en racine carrée (cf. figure 3.14). Cette fonction nous permet techniquement de conserver des calculs de localisation très rapides.

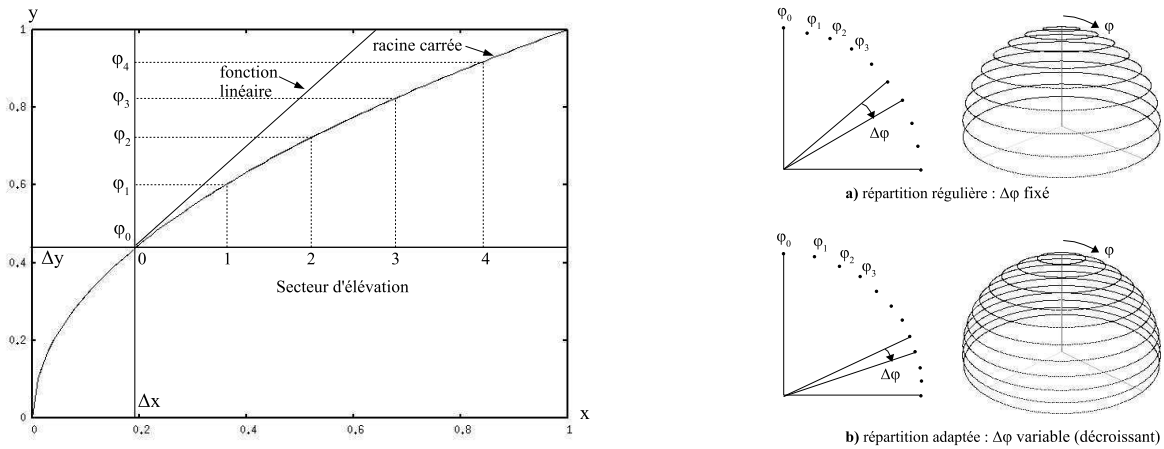


FIG. 3.14 – Répartition adaptée des directions sur l'hémisphère. La répartition de l'angle d'élévation est basée sur une portion de la fonction racine carrée et permet d'augmenter le nombre de directions aux angles rasants.

3.3.2 Modèles d'éclairement implémentés

Lors du rendu, le calcul de la couleur d'un texel s'effectue donc en appliquant un modèle d'éclairement à un point du nuage (un QLS) sélectionné par son hémisphère de visibilité. Comme une texture directionnelle stocke la visibilité des points du nuage en fonction de la direction d'observation, lors du rendu nous sélectionnons le sous ensemble des points visibles du nuage auquels nous devons appliquer le modèle d'éclairement. Le nombre de points de ce sous ensemble est toujours constant : si d est la définition des textures, sachant qu'au maximum 3 faces de la boîte sont visibles à un instant donné, nous n'auront à calculer l'illumination que de $3.d^2$ points.

Par exemple, dans nos tests, le nuage de points que nous utilisons est constitué en moyenne de 1.5 million de points. Pour une définition de texture de 128x128 texels, nous n'appliquons le calcul d'illumination qu'à 49512 points, soit 3% des points. Ceci nous permet d'employer des modèles d'illumination coûteux en temps de calculs tels que l'application de BRDF acquises physiquement, ou de calculer l'auto-ombrage des points.

Nous avons implémenté le modèle d'éclairement de Phong, le placage d'environnement ainsi que l'application de BRDF acquise physiquement (section 3.3.2.2). Nous avons également implémenté l'auto-ombrage avec une compression originale l'hémisphère d'ombrage des points, que nous présentons en section 3.3.2.3 et en Annexe B.

3.3.2.1 Repère local des points

Pour appliquer les différents modèles d'éclairement locaux, nous devons disposer d'un repère local en chaque point permettant d'y exprimer les directions d'observation et d'éclairement. Etant donné que nous ne disposons que des coordonnées 3D du point et de sa normale, nous construisons un repère orthonormé de la façon suivante : soient $\overrightarrow{v}(\theta_v, \varphi_v)$ la normale à laquelle on doit associer une base et $(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$ le repère global de la scène, on crée tout d'abord un vecteur $\overrightarrow{v_p}$ perpendiculaire à $\overrightarrow{v}$ par $\overrightarrow{v_p} = (\theta_v, \varphi_v + \pi/2)$ (figure 3.15), puis le repère est finalement obtenu en ajoutant le dernier axe : $\overrightarrow{v_B} = \overrightarrow{v} \wedge \overrightarrow{v_p}$. Les cas particuliers des pôles ($\overrightarrow{v}$ colinéaire à $\overrightarrow{y}$) sont traités séparément.

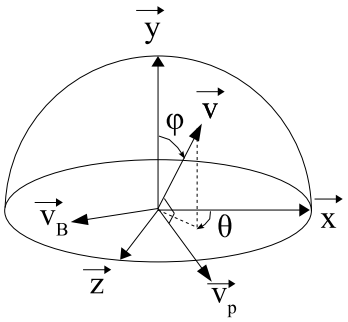


FIG. 3.15 – Construction du repère local d'un point.

L'intérêt de procéder ainsi, par rapport à une rotation du repère, est que dans notre implémentation, le nombre d'opérations arithmétiques à effectuer est inférieur, et comme nous devrons effectuer ce calcul pour chaque texel, les calculs doivent être rapides.

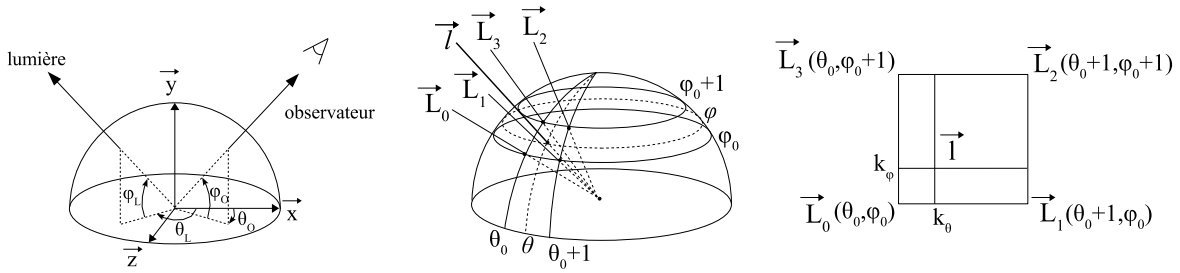
3.3.2.2 BRDF par acquisition physique

Principe Nous avons utilisé la BRDF d'une surface réelle, acquise à l'aide d'un spectrophotogoniomètre. Le principe est d'échantillonner la BRDF d'une surface selon un nombre fixé de directions et d'utiliser directement ces données pour calculer la couleur du point en fonction de la direction d'éclairement et d'observation.

Dans nos travaux, nous avions à notre disposition deux BRDF isotropes : la BRDF du satin et une BRDF synthétique. Cette dernière est créée "à la main" afin de valider l'implémentation, et donne les couleurs rouge, verte ou bleue en fonction de la direction de la lumière et de l'éclairement.

Structure des données La structure de données représentant la BRDF est un tableau B à quatre dimensions représentant la réflectance de la surface dans une direction d'observation (θ_o, φ_o) en fonction de la direction d'éclairement (θ_L, φ_L) (figure 3.16(a)). Dans notre cas, la longueur d'onde de la

lumière est constante, ainsi nous ne stockons que les composantes (R, V, B) de la couleur en chaque élément du tableau.



(a) Principe de la BRDF.

(b) Interpolations bilinéaires des couleurs.

FIG. 3.16 – BRDF par acquisition physique.

Application Pour passer d'une BRDF discrète, représentée par le tableau B à quatre dimensions, à une BRDF continue sur l'hémisphère, nous interpolons bilinéairement les couleurs entre les directions discrétisées.

Par exemple, pour une direction d'observation discrétisée $\overrightarrow{V}$ et une direction d'éclairement non discrétisée $\overrightarrow{l}$, la couleur C obtenue est calculée comme suit (voir figure 3.16(b)) :

Soit $\overrightarrow{l}(\theta, \varphi)$ la direction d'éclairement, $\overrightarrow{L_0}$, $\overrightarrow{L_1}$, $\overrightarrow{L_2}$ et $\overrightarrow{L_3}$ ses directions discrétisées adjacentes, on obtient les couleurs C_i associées aux directions d'éclairement discrétisées par $C_i = B[\overrightarrow{V}, \overrightarrow{L_i}]$, $i \in [0, 1, 2, 3]$.

Ensuite on interpole ces valeurs bilinéairement :

$$\begin{aligned} C &= C_a + k_\theta.(C_b - C_a) \\ \text{avec } C_a &= C_0 + k_\varphi.(C_3 - C_0) \\ \text{et } C_b &= C_1 + k_\varphi.(C_2 - C_1) \end{aligned}$$

Symétriquement, nous appliquons ce principe pour la direction (non discrétisée) d'observation : nous calculons la couleur finale par interpolation bilinéaire des quatres couleurs issues des directions discrétisées de la direction d'observation.

Ainsi, pour obtenir une couleur en fonction d'une direction d'observation et d'éclairement quelconque, nous devons effectuer cinq interpolations bilinéaires. C'est un calcul lourd mais, comme nous le verrons dans la section des résultats, il s'effectue en temps réel sur une machine milieu de gamme (Athlon 1Ghz).

3.3.2.3 Auto-ombrage par hémisphère d'ombrage quantifié

Le principe de l'auto-ombrage est de prendre en compte, lors du rendu d'un modèle géométrique, la projection de l'ombre qu'il induit sur son propre relief, en fonction d'une source de lumière ponctuelle. L'auto-ombrage ajoute beaucoup de réalisme au rendu, mais est également coûteux à calculer.

Parmi les techniques existantes à notre disposition, qui fonctionnent en temps réel, la méthode présentée par Williams [Wil78] permet de calculer cet auto-ombrage si l'on dispose au moment du rendu du maillage géométrique de la scène, ce qui n'est pas notre cas. Dans le cas ou le relief du modèle est représenté par des cartes d'élévation, les techniques d'*horizon-mapping* [Max88, SC00, HS99] (voir section 1.4.1.3) peuvent être employées, mais nos textures plaquées sur la boîte englobante ne sont pas exprimables sous forme de champ de hauteur : le relief peut varier complètement d'un point de vue à l'autre.

En Annexe B, nous présentons le principe de notre décomposition en harmoniques principaux de l'hémisphère d'ombrage que nous avons développé.

Le principe est de créer, pour un point donné, son hémisphère d'ombrage (voir la section 1.4.1.3 page 23). L'ensemble des directions de l'hémisphère est discrétisé en coordonnées sphériques de façon habituelle et stocke en chaque direction (α, β), correspondant à la direction de la lumière, la valeur 0 si le point est dans l'ombre, ou la valeur 1 dans le cas contraire.

Ensuite, la *fonction d'ombrage* $f(\alpha, \beta)$ (voir Annexe B) définie à partir de cet hémisphère est décomposée en harmoniques principaux avec la transformée de Fourier 2D discrète de f. Les harmoniques sont quantifiés et stockés sur 9 octets.

Ces harmoniques nous permettent, lors du rendu, de reconstruire l'hémisphère d'ombrage avec la transformée de Fourier inverse, ce qui s'avère très rapide : quelques cosinus à calculer (voir Annexe B).

La fonction d'ombrage obtenue prenant ses valeurs dans l'intervalle $[0, 1]$, nous appliquons ce coefficient à la couleur calculée par le modèle d'éclairement du point.

Durant la phase de précalculs, nous obtenons l'hémisphère d'ombrage d'un point avec un lancer de rayons dans la scène complexe d'origine, selon toutes les directions discrétisées de l'hémisphère. Dans nos tests nous avons discrétisé l'hémisphère selon 16 azimuths et 8 angles d'élévation, soit 129 directions. Nous avons expérimenté une discrétisation plus fine (513 directions) sans noter de différence visuelle significative dans la précision des ombres.

3.3.3 Représentation des données

Comme nous l'avons vu en section 3.2.5, le problème principal des textures dépendantes du point de vue est la très grande quantité de données à stocker. Nous avons donc représenté les données de façon à minimiser le coût en espace mémoire.

3.3.3.1 Points

Tout d'abord, les données associées aux points sont quantifiées. Un point est défini par une position (3D), une couleur (RVB) et une orientation (vecteur 3D). La couleur et la normale du point sont quantifiées de la même façon que pour la méthode précédente (voir section 3.2.2.2) : la normale du point est l'indice d'une direction de la sphère-unité discrétisée et la couleur est un indice dans la palette de couleurs de la scène.

La position 3D du point est exprimée par rapport à la boîte englobante car tous les points y sont contenus. Ainsi, nous représentons les coordonnées (x, y, z) du point sur un octet par composantes ; chaque composante est une proportion de la taille de la boîte le long de chaque axe.

En ajoutant les données associées aux harmoniques de l'hémisphère d'ombrage, un QLS est finalement défini par :

$$\text{QLS} \begin{cases} \text{ses coordonnées 3D discrétisées} & \text{3 octets} \\ \text{sa normale quantifiée} & \text{2 octets} \\ \text{sa couleur quantifiée} & \text{1 octet} \\ \text{son hémisphère d'ombrage quantifié} & \text{9 octets} \end{cases}$$

3.3.3.2 Textures directionnelle

Une texture directionnelle est un tableau 2D de taille fixée stockant en chaque texel un tableau 2D de taille n stockant un indice vers un point du nuage (n = le nombre de directions de l'hémisphère). Nous avons représenté cet indice sur 3 octets. Ceci nous permet d'indexer jusqu'à 2^{24} points.

3.3.3.3 Boîte englobante

Pour optimiser l'espace mémoire, tous les points du nuage n'étant pas référencés par les textures sont supprimés du nuage puis les indices des points restants sont recalculés. Nous avons observé en moyenne que plus de 40% des points étaient supprimés par cette opération pour une définition de texture de 128^2 et plus de 50% pour une définition de 64^2.

Au final, les données définissant une boîte englobante sont de deux types : les informations *géométriques* et les informations de *visibilité*. Les informations géométriques correspondent à l'ensemble des QLS. Les informations de visibilité correspondent aux textures ; celles-ci permettent d'obtenir, par le biais des hémisphères des texels, l'ensemble des points visibles selon une position de caméra.

3.3.4 Rendu

Le rendu s'effectue en deux parties : le calcul de visibilité et le calcul de la couleur des texels.

1. Le calcul de visibilité consiste, comme dans la méthode précédente (section 3.2.3), à localiser en chaque texel des faces visibles la direction la plus proche parmi les directions discrétisées, pour le rayon optique issu de la caméra (figure 3.7 page 89). A l'issue de ce calcul, nous obtenons pour chaque texel un indice référençant un point du nuage.
2. Le calcul de la couleur des texels est effectué en appliquant un modèle d'éclairement sur chaque point référencé des texels.

3.3.5 Résultats

Dans la figure 3.17, nous avons appliqué notre méthode à un modèle de tore géométriquement perturbé par *déplacement de surface,* composé originellement de 1.5 million de faces. Le taux d'affichage de ce modèle en rendu conventionnel est de 2 FPS (GeForce4 Ti, AMD Athlon 1 Ghz). Avec notre méthode, nous obtenons 20 FPS sans le calcul de l'auto-ombrage, pour définition de texture de 128^2. Avec le calcul de l'auto-ombrage, nous obtenons 15 FPS.

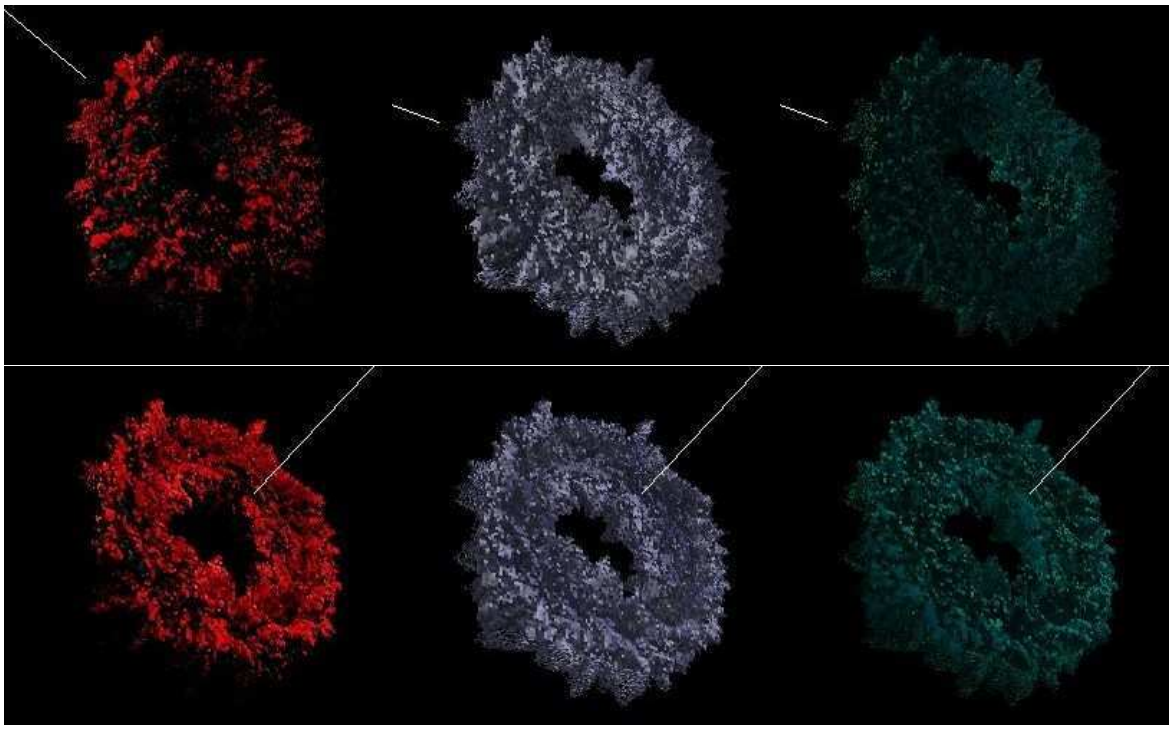


FIG. 3.17 – Résultats pour une variation de l'éclairement (de haut en bas). Définition des textures : 128x128 ; 1742 directions ; QLS : 4.97 Mo ; Hémisphères de visibilité : 326.81 Mo. Colonnes de gauche à droite : Phong (20 FPS), BRDF de Satin (15 FPS), BRDF synthétique (15 FPS). Toutes les images emploient l'auto-ombrage.

Dans la figure 3.18, nous avons appliqué la méthode à un modèle composé de très petits objets (ligne du haut). La ligne du bas présente le modèle de tore affiché avec notre méthode selon différentes définitions de texture. Nous avons gardé une distance d'observation fixe de façon à mettre en évidence les différences : plus la définition est faible, plus le rendu est rapide, mais plus les texels sont gros et dégradent l'image.

La figure 3.19 montre les différents modèles d'éclairement que nous avons implémentés : Phong, placage d'environnement, BRDF de satin et BRDF synthétique.

Nous présentons finalement en figure 3.20 la méthode d'auto-ombrage appliquée au tore perturbé et au noeud torique (avec et sans auto-ombrage).

Le tableau 3.1 résume les taux d'affichages que nous obtenons avec les différents modèles d'éclairement que nous avons utilisés : le modèle de Phong, l'utilisation de BRDF acquise physiquement, le placage d'environnement. Ce taux d'affichage ne dépend que de la définition des textures. La dernière colonne indique l'impact qu'à notre méthode d'auto-ombrage sur le temps d'affichage.

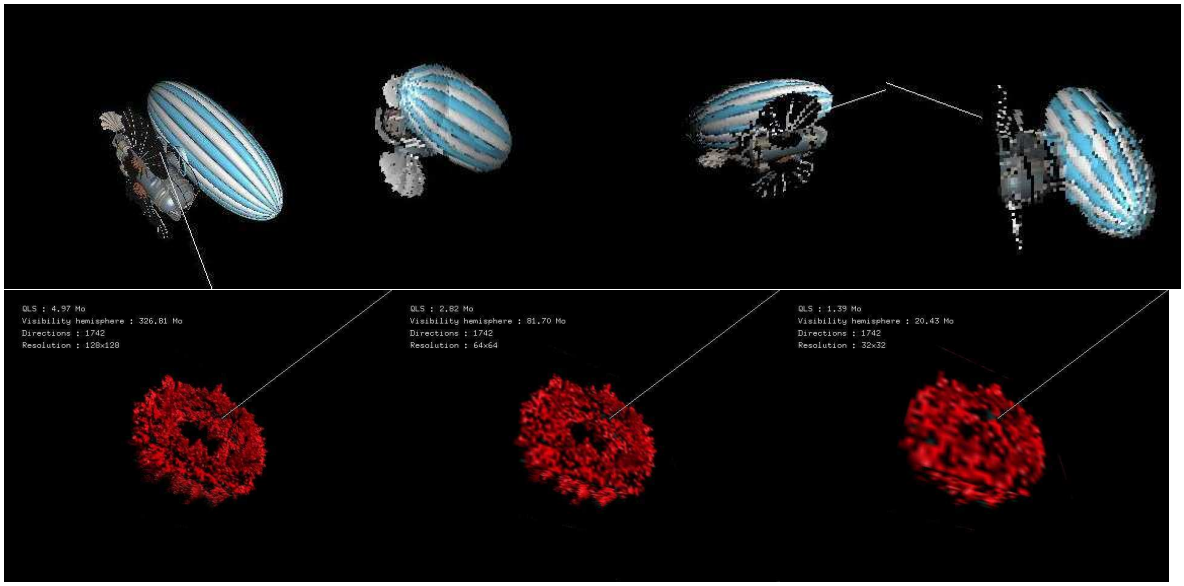


FIG. 3.18 – Méthode appliquée à un modèle composé de petits objets : le dirigeable, ligne du haut. Définition des textures : 64x64, Nb. directions : 4902, 50 FPS, pas de d'interpolation bilinéaire. Ligne du bas : résultat de la méthode appliquée au tore pour différentes définition de texture : 128^2 (20 FPS), 64^2 (50 FPS) et 32^2(70 FPS) de gauche à droite.

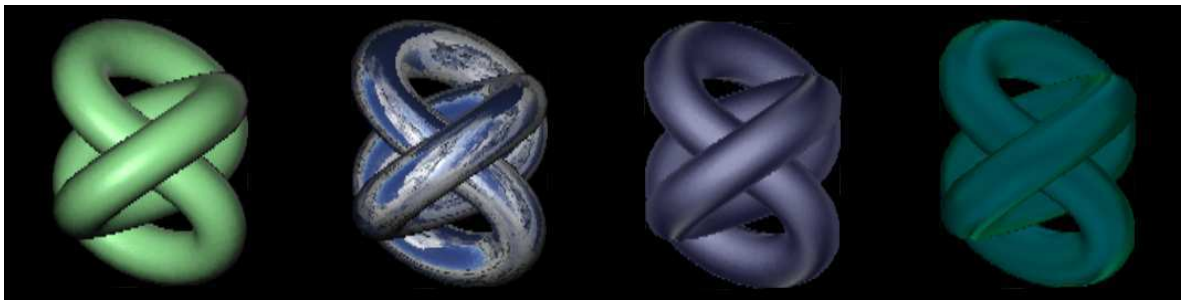


FIG. 3.19 – Modèles d'illuminations gérés par notre méthode. La définition des textures utilisées pour ces images est 128x128. Pour une qualité d'image maximale, nous avons positionné l'observateur à un pôle de l'hémisphère. De gauche à droite : Phong, Phong+placage d'environnement, BRDF satin, BRDF synthétique.

Définition (pts calculés)	**Phong**	**BRDF**	**Placage d'env.**	**sans auto-ombrage**
128x128 (49512)	20	15	18	+10 FPS
64x64 (12288)	50	47	48	+5 FPS
32 (3072)	70	70	70	+2 FPS

TAB. 3.1 – Taux d'affichage en fonction du modèle d'éclairement.

Le tableau 3.2 résume l'espace mémoire nécessaire pour stocker les six textures directionnelles d'une

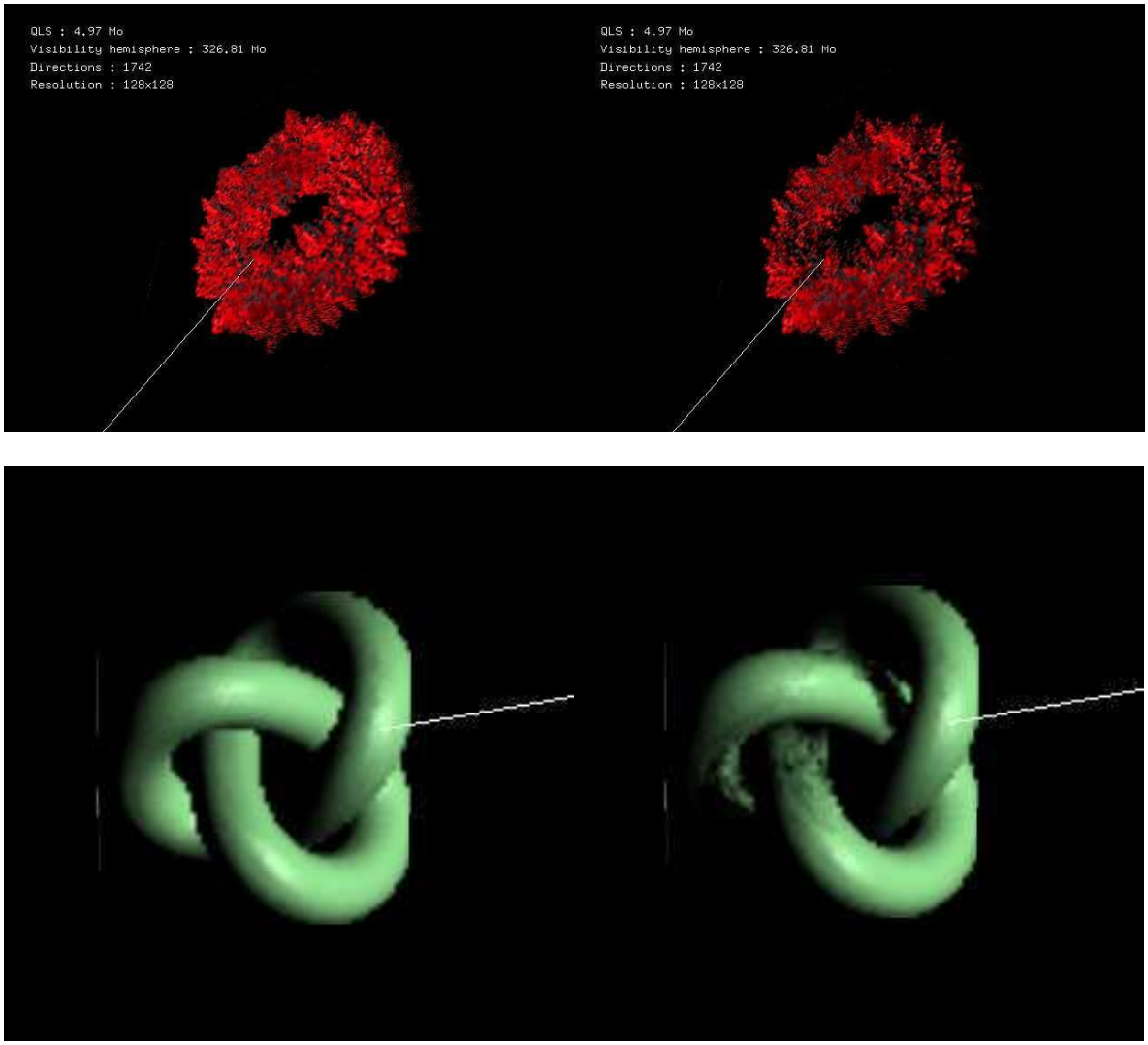


FIG. 3.20 – Application de l'auto-ombrage par compression en harmoniques principaux de l'hémisphère d'ombrage. Images de gauche : rendu standard (Phong) 30 FPS. Images de droite : application de l'auto-ombrage, 20 FPS. Le segment blanc indique la direction de l'éclairage. Définition de texture 128^2.

boîte, en séparant la taille du nuage de points (de QLS) et la structure de visibilité (les hémisphères de visibilité des texels). Les résultats sont présentés pour différentes résolutions pour les hémisphères. Bien que les résultats dépendent également de la scène complexe de base, celle-ci n'a qu'un faible impact sur les valeurs constatées.

3.3.6 Discussion

Comme le montrent les résultats, le point négatif de cette méthode est l'espace mémoire nécessaire pour stocker la structure de visibilité. Pour de bons résultats visuels, la définition de texture minimale

Nb. directions	72	262	262	262	1742	1742	1742	4902
Définition	128	32	64	128	32	64	128	64
Nuage de QLS (Mo)	3,64	0,67	1,78	3,9	1,39	2,82	4,97	4,52
Visibilité (Mo)	13,69	3,08	12,23	49,31	20,43	81,70	326,81	229,83

TAB. 3.2 – Espace mémoire nécessaire pour stocker une boîte englobante à textures directionnelles, en fonction des résolutions des hémisphères de visibilité et de la définition des textures.

est de 128^2 et plus de 2000 directions par hémisphère. Dans ce cas, la taille mémoire est supérieure à 350 Mo, ce qui est trop important pour une application de rendu en 3D temps réel.

Néanmoins, les taux d'affichages que nous obtenons sont honorables, et ne dépendent que de la définition des textures. Le principal point positif de cette approche est notre méthode d'illumination à base de surfels quantifiés (QLS). L'espace mémoire nécessaire pour les stocker est très faible : 15 octets par QLS, ce qui est comparable, voir inférieur, aux principales méthodes existantes de rendu à base de surfels [PZvG00, GBP04, GP03, RPZ02, ZGPvB01], qui pourtant ne prennent pas en compte l'auto-ombrage. De plus, notre méthode d'auto-ombrage par compression en harmoniques principaux des hémisphères d'ombrage s'avère efficace car la décompression de l'hémisphère est très rapide et donne de bons résultats visuels.

Les travaux à mener en priorité pour améliorer cette méthode doivent clairement porter sur une meilleure représentation de la structure de visibilité. Dans ce cadre, une subdivision hiérarchique du nuage de points, à la manière des *QSplat* de Rusinkiewicz [RL00] devrait permettre d'obtenir de bien meilleurs résultats.

Une autre possibilité permettant d'abaisser le nombre de directions des hémisphères de visibilité serait d'effectuer une interpolation bilinéaire des surfels : pour une direction donnée, on créerait les surfels manquant par interpolation de leurs composantes (position, normales, hémisphères de visibilité) à partir des directions les plus proches.

3.4 Conclusion

Dans ce chapitre nous avons présenté une méthode permettant de remplacer un modèle géométrique complexe par une boîte englobante à textures directionnelles 4D. Nous avons étendu le principe du *lancer de rayons virtuels* des BTFs de Dischler[Dis98] de façon à ce que le placage de texture puisse s'effectuer en temps réel.

Pour cela, nous avons développé un modèle de surfel permettant de calculer très rapidement, et pour un espace mémoire minime, l'auto-ombrage d'un point ainsi que les modèle d'éclairement locaux habituellement utilisés en rendu en temps réel (Phong et placage d'environnement). Nous avons également implémenté un modèle d'éclairement basé sur une BRDF isotrope réelle, ce qui n'a pas été fait en temps réel à notre connaissance.

4

Interpolation de points de vue par pixel

4.1 Introduction

Dans les deux précédents chapitres, nous avons implémenté différentes techniques à base d'images fonctionnant en temps réel : l'interpolation de points de vue, l'interpolation d'images et les textures se modifiant en fonction du point de vue. Bien que donnant des résultats rapides, la qualité visuelle dépend grandement du nombre d'échantillons d'image ou de visibilité que nous stockons.

Dans le cas de l'interpolation d'images, les effets de parallaxe génèrent des artéfacts de flou, qui ne peuvent être résolus qu'en augmentant le nombre de points de vue de référence. Dans le cas des textures directionnelles, les informations de visibilité nécessitent un très grand espace mémoire. Pour l'interpolation de points de vue, le principal problème est la dépendance de la méthode vis à vis du processeur central. En effet, la *déformation 3D d'image* communément utilisée pour effectuer l'interpolation [SD96] est pour l'heure impossible à implémenter en temps réel. Le *placage de texture en relief* [OBM00] est la méthode qui se rapproche le plus d'une implémentation matérielle actuellement.

Dans ce chapitre, nous présentons deux méthode permettant d'effectuer l'interpolation de points de vue de référence intégralement par la carte graphique.

La première section est consacrée à une méthode que nous avons développée qui permet d'obtenir l'image d'un modèle complexe pour un point de vue positionné sur une trajectoire prédéfinie par l'utilisateur. Dans cette section, nous décrivons la méthode d'interpolation de points de vue que nous avons utilisée tout au long de ce chapitre. Nous décrivons également notre méthode de compression/décompression de tampon de profondeurs qui permet d'obtenir en temps réel une image de profondeur de la scène complexe en fonction de la position de l'observateur sur la trajectoire. Ce tampon de profondeur reconstruit nous sert de base pour l'interpolation de points de vue.

Dans la seconde section, nous présentons l'application de la méthode d'interpolation de points de vue à un maillage simplifié du modèle complexe. L'utilisation d'un maillage simplifié nous permet d'étendre la méthode à un point de vue arbitraire. Nous présentons nos résultats ainsi que les travaux en cours et futurs.

4.2 Interpolation pour une trajectoire fixée

4.2.1 Principe

Dans ce partie, nous présentons une nouvelle méthode d'interpolation de points de vue [PGD04] permettant d'effectuer le rendu en temps réel d'une scène complexe pour un point de vue résidant sur un chemin 3D positionné par l'utilisateur (figure 4.1).

La différence principale avec les méthodes existantes est que nous ne déformons pas les images de référence, pas plus que nous n'effectuons de mélange de couleurs. Au lieu de cela, nous retrouvons le meilleur point stocké dans les images de référence pour chaque pixel affiché.

Le rendu se décompose en deux passes distinctes :

- Durant la première passe, nous reconstruisons la surface approchée de la scène originelle (le tampon de profondeur), vue selon un point de vue résidant sur le chemin 3D.
- Durant la seconde passe, nous appliquons notre technique d'interpolation de points de vue de façon à reconstruire l'apparence de la scène (couleur et relief).

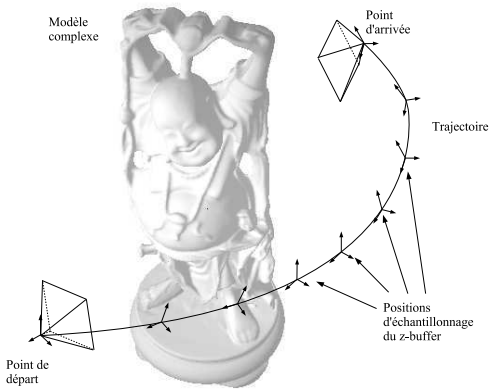


FIG. 4.1 – Trajectoire de la caméra.

La méthode repose sur la reconstruction du tampon de profondeur de la scène pour un point de vue résidant sur un chemin 3D pré-positionné par l'utilisateur. L'orientation de la caméra, quand celle-ci se déplace sur la trajectoire, est également enregistrée. Comme nous l'avons montré plus haut, le tampon de profondeur représente le relief de la scène pour une position de caméra donnée. Dans nos expérimentations, la définition du chemin s'effectue en positionnant une caméra de départ et une caméra d'arrivée orientées vers le centre de la scène (figure 4.1). Le chemin est alors défini par toutes les positions de caméra intermédiaires interpolées en coordonnées sphériques. Ceci ne constitue pas une restriction à la méthode mais nous permet de créer une trajectoire facilement : la position et l'orientation de la caméra sont définies en fonction de l'abscisse curviligne de l'observateur, entre $t = 0$ et $t = 1$.

4.2.2 Simplification de la surface

La reconstruction du tampon de profondeur de la scène à une position donnée est basée sur la variation de profondeur affectant chacun des pixels de la caméra, quand celle-ci se déplace le long de la trajectoire. Considérant un pixel à l'écran, nous représentons sa variation de profondeur entre l'instant $t = 0$ (départ) et $t = 1$ (arrivée) et nous créons une structure approximant au mieux cette variation.

4.2.2.1 Acquisition des échantillons

Pour obtenir cette variation de profondeur, nous effectuons le rendu de la scène complexe pour un nombre fixé de positions sur la trajectoire et en stockant à chaque fois le tampon de profondeur. Nous discrétisons donc la trajectoire en un nombre arbitraire de points, plus il y en aura, meilleure sera l'approximation des variations de profondeurs.

Considérant un pixel isolé, le principe de la méthode est d'approximer sa variation de profondeur en un ensemble restreint d'intervalles de variation linéaire (des segments).

4.2.2.2 Compression

Le nombre d'intervalles de variation linéaire est fixé de façon à contrôler l'espace mémoire nécessaire. Le problème est donc de trouver la meilleure façon de répartir un nombre fixé de segments pour approximer au mieux la variation réelle de profondeur. En figure 4.2, nous avons représenté la position de la caméra en abscisse et la profondeur du pixel sur les ordonnées.

L'algorithme que nous avons créé est adaptatif : nous commençons par approximer l'ensemble des points par un seul segment joignant le point de la profondeur de départ au point de la profondeur d'arrivée. Ensuite, le segment est divisé en deux à l'abscisse présentant la plus grande différence de profondeur entre la profondeur réelle et le segment (figure 4.2(a)).

Ensuite, l'algorithme est appliqué aux deux nouveaux segments créés. Par la suite, nous subdivisons, parmi un ensemble donné de segments, celui qui génère le point de plus grande différence entre la profondeur réelle et la profondeur linéairement approximée.

Durant la segmentation, nous supprimons tous les segments montrant une disparité de profondeur, c'est à dire ceux qui relient deux points consécutifs sur l'axe des abscisses (figure 4.2(b)). Cette disparité de profondeur correspond à une brutale variation de profondeur due à une bordure dans l'image (voir la figure 2.21 page 76 de la section 2.3.4 sur l'*interpolation de points de vue* par pixel).

Quand nous atteignons le nombre fixé de segments, l'algorithme se termine.

Finalement, nous obtenons un liste ordonnée de segments pour chacun des pixels. Chaque segment correspond à un intervalle de déplacement de la caméra et est constitué des informations suivantes :

$$\begin{cases} \text{une profondeur de départ} & d_0 \\ \text{une profondeur d'arrivée} & d_1 \\ \text{l'abscisse du point d'arrivée} & T \end{cases}$$

Les segments sont stockés dans l'ordre des T croissants, c'est à dire, comme le montre la figure 4.2(c), de gauche à droite.

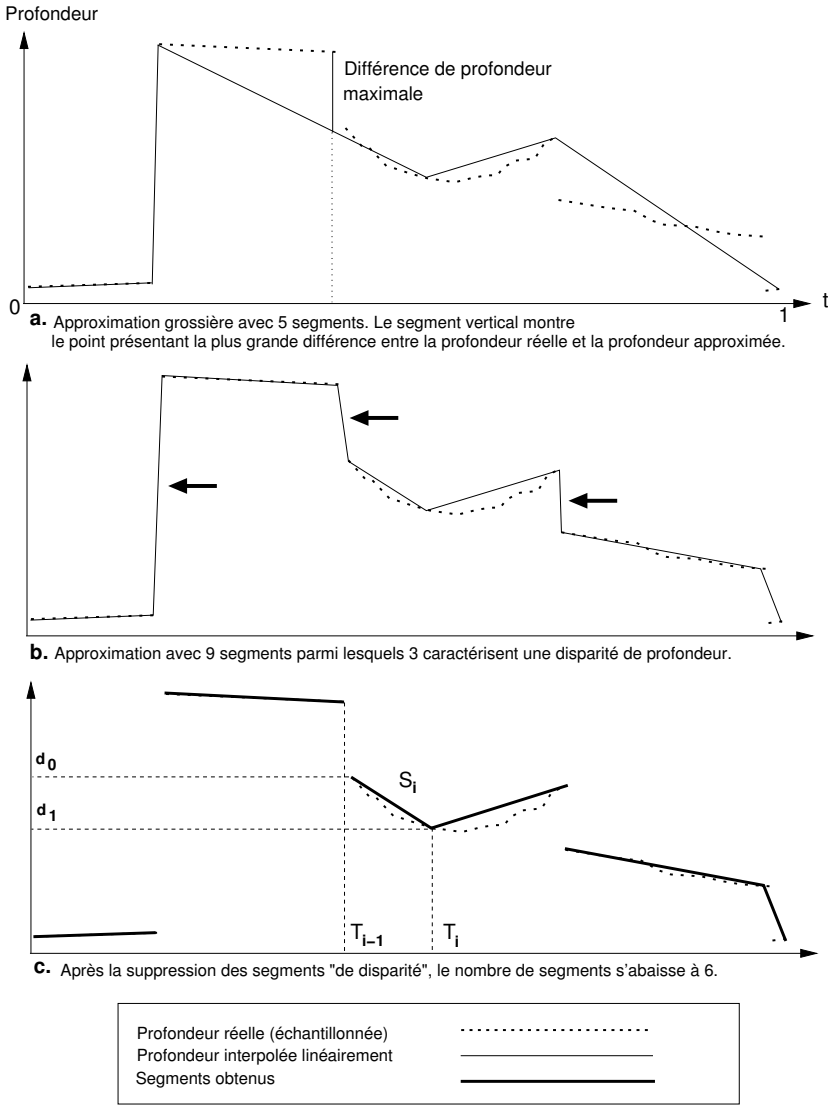


FIG. 4.2 – Segmentation de la variation de profondeur d'un pixel de l'écran. Abscisses : abscisse curviligne de la caméra sur la trajectoire. Ordonnées : profondeur du pixel.

4.2.2.3 Décompression

La décompression est l'opération qui permet de retrouver la profondeur d'un pixel de l'écran à un instant t donné de la trajectoire. Pour déterminer cette profondeur, nous devons simplement trouver le segment S_i qui se termine avec $T_i > t$, en partant du début de la liste. A partir des données de profondeur de départ et d'arrivée stockée par le segment, la profondeur d du point est obtenue avec une simple interpolation linéaire :

$$d = d_0 + \frac{t - T_i}{T_i - T_{i-1}}(d_1 - d_0)$$

Finalement, ceci nous permet de retrouver la profondeur de chaque pixel, pour un instant donné du

mouvement de la caméra le long de la trajectoire. Pour obtenir la profondeur de l'image complete, nous effectuons ce parcours de liste pour chaque pixel.

4.2.3 Reconstruction du relief et de la couleur de la surface

Pour effectuer le rendu complet, nous devons maintenant calculer la couleur des pixels, et corriger l'erreur que nous avons commise en approximant la variation de profondeur des points. Ceci est effectué en extrayant ces informations des images de référence de la scène complexe de base.

4.2.3.1 Acquisition des points de vue de référence

Un petit nombre d'images de référence de la scène complexe est pris durant la phase de précalculs. Ces points de vue de référence stockent d'une part les couleurs de la scène, mais également la profondeur des points (le tampon de profondeur) ainsi que les normales de la surface, soit trois textures :

1. Les couleurs sont obtenues en affichant le modèle sans gestion de l'éclairement, de façon brute.
2. Les normales sont obtenues de la même façon que la méthode exposée en section 2.2.4.2, c'est à dire par substitution des couleurs des sommets du maillage par leur normales, et affichage du modèle.
3. La profondeur des points correspond au tampon de profondeur de l'image.

Ainsi, un *point de vue* de référence est constitué des informations suivantes :

une texture de couleurs	RGB sur 8 bits
une texture de profondeur	sur 16 bits
une texture de normales	XYZ sur 8 bits (coordonnées globales)
la matrice de transformation de la caméra	4x4 flottants
la matrice de transformation inverse	"

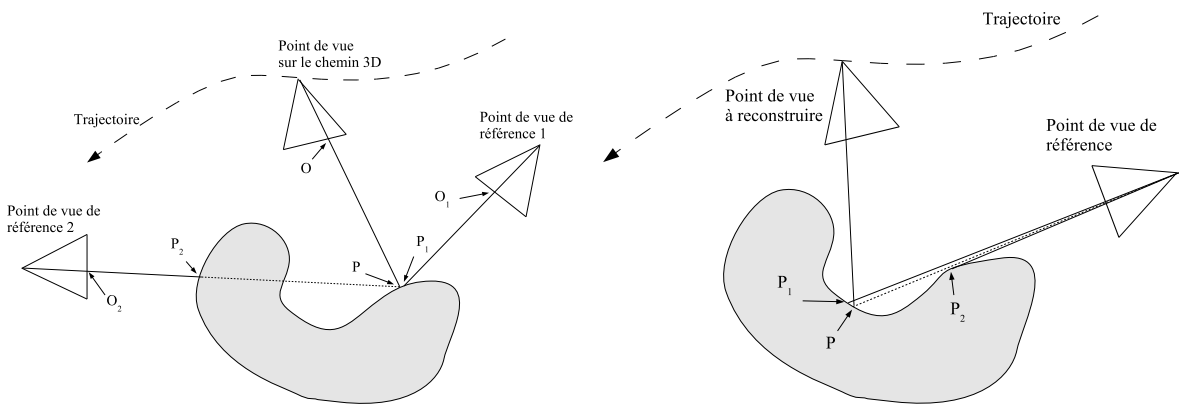
Les points de vue sont positionnés par l'utilisateur, de façon à couvrir le plus possible la surface de la scène. Ils ne sont pas obligatoirement positionnés sur la trajectoire. Le but idéal serait de disposer d'une méthode automatique de placement de point de vue permettant d'obtenir un recouvrement maximal, mais le placement à la main n'est pas très contraignant et en pratique, quelques points de vue suffisent à couvrir la scène de façon satisfaisante.

4.2.3.2 Interpolation des points de vue de référence

L'interpolation s'effectue au niveau du pixel, avec le concours de la carte graphique. Etant donnée la profondeur du point calculée avec la segmentation précédemment décrite, l'algorithme permettant d'extraire le meilleur pixel parmi les pixels des points de vue de référence se déroule comme suit.

Etant donné un point $O(u, v)$ à l'écran, de profondeur segmentée p_O, nous le transformons dans le repère de la scène, ce qui donne le point 3D P (figure 4.3(a)).

Ce point est projeté sur le plan de projection des points de vue de référence. Nous obtenons donc un ensemble de points 2D sur les caméras de référence (O_1 et O_2 sur la figure). Chacun de ces points



(a) Interpolation de deux points de vue.

(b) Disparité de profondeur.

FIG. 4.3 – Principe de la reconstruction de la surface.

correspond, par rétroprojection, à un point 3D dans l'espace de la scène (P_1 et P_2). Parmi ces points 3D, le meilleur est celui qui est le plus proche de P. Dans l'exemple, le point de vue 1 est le meilleur point de vue pour le pixel O : la distance entre P_1 et P est inférieure à la distance entre P_2 et P. Nous attribuons donc au pixel O les attributs stockés dans les textures du point de vue 1, aux coordonnée 2D de O_1 : couleur, normale et profondeur.

En pratique, à cause des limitations en taille des shaders, nous n'effectuons les comparaisons de cette façon qu'avec au maximum trois des plus proches points de vue par rapport à la caméra virtuelle (sur la trajectoire).

4.2.4 Rendu

Le rendu en temps réel s'effectue donc en deux passes : la première (section 4.2.2.2), effectuée par le processeur central, consiste à reconstruire le tampon de profondeur approché de la scène réelle à la position courante de la caméra. La seconde passe (section 4.2.3.2), effectuée intégralement par la carte graphique, consiste à plaquer sur ce relief approché les couleurs de la scène de base ainsi qu'à corriger la profondeur segmentée avec les informations sur le relief original stockées dans les images de référence.

La reconstruction du tampon de profondeur n'intervient que quand la position de la caméra a changé, dans le cas contraire, nous ne recalculons que l'éclairement.

4.2.4.1 Affichage

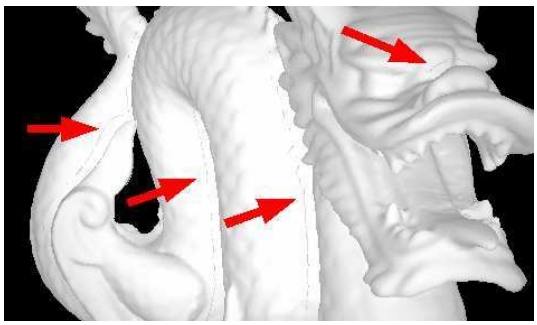
Etant donnés le tampon de profondeur reconstruit pour le point de vue virtuel et les points de vue de référence, le rendu s'effectue en affichant un rectangle occupant tout l'écran, et dont les coordonnées de texture des sommets sont $(0, 0)$ pour le coin haut/gauche et $(1, 1)$ pour le coin bas droit. Cette paramétrisation nous permet de localiser un pixel dessiné à l'écran au sein du pixel shader. Les données transmises au pixel shader sont la texture de profondeurs reconstruite et les textures des points de vue

de référence (profondeurs, couleurs, normales). Par exemple, avec 3 points de vue de référence, il faudrait transmettre 10 textures. En codant la profondeur (une seule composante) dans la composante alpha de la texture de couleurs, nous abaissons ce nombre à 7. Les matrices de transformation des points de vue de référence et de la caméra doivent également être transmis, ainsi que les matrices inverses correspondantes.

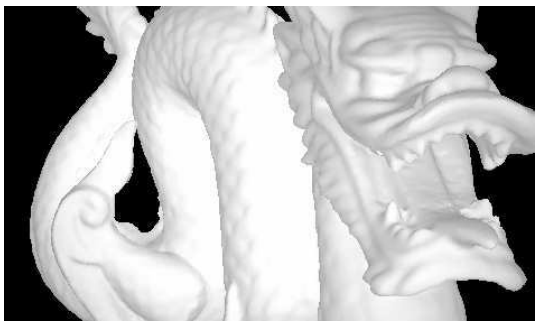
4.2.4.2 Disparités de profondeur

Durant nos expérimentations, nous avons noté des artéfacts visuels se produisant sur les bordures de l'objet (figure 4.4(a)). La figure 4.3(b) détaille le problème : quand P est projeté sur le point de vue de référence, sa rétroprojection dans l'espace de la scène donne le point P_2 au lieu de P_1 à cause de la discrétisation du plan de projection de la caméra de référence. Cette discrétisation correspond à la définition de la texture, mais l'artefact subsiste quelle que soit la finesse de la définition.

Pour résoudre ce problème, la profondeur du point à rétroprojeter n'est plus issue uniquement d'une seule profondeur, mais de l'interpolation bilinéaire des profondeurs des texels adjacents. L'image corrigée est exposée en figure 4.4(b).



(a) Artféfacts dûs à la disparité de profondeur entre deux pixels d'une bordure.



(b) Avec interpolation bilinéaire des profondeurs.

FIG. 4.4 – Disparités de profondeur.

4.2.4.3 Algorithme multi-passes

A cause de la limitation de la taille des shaders, nous ne pouvons pas utiliser un nombre quelconque de points de vue à interpoler en une seule fois. En pratique, nous avons développé deux méthodes :

1. La première est utilisée si la scène est bien couverte avec trois points de vue. L'algorithme est directement celui décrit précédemment.
2. La seconde est utilisée si il est nécessaire d'utiliser plus de trois points de vue. Le rendu s'effectue en plusieurs passes, une par point de vue de référence.

Dans le second cas, une passe consiste à calculer la distance entre le point P et le point rétroprojeté issu du point de vue de référence (voir figure 4.3(a)) et à retourner cette distance dans le tampon de

profondeur de la carte graphique, en même temps que la couleur calculée. Cette opération est effectuée pour chaque point de vue, c'est à dire que nous affichons autant de fois le polygone occupant tout l'écran qu'il y a de points de vue. La fonction de comparaison du tampon de profondeur est choisie de façon à n'autoriser l'affichage que des points de "profondeur"[33] inférieure à la profondeur stockée précédemment. Ainsi la couleur restante est celle associée au point de vue donnant la plus petite distance (P, P_i), ce qui est le but recherché.

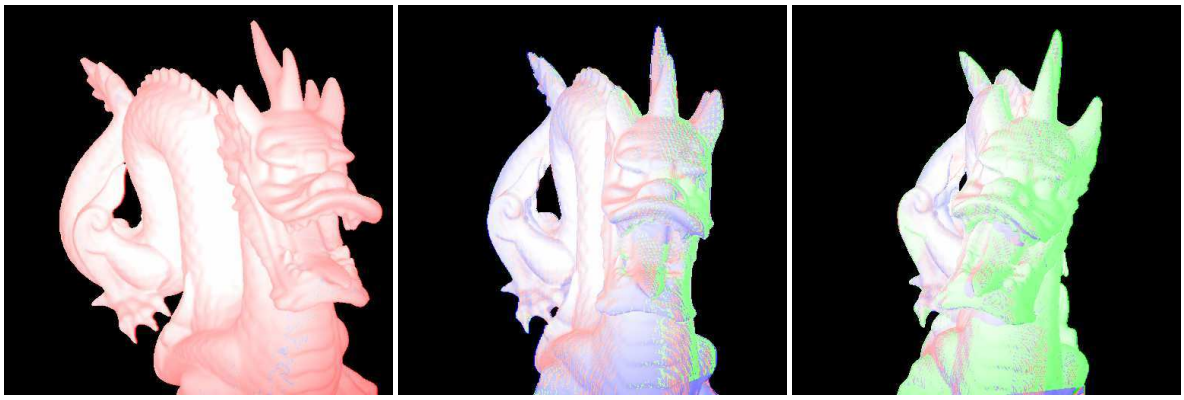


FIG. 4.5 – Origine des points utilisés pour reconstruitre les pixels de l'image.

En pratique, à cause de la précision limitée du tampon de profondeur de la carte graphique, nous n'utilisons pas directement la distance calculée en 3D. A la place, nous reprojetons les points P et P_i et calculons leur distance en 2D, sur le plan de la caméra de référence.

La figure 4.5 montre la scène reconstruite à différentes positions du déplacement de la caméra. Dans cette image, nous avons utilisé 3 points de vue de référence, mis en évidence avec 3 couleurs différentes : points rouges pour le premier, verts et bleus pour le second et le troisième. Ainsi, on peut visualiser le point de vue qui a servi à calculer un pixel de l'image.

4.2.5 Résultats

La figure 4.6 montre le résultat de la méthode pour une scène composée du Buddha du Laboratoire d'Informatique Graphique de Stanford. Tous nos tests ont été faits avec une résolution d'écran et de textures de 512x512 pixels. Le nombre de tampon de profondeurs utilisés pour l'échantillonnage de la trajectoire est de 150, ce qui s'est avéré largement suffisant pour capturer toutes les variations de profondeur pour un tour complet autour de l'objet. Avec ces résolutions, le temps de précalcul (acquisition et segmentation) est d'environ 1 minute pour 15 segments. La taille en mémoire de la structure représentant les tampons de profondeurs est présentée dans la table 4.1.

La table 4.2 détaille les caractéristiques des modèles que nous avons utilisés. La table 4.3 présente les taux d'affichage que nous obtenons en utilisant un nombre croissant de points de vue de référence. La seconde colonne correspond au taux d'affichage sans utiliser de compression des tampon de profondeurs, la colonne 3 avec compression sur 15 segments, et la dernière colonne en utilisant l'interpolation bilinéaire des profondeurs.

[33] Correspondant en fait à la distance (P, P_i).

Sans compression (150 z-buffers)	75 Mo
15 segments	20 Mo
8 segments	10 Mo
6 segments	5 Mo

TAB. 4.1 – Taille de la structure selon différentes précisions pour la segmentation.

(a) Modèle originel

(b) Image reconstruite sans segmentation des profondeurs.

(c) Segmentation sur 12 segments.

(d) Compression sur 8 segments.

FIG. 4.6 – Résultats de l'interpolation.

Le temps de rendu est en pratique indépendant de la complexité géométrique et dépend essentiellement de la bande passante maximale entre la mémoire vidéo et le GPU. La seconde colonne illustre ce point : avec un seul point de vue il y a un faible nombre de lectures en mémoire vidéo (adressage de texels). Plus le nombre de points de vue augmente, plus le nombre de lectures augmente et fait baisser les performances. Le pire cas étant d'utiliser l'interpolation bilinéaire, qui multiplie par 4 le nombre de lectures, faisant baisser le taux d'affichage en dessous de 10 FPS (NVIDIA GeForceFx 5600). Les deux dernières lignes montrent les résultats obtenus avec une ATI Radeon 9700 Pro et la bibliothèque graphique DirectX 9 (Pixel Shaders v.2). Cette carte est plus puissante et donne de meilleurs taux d'affichage, mais est très limitée en termes de longueur de shaders et empêche l'utilisation de l'interpolation bilinéaire des profondeurs. De plus, nous ne pouvons pas utiliser plus de deux points de vues de référence en même temps, ce qui nous oblige à utiliser l'algorithme multi-passes.

La figure 4.7 montre le résultat de l'interpolation sur un tore à géométrie perturbée en utilisant différents nombres de segments pour la compression. A droite de chaque image nous avons comparé pixel

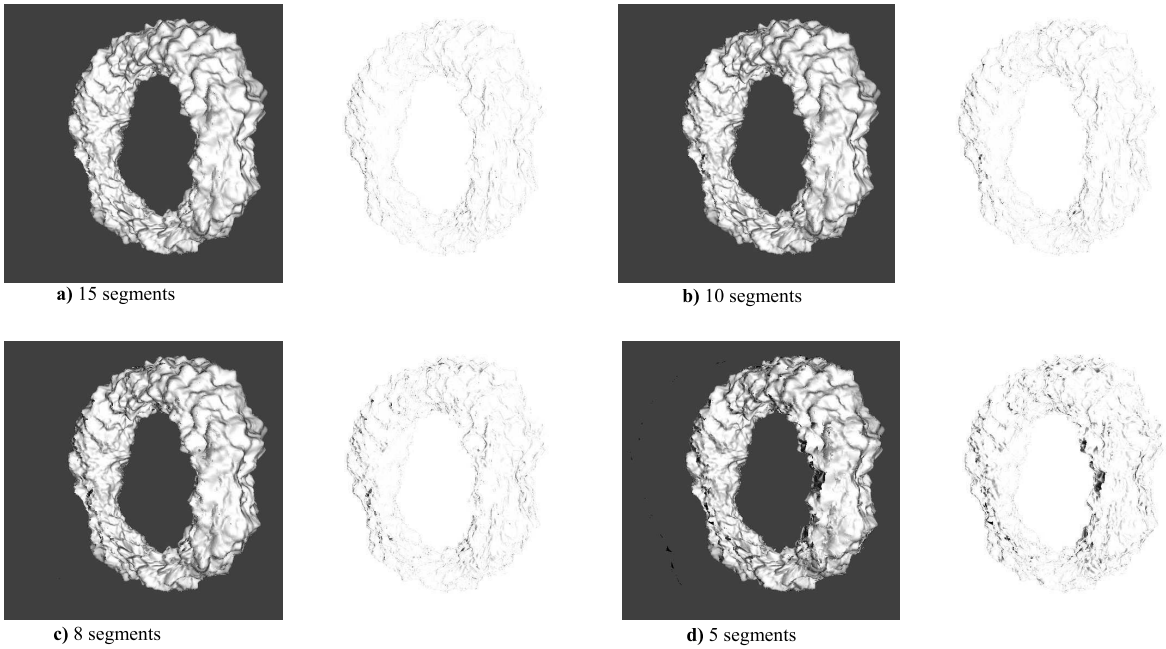


FIG. 4.7 – Comparaisons de la méthode avec différentes compressions.

par pixel la différence visuelle avec le rendu de l'objet réel. L'image résultante représente, sur une échelle en niveaux de gris, la différence entre la couleur de l'image reconstruite et l'image originelle, plus les pixels sont noirs, plus la différence est importante.

Modèle	**Triangles**	**Taille (Mo)**	**FPS**
Armadillo	345944	30.5	4.5
Tore	250000	11	10
Buddha	1087716	95	2
Dragon	871414	76.5	2.5

TAB. 4.2 – Caractéristiques des modèles que nous avons utilisés. Le taux d'affichage correspond au rendu conventionnel du modèle avec une GeForceFx 5600, sans optimisations spécifiques.

Nb. points de vue	1	2	3	3(ATI)	4(ATI)
Sans compression	80	22	14	80	75
Compression	16	16	13	30	25
Compr.+interp. bilinéaire	15	10	6	o	N.A.

TAB. 4.3 – Taux d'affichage obtenus (compression avec 15 segments).

4.2.6 Discussion

Dans cette partie nous avons présenté une méthode permettant d'effectuer le rendu interactif d'un modèle complexe de façon presque indépendante de sa complexité géométrique. En effet, celle-ci se retrouve exprimée dans la profondeur segmentée des pixels, et bien que notre algorithme de segmentation se soit avéré suffisamment performant pour les scènes standard que nous avons utilisées, il serait mis en échec par un modèle intriqué tel qu'un arbre. Malgré ce handicap, nous obtenons de bons résultats en termes de temps d'affichage. Dans nos tests, le modèle nécessitant le plus grand nombre de segments est le modèle "Armadillo" (figure 4.8 page 116) : pour une trajectoire décrivant un cercle complet autour du modèle, 30 segments sont nécessaires.

Malgré l'utilisation de la carte graphique, le goulot d'étranglement de la méthode est la décompression, qui est calculée par le processeur central. Du fait de représentation en liste de segments, le temps de décompression est plus long vers la fin de la trajectoire ($t = 1$) car il est nécessaire de parcourir toute la liste. La variation du taux d'affichage atteint au maximum 5 FPS (AMD Athlon 1.4Ghz).

L'interpolation de points de vue, elle, est très peu gourmande en espace mémoire (quelques textures) et s'effectue intégralement avec l'aide de la carte graphique.

Les avantages et inconvénients de la méthode sont résumés ci-dessous.

4.2.6.1 Avantages

- simplicité de la méthode : les données consistent simplement en un ensemble d'images du modèle.
- l'indépendance du temps d'affichage vis à vis de la complexité géométrique
- le qualité visuelle du résultat
- support de l'éclairement variable, bien que les projections d'ombres ne soient pas possibles : on ne connait la géométrie de la scène que pour des caméras positionnées sur la trajectoire, donc il est impossible de projeter l'ombre du modèle si la lumière (supposée ponctuelle) est positionnée à l'extérieur de celle-ci.
- possibilité d'intégration dans une scène existante rendue de façon conventionnelle : l'image générée contient toutes les informations de profondeur nécessaires pour une intégration dans un moteur de rendu basé sur un tampon de profondeur pour gérer la visibilité.

4.2.6.2 Inconvénients

- l'espace mémoire occupé pour stocker la variation de profondeur des pixels
- les points de vues autorisés sont fixés
- les points de vue de référence doivent être positionnés à la main, ce qui dans certains cas peut s'avérer délicat

4.2.6.3 Perspectives

De nombreux points pourraient être améliorés, dont notamment la segmentation des profondeurs. Il est envisageable de créer un algorithme de segmentation qui se baserait sur un seuil d'erreur visuelle pour subdiviser un segment. Ceci permettrait de ne pas subdiviser systématiquement les segments à chaque étape et de créer de plus grandes portions linéaires aux meilleurs endroits. La mémoire en

serait d'autant réduite.

Concernant la liste de segments, il serait envisageable de l'organiser en une structure hiérachique permettant d'en accélérer l'accès. Un arbre binaire serait optimal pour cette tâche. Dans ce cas, le problème à traiter serait principalement l'espace mémoire nécessaire pour stocker un arbre par pixel.

Par ailleurs, il est possible d'augmenter le degré de liberté de la caméra virtuelle :

1. En autorisant un changement d'angle de vue bien qu'en restant sur la trajectoire : si la scène complète est vue d'un point donné, nous pourrions considérer le plan de projection de la caméra (pour lequel on reconstruit l'image) comme un plan 3D et l'afficher dans l'espace de la scène, comme un *billboard*[34]. L'image obtenue serait exacte, tant que l'observateur ne s'écarte pas de la trajectoire.
2. Il est possible de s'éloigner de la scène, dans l'axe de la direction de la caméra : le plan de projection originel serait également affiché dans l'espace de la scène comme un *billboard*.

D'autre part, les segments, représentés dans l'espace de la scène, pourraient être reliés de façon à créer une surface 3D, correspondant finalement à un modèle simplifié du modèle de base. Combiné à notre façon de repérer une disparité de profondeur (paragraphe 4.2.2.2 page 107), la surface reconstruite pourrait être séparée en différentes couches, et nous éviterions ainsi les effets d'étirement (*rubber-sheet*) dans le maillage.

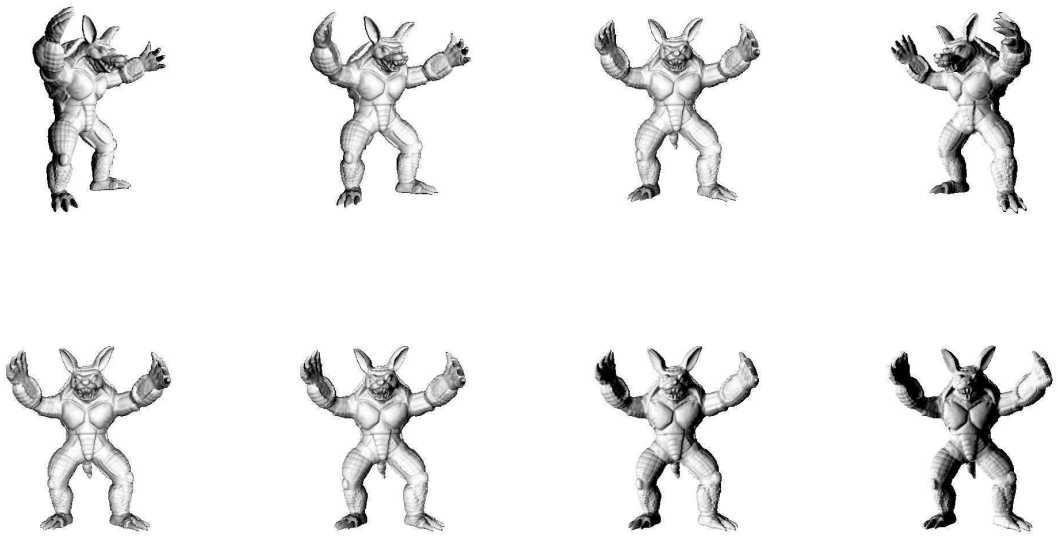


FIG. 4.8 – Application de la méthode au modèle *Armadillo*, compression sur 10 segments. Ligne du haut : interpolation à différentes position de la trajectoire. Ligne du bas : variation de la position de l'éclairage.

[34] Voir la partie 1.4.2.1 page 36.

4.3 Placage de relief en temps réel

4.3.1 Principe

Dans cette partie, nous présentons une extension de la méthode précédente consistant à utiliser un maillage simplifié comme base pour l'interpolation de points de vue [PDG04]. C'est à dire qu'au lieu de reconstruire de tampon de profondeur de la caméra virtuelle, nous affichons uniquement un maillage simplifié du modèle et utilisons le tampon de profondeur de l'image obtenue pour appliquer l'interpolation.

Dans nos expérimentations, le maillage simplifié est un maillage statique obtenu avec l'algorithme de simplification de Hoppe[Hop96], mais rien n'empêche d'utiliser un maillage progressif car la méthode ne se base que sur la profondeur des pixels affichés.

Principalement, la méthode se décompose en deux parties :

1. L'acquisition des points de vue de référence, pour laquelle nous fournissons à l'utilisateur une aide visuelle.
2. Le rendu, qui s'effectue en affichant le maillage simplifié sur lequel nous "reprojetons le relief" stocké dans les points de vue de référence.

4.3.2 Acquisition des points de vue de référence

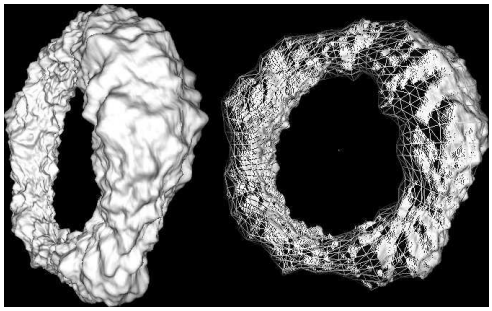
Pour aider l'utilisateur à trouver le meilleur ensemble de points de vue, nous affichons le nuage de points issu de chacune des images qu'il capture. Pratiquement, l'utilisateur navigue autour de l'objet simplifié dessiné en mode "fil-de-fer", puis il sélectionne un endroit qu'il estime le mieux recouvrir la surface de l'objet (figure 4.9(a)). Nous effectuons l'affichage du modèle complexe pour cette position, puis nous ajoutons l'ensemble des points rétroprojetés de l'image à l'ensemble des points déja affichés. Ainsi la surface de l'objet complexe de base se reconstruit graduellement par dessus le modèle simplifié.

Ceci permet de répartir les points de vue de façon à maximiser la couverture de la surface pour chaque point de vue, et permet également de contrôler visuellement l'échantillonnage des images (figure 4.9(b)).

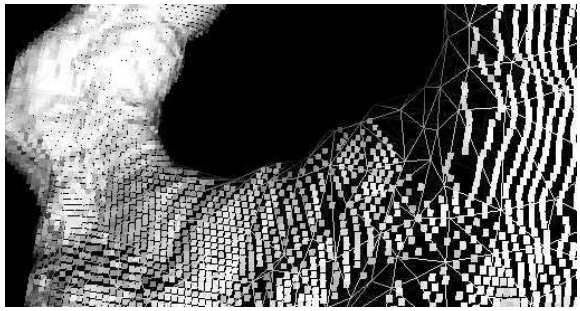
4.3.3 Rendu

La rendu s'effectue en affichant directement le modèle simplifié, tout l'algorithme étant implanté dans le pixel shader qui calcule la couleur et la profondeur de chacun des points affichés en une seule passe.

La figure 4.10(a) illustre l'algorithme : P représente un point 3D de la surface du modèle simplifié qui se projete sur le pixel O de l'écran de la caméra virtuelle. P est une approximation du point réel de la surface du modèle complexe, P'. Pour trouver le meilleur point 3D stocké dans les points de vue de référence (V_1 et V_2 sur la figure), nous projetons P sur les plans de projection des caméras virtuelles. Nous obtenons les points 2D O_1 et O_2 . Chacun de ces points correspond, par rétroprojection, à un point 3D dans l'espace de la scène : P_1 et P_2 respectivement. Parmi ces points, celui qui approche le mieux P' est celui pour lequel la distance P_i/P' est la plus petite : P_2 sur la figure. Finalement, nous assimilons P' à P_2 c'est à dire que nous calculons la couleur du pixel O avec les informations de couleur, normale et profondeur du point O_2 .

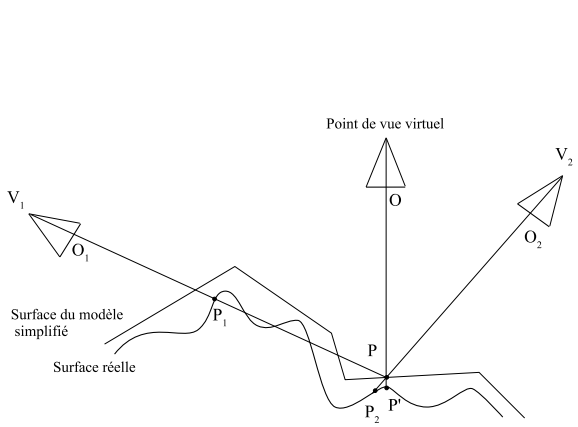


(a) Nuage de points issu d'une capture de l'objet complexe (image de gauche) observé selon un autre point de vue (image de droite)

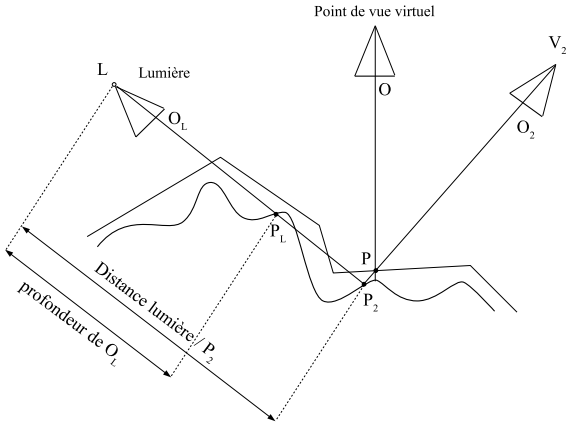


(b) Vue rapprochée du nuage de points capturé (figure (a)). Les rectangles blancs correspondent à la rétroprojection des pixels de l'image capturée.

FIG. 4.9 – Acquisition des points de vue de référence.



(a) Calcul du meilleur point de vue.



(b) Calcul de l'auto-ombrage.

FIG. 4.10 – Principe du placage de géométrie sur le modèle simplifié.

4.3.3.1 Calcul de l'auto-ombrage

Pour calculer l'auto-ombrage, nous avons utilisé la méthode de Williams[Wil78]. Le principe est de déterminer si un point affiché est ou non dans l'ombre en comparant sa distance par rapport à la source de lumière avec la distance stockée dans le tampon de profondeur de la scène vue selon la position de la lumière.

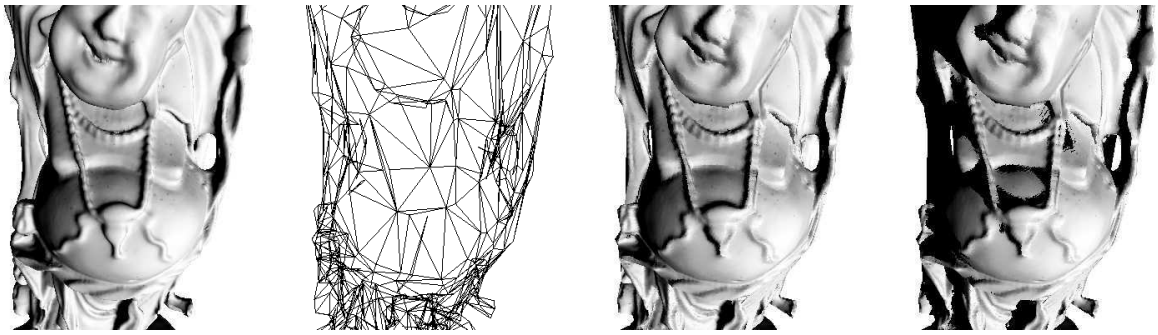
Ceci nécessite d'effectuer le rendu du modèle simplifié à deux reprises : une selon la position de la lumière et une selon la position de la caméra.

La figure 4.10(b) présente l'algorithme. L'étape de calcul du meilleur point de vue pour P a donné le point P_2. Nous le projetons sur le plan de la caméra associée à la lumière : on obtient le point 2D O_L. Nous rétroprojetons O_L dans l'espace de la scène à l'aide du tampon de profondeur de la caméra

de la lumière, ce qui nous donne le point P_L dans l'espace de la scène. Ensuite, nous comparons la distance L/P_2 à la distance L/P_L . Si la différence de profondeur est supérieure à une petite valeur fixée, nous considérons qu'une partie du relief cache le point P_2 de L (la lumière). C'est le cas dans notre exemple : P_2 (et donc P) est dans l'ombre.

4.3.4 Résultats

La figure 4.11 présente le résultat de la méthode appliquée au modèle du Buddha. Nous avons dans ce cas utilisé un maillage simplifié composé de 2000 triangles (sous-figure (b)). Le résultat de la méthode est visible en sous-figure (c). Dans la sous-figure (d) nous avons ajouté l'auto-ombrage aux calculs. Comme le montre la figure 4.12(d) l'ombrage des pixels étant calculé à partir des profondeurs extraites des images de référence, nous reconstruisons "réellement" le relief du modèle originel.



(a) Le modèle original (1,1 million de faces) - 25 FPS.

(b) Le modèle simplifié (2000 faces).

(c) Rendu du modèle simplifié avec notre méthode - 160 FPS.

(d) = (c) + prise en compte de l'auto-ombrage - 103 FPS.

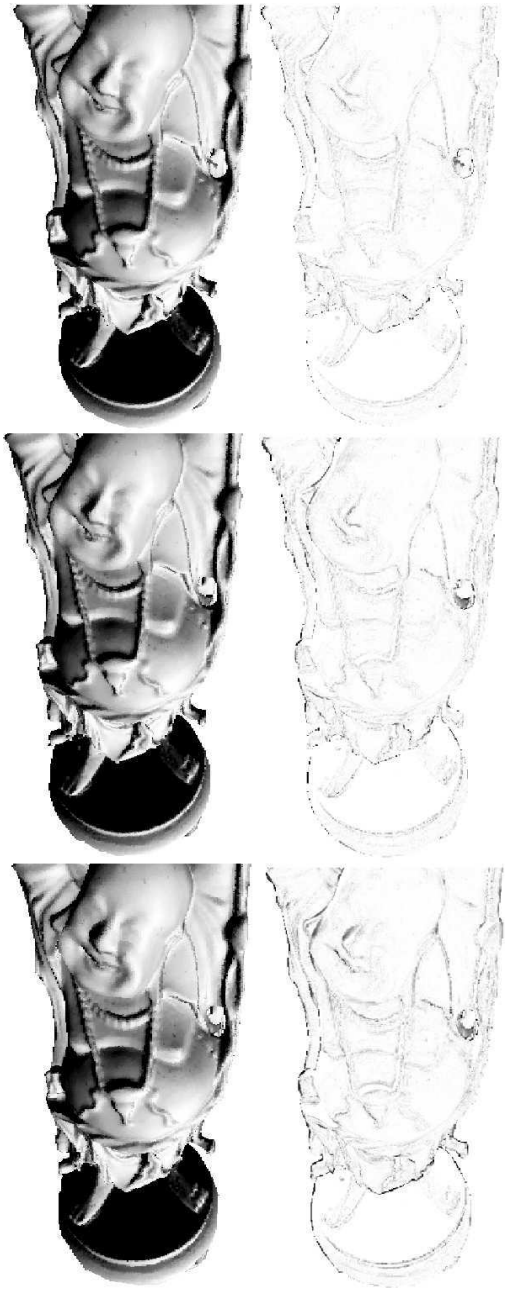
FIG. 4.11 – Application de notre méthode d'interpolation de points de vue au modèle du Buddha, composé de 1.1 million de faces.

Dans la figure 4.12(a) nous avons comparé visuellement le résultat de notre algorithme appliqué à différents maillages simplifiés. Comme on peut le voir, les points pour lesquels la différence est maximale sont situés aux bordures du maillage simplifié. En effet, notre méthode ne permet pas de reconstruire le relief en bordure du maillage simplifié car nous nous basons sur celui-ci pour calculer la couleur des points. En revanche, pour les points à l'intérieur du maillage, la différence visuelle est indiscernable.

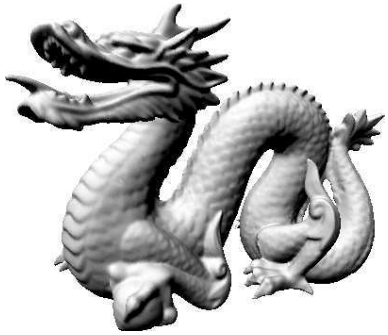
4.3.5 Discussion

La méthode donne de très bons résultats visuels, ainsi qu'un taux d'affichage très intéressant, supérieur à 100 images par secondes (GeForceFx 5950). Dans nos expérimentations, nous avons employé la méthode pour effectuer le rendu d'un seul modèle complexe. Néanmoins, rien n'empêche de l'employer pour une scène composée de multiples modèles complexes. Comme le temps de rendu ne dépend plus de la complexité géométrique[35], les performances de la méthode seraient grandement mises en avant.

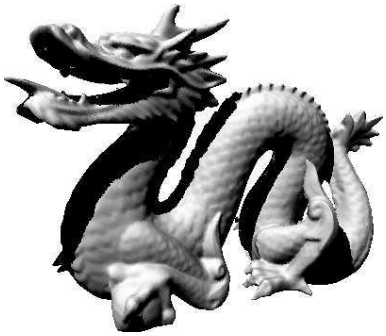
[35]Le rendu d'un maillage simplifié composé de moins de 10000 polygones est insignifiant par rapport au calcul des couleurs des points.



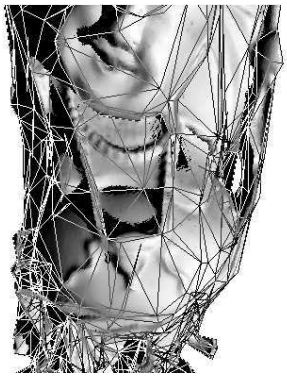
a) Colonne de gauche : le modèle reconstruit. Colonne de droite : la différence visuelle avec le modèle de base. De haut en bas, le maillage simplifié est composé de : 12K triangles, 6K triangles et 3K triangles.



b) Modèle original : 800K triangles, 30 FPS



c) Modèle reconstruit : 10K triangles, 100 FPS



d) Vue rapprochée de la projection d'ombre. Comme on peut le voir, la bordure de l'ombre est courbée au sein des triangles car on reconstruit la profondeur « réelle » de la surface.

FIG. 4.12 – A gauche, résultats de la méthode avec différents modèles simplifiés. A doite, application de la méthode au modèle de dragon. En bas à droite, vue détaillée de l'application de l'auto-ombrage.

Finalement, nous résumons ci-dessous les avantages et inconvénients de la méthode.

4.3.5.1 Avantages

Par rapport à la méthode précédente basée sur une reconstruction du tampon de profondeur de l'image, nous n'avons pas de restrictions sur la position du point de vue : ils peuvent être quelconques autour de l'objet, aux problèmes d'échantillonnage près, détaillés dans les inconvénients.

De plus, les résultats sont très bons en termes de rapidité d'affichage, de qualité visuelle et d'espace mémoire : seules quelques textures sont nécessaires pour stocker les points de vues.

La mise en oeuvre de la méthode est très simple car, comme dans la méthode précédente, il ne faut effectuer que l'affichage du modèle complexe et stocker le résultat dans des textures.

Comme le rendu consiste simplement à afficher un maillage simplifié, l'intégration de la méthode est immédiate dans un moteur de rendu conventionnel.

La méthode permet d'employer les modèles d'éclairement habituels en rendu en temps réel, notamment le modèle de Phong, ainsi que la projection d'ombre.

4.3.5.2 Inconvénients

Le problème visuel le plus remarquable peut se voir sur les bords de l'objet : la silhouette obtenue est celle du modèle simplifié, non celle du modèle de base. Ceci crée des bordures franches qui peuvent être très visibles si l'on se rapproche du modèle.

Par ailleurs, au niveau de l'algorithme lui même, nous avons certaines restrictions. Tout d'abord, la méthode ne fonctionne correctement que si la surface du maillage simplifié n'est pas trop éloignée du modèle originel. Par exemple, il nous est impossible d'appliquer cette technique avec un maillage simplifié qui serait une simple boîte englobante car les erreurs commises seraient trop grandes (voir la figure 4.10(a) : l'erreur commise correspond au fait que le point idéal est le point P' et que nous choisissons le point P_2).

Par ailleurs, des calculs sont redondants : pour un point donné de la surface, le résultat est toujours le même, c'est à dire que le meilleur point de vue est toujours identique. Pourtant nous effectuons systématiquement ce calcul de visibilité.

Au niveau de l'extraction des points des textures de référence, nous ne traitons pas les problèmes de sous- et sur-échantillonnage. Ceci peut mener à des situations ou la surface reconstruite est "pixellisée" quand l'observateur se rapproche extrêmement de la surface reconstruite.

Le dernier point est que bien que la place nécessaire en mémoire vidéo soit faible, celle-ci n'est pas minimale car de grandes zones sont vides sur les textures des points de vue de référence : l'objet n'occupe qu'une petite partie de l'image.

4.3.5.3 Travaux futurs

La méthode pourrait être améliorée de plusieurs façons.

Concernant l'acquisition des points de vue de référence, un algorithme de positionnement automatique de caméra tel que [VFSH03] devrait permettre d'automatiser cette tâche. Concernant le choix des trois points de vue de référence parmi l'ensemble des points de vue, nous choisissons actuellement ceux dont les centres de projections sont les plus proches du centre de projection du point de vue

virtuel. Cette méthode est valide quand les points de vue sont tous orientés approximativement vers le centre de la scène. Néanmoins, dans le cas général, il faudrait utiliser une méthode permettant de prendre en compte une orientation arbitraire des points de vue.

Concernant les problèmes d'échantillonnage, ceux-ci devraient être corrigés, ou tout au moins réduits, en utilisant une interpolation bilinéaire des points projetés du maillage simplifié, dans les textures de référence. Nous pourrions pour cela utiliser le placage de texture projectif, intégré depuis longtemps dans les cartes graphiques. Pour cela, lors de la rastérisation d'un triangle du maillage simplifié, nous devons également balayer ce triangle dans les points de vue de référence, en prenant en compte l'orientation de la face par rapport au point de vue (figure 4.13).

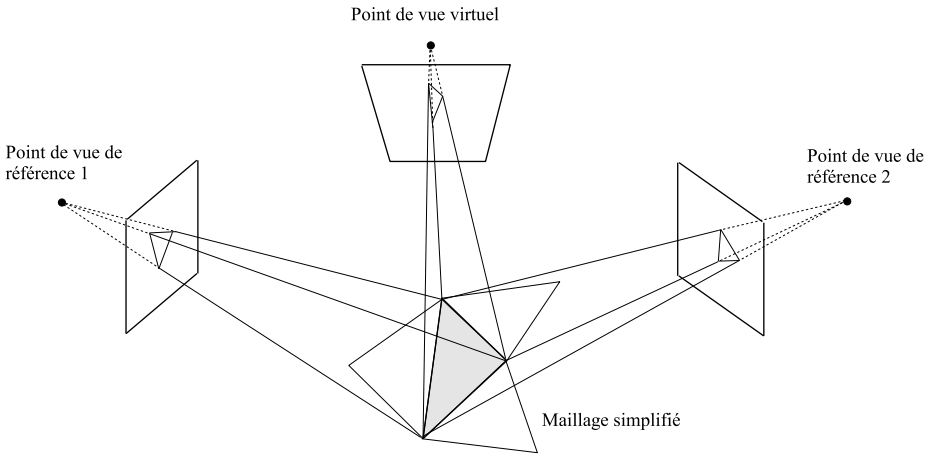


FIG. 4.13 – Améliorations. Plutôt que de reprojeter un à un les points affichés d'un triangle t du maillage simplifié (en gris) sur chaque point de vue de référence, nous pourrions obtenir ces points directement en pré-projetant t dans les images de référence.

C'est actuellement le fonctionnement de base du placage de texture implanté dans les cartes graphiques, ceci permet de corriger les effets de déformation dans une texture plaquée par exemple sur un triangle affiché selon un angle rasant. Les informations à ajouter par rapport à notre méthode sont simplement la profondeur des sommets par rapport aux points de vue de référence, ce qui devrait être facile à mettre en oeuvre.

Concernant le problème des bordures franches, une solution que nous sommes en train d'expérimenter consiste à échantillonner les rayons optiques de façon à ne plus simplement baser nos calculs sur un seul point P mais sur plusieurs points répartis à la suite de P (figure 4.14). Le problème est de répartir le mieux possible ces points de façon à minimiser la complexité du calcul et de converger rapidement vers le "bon" point (P' sur la figure).

Au niveau purement "technique", une amélioration immédiate des performances concerne la projection d'ombre : nous avons réimplémenté manuellement l'algorithme de Williams alors que celui-ci est spécifiquement implémenté comme une fonctionnalité "cablée" des cartes graphiques.

Un autre point serait d'évaluer la méthode sur une scène complète, composée de multiples modèles

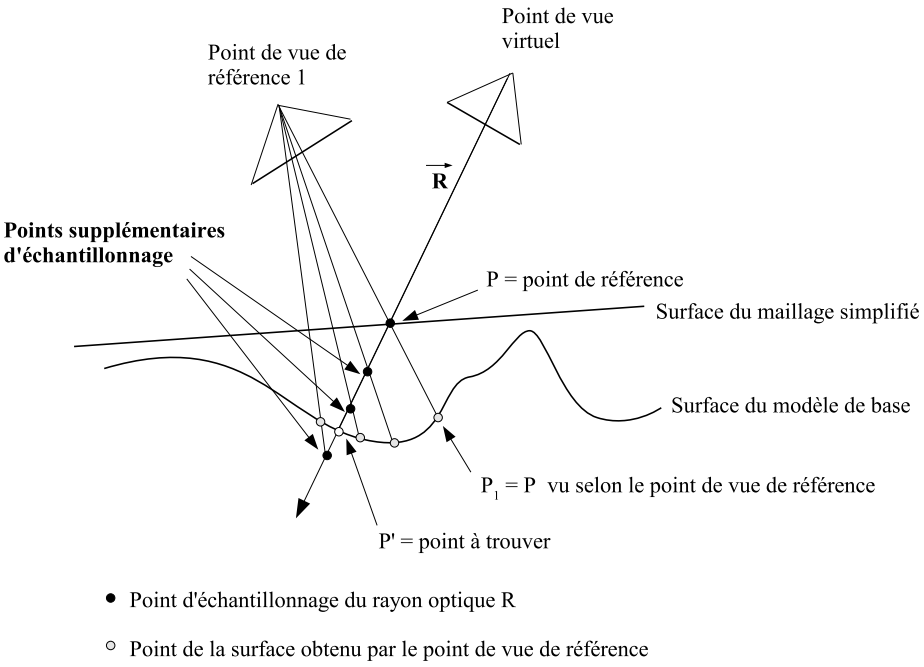


FIG. 4.14 – Echantillonnage d'un rayon optique $\overrightarrow{R}$. P_1 = le point que nous utilisons actuellement pour afficher P. P' = le point réel de la surface du maillage complexe de base. L'idée est de converger vers P' par échantillonnage de $\overrightarrow{R}$ à partir du point P.

complexes en utilisant des images de référence de l'ensemble ; les performances de la méthode seraient d'autant mises en avant.

4.4 Conclusion

Dans ce chapitre nous avons proposé une méthode d'interpolation de points de vue fonctionnant en temps réel et utilisant la carte graphique. Pour fonctionner, cette méthode nécessite la connaissance d'une version approchée du tampon de profondeur du modèle complexe.

Nous avons développé une méthode permettant d'obtenir ce tampon de profondeur approché pour une trajectoire d'observation précalculée, à un coût mémoire restreint. La précision du tampon de profondeur reconstruit est facilement contrôlable, ainsi que son occupation mémoire.

Nous avons ensuite étendu la méthode afin qu'elle fonctionne sans restrictions sur les points de vues. En utilisant un maillage simplifié du modèle original, nous obtenons également un tampon de profondeur approché du modèle. Notre méthode plaque le relief "manquant" stocké dans les images sur le modèle simplifié.

Ainsi nous cumulons les avantages d'une représentation à base d'image de la scène, stockant un relief détaillé, avec une représentation géométrique minimaliste de l'objet permettant de guider l'interpolation.

Outre le temps de rendu très satisfaisant, notre méthode cumule plusieurs avantages par rapport aux

méthodes existantes. Tout d'abord son implémentation est facile : la phase de précalculs ne consiste qu'à afficher un certain nombre de fois l'objet complexe sous différents angles de vue. De plus, aucune paramétrisation du maillage simplifié n'est nécessaire, le calcul des coordonnées de textures est effectué au vol, lors du rendu. Les calculs sont délocalisés dans le processeur de la carte graphique, ainsi la dépendance vis à vis du processeur central est très faible : il suffit de transmettre quelques textures à la carte graphique après un calcul très simple de proximité du point de vue virtuel avec les points de vue de référence. Finalement, cette méthode s'intègre directement dans un moteur de rendu conventionnel. En particulier, il est très facile d'implémenter l'auto-ombrage.

Néanmoins des améliorations sont possibles, notamment au niveau de la redondance des calculs et de la qualité de l'interpolation.

Conclusion et perspectives

D'un point du vue général, le rendu en temps réel est un sujet crucial en synthèse d'images. Dans le domaine de la simulation d'environnements virtuels, comme les jeux vidéos ou les simulateurs, l'interaction avec l'environnement est primordiale. Néanmoins, les contraintes de temps sont drastiques : la synthèse des images doit s'effectuer à une cadence au moins égale à 20 images par seconde, sous peine de perdre la sensation d'immersion dans l'environnement virtuel. Jusqu'à présent, il y a toujours un grand décalage entre la qualité visuelle des images de synthèse conventionnelles, comme par exemple celles utilisées dans le domaine de la post-production cinématographique, avec les images de synthèse calculées en temps réel. Avec l'apparition des cartes graphiques spécialement dédiées aux calculs de rendu, cette distinction tend à s'estomper.

De très nombreux travaux ont porté sur le rendu en temps réel de modèles complexes. Les techniques de simplification de maillage, de rendu à base d'images ou de points sont directement liées à cette problématique. Chacune de ces classes de méthode a ses propres avantages et inconvénients, que nous avons détaillés dans notre état de l'art.

Nous nous sommes efforcés, tout au long de cette thèse, de tirer parti des avantages de ces différentes approches. Nous avons proposé des méthodes transversales, hybrides, avec à chaque fois l'optique "temps-réel" en ligne de mire.

Nos premiers travaux ont porté sur l'extension au temps réel des méthodes d'interpolation d'images. L'interpolation d'image est séduisante car elle permet de générer les images intermédiaires à des points de vue de référence pré-acquis, par un calcul dépendant uniquement du nombre de pixels affichés, c'est à dire totalement indépendant de la complexité géométrique de la scène. Nous avons dans ce cadre développé trois méthodes distinctes, fonctionnant en temps réel.

La première s'apparente aux *textures dépendantes du point de vue* (VDTM, "View-Dependent Texture Mapping") de Debevec [DYB98, DTM96], car nous effectuons une moyenne pondérée de couleurs issues de points de vue de référence. Néanmoins, notre méthode est appliquée à un modèle virtuel, et non à des images réelles : le maillage simplifié de l'objet est peut alors être automatiquement créé, à l'aide des techniques de simplification de maillage conventionnelles, ce qui n'est pas le cas des VDTM qui nécessitent un lourd travail de reconstruction manuel. De plus, notre modèle est automatiquement paramétré, lors du rendu, ce qui n'est pas le cas du VDTM. Finalement, nous prenons en compte un éclairement variable de la scène, à l'aide de textures de normales associées au point de vue. Outre la possibilité de faire varier l'éclairement de la surface du modèle de façon réaliste, les textures de normales, également interpolées lors du mélange de points de vue, améliorent notablement la qualité du rendu.

La seconde méthode d'interpolation d'images que nous avons étudiée est le transfert de points, au sens de l'*interpolation de points de vue* de Seitz et Dyer [SD96]. Nous avons implémenté la méthode

à l'aide du GPU, après différents travaux préliminaires. Notre méthode, quoique imparfaite dans le cas général, donne de bons résultats quand les points de vue de référence ne sont pas trop éloignés.

La dernière méthode que nous avons développée est la création d'un maillage basé sur une image de profondeur. Nous n'avons fait qu'aborder le problème. La solution que nous avons proposée génère trop de triangles. La création de tels maillages basés sur une image de profondeurs est un sujet complexe qui pourrait faire l'objet d'une thèse à lui seul.

Une solution pour obtenir l'image d'un modèle complexe en un temps indépendant de sa complexité géométrique est de précalculer la visibilité des points de sa surface dans toutes les directions, à partir d'un plan positionné autour de lui. Nous avons implémenté cette approche directe en développant les *textures directionnelles*. Celles-ci s'apparentent aux *BTFs* de Dischler [Dis98], car une texture stocke en chaque élément de texture la géométrie d'un modèle complexe. Nous avons étendu le principe au temps réel en développant une structure à base de surfels quantifiés qui permet de stocker toutes les informations nécessaires au calcul de l'illumination d'un point, y compris sa visibilité. Nous avons développé une méthode de représentation de l'hémisphère de visibilité sous une forme compacte. La structure décrivant la scène complexe, le nuage de surfels, occupe très peu d'espace mémoire. Néanmoins, la structure de visibilité occupe beaucoup de mémoire, ce qui restreint l'intérêt des textures directionnelles en l'état.

Finalement, nous avons implémenté une technique d'interpolation de points de vue basée sur la rétroprojection de points qui donne d'excellents résultats, tant en termes d'espace mémoire et de vitesse de rendu qu'en termes de qualité visuelle. Nous avons appliqué cette méthode de deux façons différentes. La première permet de reconstruire l'image du modèle complexe pour un point de vue se situant sur une trajectoire fixe autour de la scène. La seconde se base sur un maillage simplifié du modèle. Cette dernière méthode est la plus rapide car elle ne dépend pratiquement pas du processeur central. Le temps de rendu dépend uniquement du nombre de pixels affichés, c'est à dire de la proportion d'espace écran qu'occupe le maillage simplifié. La méthode est en outre très facile d'implémentation et directement compatible avec tous les modèles d'éclairement courants en rendu en temps réel.

Concernant cette dernière méthode, de nombreux points restent améliorables. La silhouette, par exemple, est celle du maillage simplifié, ce qui peut poser problème dans certains cas. De plus, l'efficacité de l'interpolation pourrait être améliorée en évitant certains calculs redondants.

Concernant notre représentation en harmoniques de l'hémisphère d'ombrage des points, il serait facile d'implémenter cette fonctionnalité à l'aide de la carte graphique, en stockant les valeurs des principaux harmoniques dans une texture.

D'un point de vue général, le sens que nous donnons au mot *image* dans le domaine du rendu, a évolué. Nous parlons maintenant d'*image de profondeurs*, nous manipulons des *images* (ou *textures*) *de normales,* en plus de l'image traditionnelle de couleurs. Nos travaux ont fait intensivement appel à une représentation en *image de profondeurs* (*z-buffer*) de la scène. A bien des égards, ce type d'image s'apparente à un nuage de points : l'ensemble des points se projetant sur la caméra, exprimés dans son repère. Les dernières évolutions en matière de programmabilité des cartes graphiques permettent de tirer parti, comme nous l'avons fait, de cette connaissance explicite du relief de la scène. Cette approche hybride est très féconde et de nombreux travaux restent à faire : des algorithmes plus rapides et donnant de meilleurs résultats visuels peuvent être trouvés.

A

Shaders

Durant toute cette thèse, nous avons fait un usage intensif des dernières fonctionnalités de programmabilité des cartes graphiques : les *shaders*.

Les *shaders* sont des programmes informatiques fonctionnant sur les cartes graphiques. On distingue les *vertex shaders*, qui sont exécutés pour chaque sommet d'un maillage 3D à afficher, et les *pixel shaders*, ou *fragment shaders*, qui sont exécutés pour chaque pixel affiché.

Historiquement, les *shaders* ont commencé à être utilisés dans le contexte du rendu non-interactif dans des logiciels de rendu tels que *RenderMan* de Pixar, utilisé pour créer le film *Toy Story*. Depuis quelques années, les constructeurs NVIDIA et ATI ont développé des architectures matérielles dans leurs cartes graphiques permettant d'utiliser de tels programmes.

A.1 Le pipeline de rendu

L'algorithme 1 présente le pipeline de rendu standard, utilisé par toutes les applications de rendu en temps réel. L'algorithme est très sommaire, le but est uniquement de visualiser à quel niveau les *shaders* se positionnent dans le pipeline.

Algorithm 1 Pipeline de rendu pour l'affichage d'un maillage M de triangles.

Pour chaque triangle t de M **faire**

Pour chaque sommet s de t **faire**

transformer s dans le repère de la caméra
projeter s sur le plan de projection de la caméra
calculer l'éclairement de s

Vertex shader

Fin pour

Pour chaque pixel p de t **faire**

calculer sa couleur
calculer sa profondeur

Pixel shader

Fin pour

Fin pour

A.2 Vertex shader

Le *vertex shader* gère toutes les opérations à effectuer sur les sommets du maillage. Ces opérations sont principalement de deux types : les transformations géométriques, et les calculs d'éclairement.

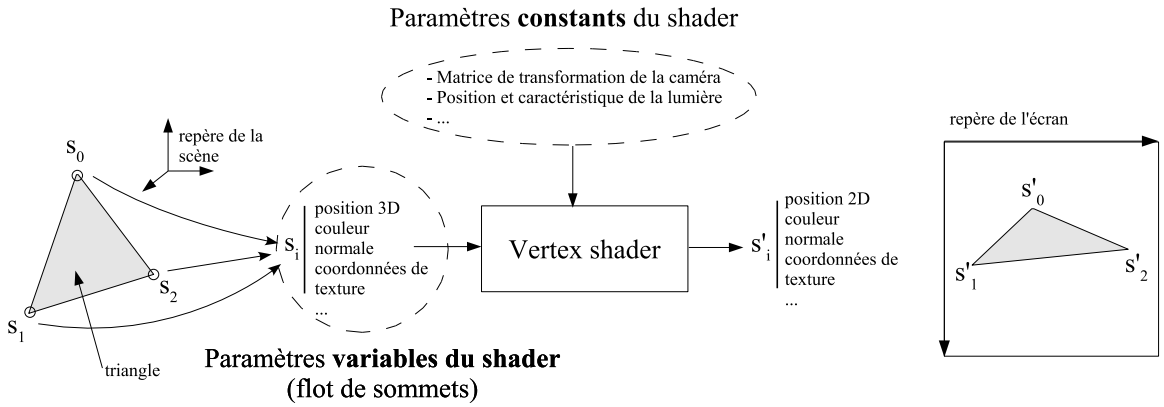


FIG. A.1 – *Vertex Shader*

Comme le montre la figure A.1, un *vertex shader* opère sur les sommets du maillage à afficher. Les paramètres d'un *shader* sont de deux types : les paramètres variables, c'est à dire les sommets, et les paramètres constants. Les paramètres constants sont typiquement les matrices de transformations représentant les opérations géométriques à appliquer aux sommets pour les projeter en 2D sur l'écran.

A.3 Pixel shader

Le *pixel shader* est exécuté pour chaque pixel affiché d'un triangle préalablement transformé par le *vertex shader*. Ses paramètres variables sont les paramètres variables **interpolés** calculés par le *vertex shader* (voir figure A.2 page 129). L'interpolation est effectuée automatiquement par la carte graphique qui elle seule dispose des informations de connexité entre les sommets d'un triangle.

Les paramètres constant d'un *pixel shader* sont typiquement les textures et des matrices de transformation.

A la sortie d'un *pixel shader* sont calculés la couleur du point (RVBα) et sa profondeur ; cet ensemble de données est appelé un *fragment* d'image. La couleur est représentée par ses composantes RVB et le canal de transparence α. La couleur est finalement stockée dans le tampon de couleur de l'écran, et la composante α dans le tampon α. La profondeur z du point est stockée dans le tampon de profondeur (*Z-Buffer*).

Ces différents tampons, stockés dans le *frame buffer*, sont combinés entre eux pour donner l'image finale, une matrice de couleurs, affichée à l'écran.

L'*unité de rastérisation* de la carte graphique contrôle la façon dont sont combinés ces tampons.

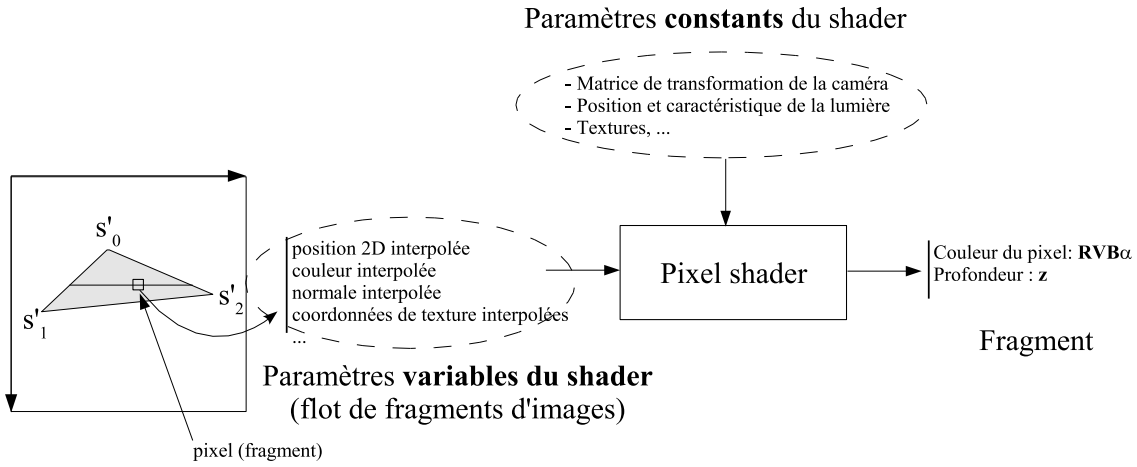


FIG. A.2 – *Pixel Shader*

A.4 Unité de rastérisation

L'unité de rastérisation (voir figure A.3 page 130) opére à la fin du pipeline de rendu et contrôle toutes les opérations de combinaison à effectuer sur l'image entière. Cette unité n'est pas programmable comme les *shaders* mais les opérations sont contrôlées par des paramètres que l'on peut choisir. Par exemple la façon dont se comporte le tampon de profondeur est définie par un test qui est effectué pour chaque pixel affiché. Dans le cas du tampon de profondeur, le test est généralement "autoriser l'affichage d'un pixel $p(x, y, z)$ si sa profondeur z est inférieure à la valeur stockée dans le tampon de profondeur aux coordonnées (x, y)". Le programmeur dispose ainsi d'un panel de fonctions pré-programmées.

Cette unité de rastérisation applique successivement une série d'opération sur chaque fragment calculé par le *pixel shader*[36]. Ces opérations sont (voir figure A.3) :

1. *le découpage* qui élimine le fragment si il se trouve hors de l'écran.
2. *le test* α qui détermine si le fragment doit être éliminé en fonction de sa composante α et d'une valeur de référence fixée, de la même façon que pour le test de profondeur (ci-dessous).
3. *le test du stencil* qui détermine si le fragment doit être éliminé en fonction d'une valeur stockée en (x, y) dans le *stencil buffer* et d'une fonction de test. Le *stencil buffer* est une matrice de la résolution de l'écran stockant en chaque élément une valeur sur 8 bits qui est mise à jour lors du test (la fonction de mise à jour est configurable). De nombreux effets tels que la projection d'ombre peuvent être effectués à l'aide du *stencil buffer.*
4. *le test de profondeur* (test de visibilité) qui détermine si le fragment doit être éliminé en fonction de sa profondeur z par rapport à la profondeur d'un précédent fragment affiché en (x, y).
5. *le mélange* α qui calcule la couleur finale d'un pixel en combinant la couleur stockée dans le tampon de couleurs en (x, y) avec la couleur du fragment.

[36]Le terme consacré de *pixel shader* est impropre, on devrait toujours employer *fragment shader*, car un pixel (*Picture Element*) est composé uniquement de ses 3 composantes RVB, tandis qu'un *fragment* comporte en plus une profondeur et une composante de transparence. La bibliothèque graphique OpenGL emploie le terme *Fragment Shader* tandis que la bibliothèque Microsoft DirectX emploie le terme *Pixel Shader*.

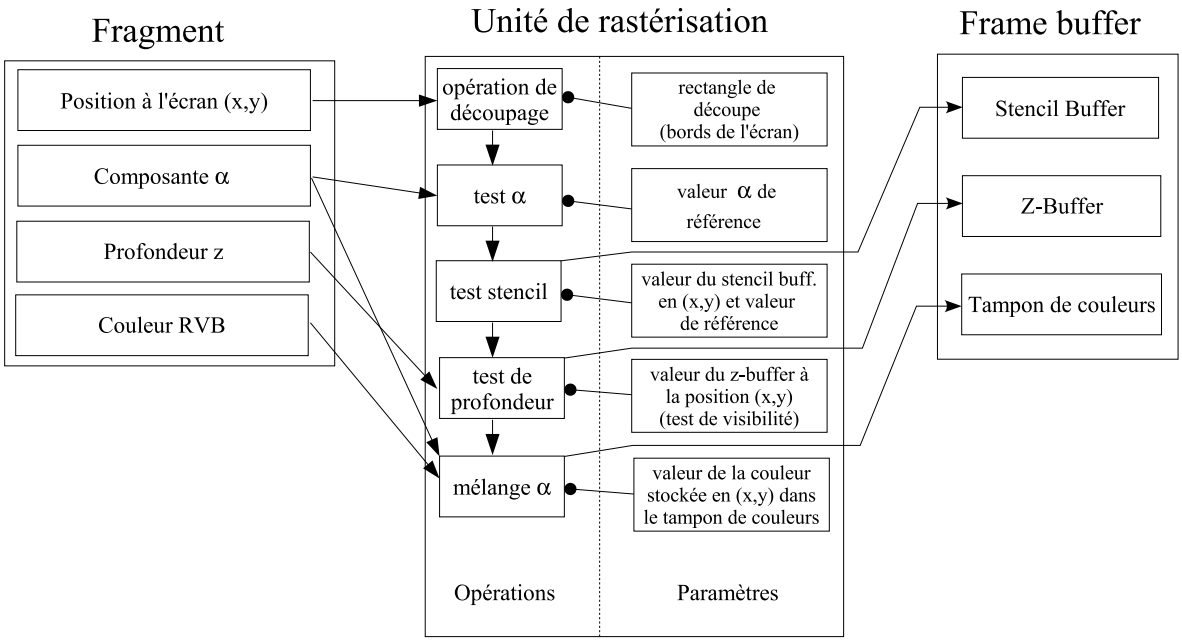


FIG. A.3 – Unité de rastérisation

A.5 Caractéristiques des shaders

Les *shaders* sont donc des programmes exécutés par le GPU[37]. Ils sont caractérisés par :

- leur taille, exprimée en nombre d'accès aux textures et en nombre d'opérations arithmétiques
- la précision interne des calculs. Différents formats de données coexistent : entier sur 16 ou 32 bits, flottant sur 16, 24, 32 ou 64 bits.
- la prise en compte des branchement conditionnels (sauts, boucles) : seules les toutes dernières cartes graphiques permettent un tel contrôle de flux dynamique.

Le tableau A.1 page 131 présente les caractéristiques des différentes versions des *shaders* depuis leur apparition, de la version 1 à 3.

Comme on peut le voir, les limitations sont drastiques. Bien que les opérations arithmétiques soient spécialement dédiées au calcul d'éclairement (par exemple les produits vectoriels et scalaires sont "cablés"), un *shader* est un programme très court.

A.6 Langages

Les deux bibliothèques graphiques temps réel exploitant les *shaders* sont OpenGL et Microsoft DirectX. On peut distinguer actuellement quatre niveaux dans la définition d'un *shader.*

[37]Le terme de GPU ("Graphics Processing Unit") a été historiquement introduit par NVIDIA, par analogie avec le CPU ("Central Processing Unit"). On trouve également la dénomination de VPU ("Visual Processing Unit"), introduite par la suite par la firme ATI.

	vs1	ps1	vs2	ps2	vs3	ps3
nb. op. arithmétiques	128	8	256	64	512	512
nb. accès textures	-	4	-	32	512	512
paramètres en entrée	16	8f+2e	16	8f+2e	16	10f
paramètres constants	96	8	256f +16b +16e	32f +16b +16e	256f +16b +16e	224f +16b +16e
registres temporaires	12	2	12	12	32	32
contrôle de flux	-	-	stat.	-	dyn.	dyn.

TAB. A.1 – Caractéristiques des différentes versions des *Shaders.* ps = pixel shader, vs = vertex shader, f = flottant, b = booléen, e = entier.

1. *Le langage machine*. Il dépend de la carte graphique (ATI ou NVIDIA) et est inaccessible par le programmeur.
2. *Le langage assembleur*. Il dépend de la bibliothèque graphique que l'on utilise (DirectX ou OpenGL). Ce niveau est dénommé *profil* dans la terminologie spécifique de la programmation sur GPU. Avec DirectX, les profils sont dénommés **PSx** et **VSx** pour les *pixel* et *vertex shaders* (x représentant la version de *shader*). Ils ont leur équivalent avec OpenGL, sous la dénomination **ARB_vertex_program_x** et **ARB_fragment_program_x**.
   En pratique, c'est un programme exprimé sous cette forme qu'on manipule. Le pilote (OpenGL ou DirectX), développé par le constructeur de la carte que l'on utilise, se charge de le compiler et de l'envoyer à la carte graphique. NVIDIA propose également des profils spécifiques à leur carte, sous OpenGL, très optimisés et permettant de tirer parti de leurs spécificités. Ils sont dénommés **NVxy**.
3. *Le langage de haut niveau*. Sous DirectX, le langage de programmation (proche du C) est **HLSL** (”High-Level Shading Language”), pour OpenGL c'est **GLSL** (”openGL Shading Language”). Il existe également **CG** (”C for Graphics”), développé par Nvdia, qui fonctionne avec les deux bibliothèques. Ces différents langages sont très proches les un des autres.
4. *L”'atelier de production” de shaders*. C'est un logiciel permettant de générer interactivement des *shaders* dans différents profils ou langages de haut niveau. L'utilisateur visualise en temps réel le résultat de ses manipulations, ce qui lui permet d'obtenir très finement les effets visuels souhaités. Le logiciel dédié de ATI est RenderMonkey et FX Composer pour NVIDIA.

A.7 Extensions

Les cartes graphiques devenant programmables, le GPU est apparu comme un processeur auxiliaire à part entière. Partant de la constatation que la puissance de calcul brute d'une carte graphique (programmable) est plusieurs fois supérieure au plus puissant des CPU, des chercheurs de l'Université de Stanford ont eu l'idée d'adapter leur langage de programmation dédié à du calcul en parallèle nommé *Brook*, aux cartes graphiques : c'est le projet *BrookGPU*.

BrookGPU est une bibliothèque de calcul en parallèle générant des *shaders* simples, des ”noyaux” qui sont appliqués à des données stockées dans des textures. Leur bibliothèque fournit une couche abstraction aux données que l'on manipule de façon transparente. Par exemple, un système de particule

se prète très bien à ce type de calcul : le nuage est stocké dans une texture ($XYZ \mapsto RGB$) , les calculs de mouvement sont effectués par le *pixel shader*, et les données obtenues sont retransformées en coordonnées de point ($RGB \mapsto XYZ$).

B

Compression de l'hémisphère d'ombrage d'un point

Dans cette annexe, nous présentons la méthode que nous avons employée pour représenter l'*hémisphère d'ombrage* d'un point sur quelques octets.

L'*hémisphère d'ombrage* est défini pour un point 3D p appartenant à la surface d'un maillage géométrique, et représente la visibilité de ce point selon toutes les directions d'un hémisphère axé selon sa normale (voir figure B.1). Une direction est représentée en coordonnées sphériques par ses deux angles (α, β) dans un repère orthonormé $(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$ centré sur p, avec $\overrightarrow{n} = \overrightarrow{y}$.

Un point est *visible* selon une direction $\overrightarrow{d}(\alpha, \beta)$ si un rayon issu de p et d'orientation $\overrightarrow{d}$ n'intersecte pas le maillage. Dans le cas contraire, le point est dans l'ombre. Sur la figure, p est visible par $\overrightarrow{d_0}$ et dans l'ombre par $\overrightarrow{d_1}$.

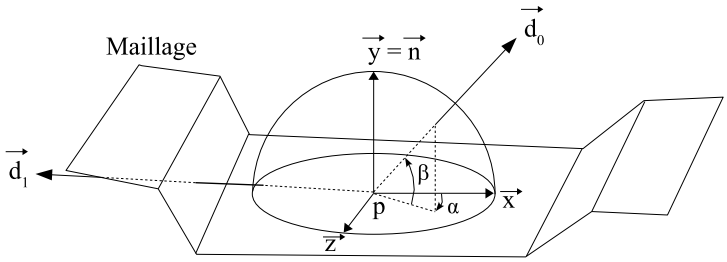


FIG. B.1 – Hémisphère d'ombrage.

B.1 Fonction d'ombrage

La fonction d'ombrage f est une fonction continue à valeurs réelles définie sur la sphère unité de $\mathbb{R}^3$ et représentant la visibilité du point :

$$f : [0, 2\pi] \times [-\pi, \pi] \mapsto [0, 1]$$

$$f(\alpha,\beta) = \begin{cases} 1 & \text{si le point est visible dans la direction } (\alpha,\beta) \\ 0 & \text{dans le cas contraire} \end{cases}$$

En pratique, nous discrétisons l'ensemble des directions en discrétisant les angles α et β sur N intervalles.

Pour chaque direction discrétisée de l'hémisphère supérieur nous effectuons un lancer de rayons afin de calculer la valeur de f. Ces valeurs sont recopiées dans l'hémisphère inférieur par symétrie par rapport au centre de la sphère.

Finalement, nous obtenons un tableau à deux dimensions (u,v) stockant les valeurs discrètes de f. C'est à dire :

$$f(\alpha,\beta) = f(u\Delta\alpha, v\Delta\beta) \equiv f(u,v) \tag{B.1}$$

Pour pouvoir appliquer la transformée de Fourier discrète en deux dimensions, nous utilisons un support carré pour f (i.e. $N \times N$).

B.2 Transformée de Fourier 2D discrète

L'expression de la transformée de Fourier discrète (TFD) de f est donc :

$$F(x,y) = \frac{1}{N^2} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} f(u,v).e^{\frac{-2\pi j}{N}(xu+yv)}$$

Après application de la TFD, nous obtenons le spectre de la fonction, c'est à dire un tableau de complexes de dimension N.

B.3 Harmoniques

Notre méthode consiste à ne stocker que les premiers harmoniques de la série de Fourier, les harmoniques principaux. Nous stockons :

$$F(0,0),\ F(1,0),\ F(0,1),\ F(1,1) \text{ et } F(-1,1) \tag{B.2}$$

$F(0,0)$, qui représente la moyenne de f, est représentée sur un octet. Les autres harmoniques sont stockés sur deux octets (i.e. amplitude et phase).

Ainsi, l'hémisphère d'ombrage est finalement représenté par les valeurs q_0, (q_1,φ_1), (q_2,φ_2),(q_3,φ_3) et (q_4,φ_4) soit 9 octets.

B.4 Reconstruction de l'hémisphère

L'expression de la transformée de Fourier 2D discrète inverse est :

$$f(u,v) = \frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} F(x,y).e^{\frac{2\pi j}{N}(xu+yv)}$$

Etant donné que nous travaillons sur un signal physique, nous devons extraire la partie réelle de la fonction complexe calculée avec la TFD inverse.

Soit :

$$H(x,y,u,v) = F(x,y).e^{\frac{2\pi j}{N}(xu+yv)}$$

et :

$$F(x,y) = q_i.e^{j\varphi_i}, \text{ avec } i \text{ l'indice de l'harmonique}$$

On a, avec $\Re$ représentant la partie réelle d'un complexe :

$$\begin{aligned}\Re[H] &= \Re[q_i(\cos\varphi_i + j.\sin\varphi_i).(\cos\frac{2\pi}{N}(xu+yv) + j.\sin\frac{2\pi}{N}(xu+yv))] \\ &= q_i(\cos\varphi_i\cos\frac{2\pi}{N}(xu+yv) - \sin\varphi_i\sin\frac{2\pi}{N}(xu+yv)) \\ &= q_i\cos(\varphi_i + \frac{2\pi}{N}(xu+yv))\end{aligned}$$

d'où :

$$\Re[f(u,v)] = \frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} q_i\cos(\varphi_i + \frac{2\pi}{N}(xu+yv))$$

Le signal discrétisé est reconstruit par B.1, soit :

$$\Re[f(\alpha,\beta)] = \frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} q_i\cos(\varphi_i + x\alpha + y\beta)$$

Soit, avec les harmoniques choisis (B.2) :

$$f(\alpha,\beta) = q_0 + q_1\cos(\alpha+\varphi_1) + q_2\cos(\beta+\varphi_2) + q_3\cos(\alpha+\beta+\varphi_3) + q_4\cos(-\alpha+\beta+\varphi_4)$$

Bibliographie

[AB91] E. H. Adelson and J. R. Bergen. *Computational Models of Visual Processing*, chapter 1 (The Plenoptic Function and the Elements of Early Vision). M. Landy and J. A. Movshon editors, MIT Press, Cambridge, MA, 1991.

[Aba03] Frédéric Abad. *Contribution à la synthèse de nouvelles vues à partir de photographies*. PhD thesis, Université de Nice-Sophia Antipolis - École Doctorale STIC, 2003.

[Ali96] D. G. Aliaga. Visualization of complex models using dynamic texture-based simplification. *Proceedings of IEEE Visualization conference*, pages 101 – 106, 1996.

[AMH02] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering, Second Edition*. 2002. ISBN 156881-182-9.

[Bli78] James F. Blinn. Simulation of wrinkled surfaces. *Computer Graphics (Proceedings of SIGGRAPH 78)*, 12(3) :286–292, August 1978. Held in Atlanta, Georgia.

[BN76] James F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10) :542–547, October 1976.

[BN92] Thaddeus Beier and Shawn Neely. Feature based image metamorphosis. In Edwin E. Catmull, editor, *Proceedings of the 19th Annual ACM Conference on Computer Graphics and Interactive Techniques*, pages 35–42, Chicago, IL, July 1992. ACM Press.

[Car84] L. Carpenter. The a-buffer, an antialiased hidden surface method. *Proc. of ACM Siggraph*, pages 103–108, 1984.

[Cat74] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, 1974.

[CBCG02] 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 *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 447–456. ACM Press, 2002.

[CBL99] Chun-Fa Chang, Gary Bishop, and Anselmo Lastra. Ldi tree : a hierarchical representation for image-based rendering. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 291–298. ACM Press/Addison-Wesley Publishing Co., 1999.

[Che95] Shenchang Eric Chen. QuickTime©VR : An image-based approach to virtual environment navigation. In Robert Cook, editor, *Proceedings of the Conference on Computer Graphics (SIGGRAPH-95)*, pages 29–38, Los Angeles, CA, USA, August 6–11 1995. ACM Press. ISBN 0-80791-701-4.

[Cla96] Dean Clark. Color Quantization Using Octrees. *Dr. Dobb's Journal*, 21(1) :54–57, 102–104, January 1996.

[COM98] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 115–122. ACM Press, 1998.

[Coo84] Robert L. Cook. Shade trees. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 223–231. ACM Press, 1984.

[CSKK99] Baoquan Chen, J. Edward Swan, II, Eddy Kuo, and Arie Kaufman. Lod-sprite technique for accelerated terrain rendering. In *Proceedings of the conference on Visualization '99*, pages 291–298. IEEE Computer Society Press, 1999.

[CVM+96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, P. Frederick Brooks, Jr., and William Wrigh. Simplification envelopes. In *Proceedings of the ACM Conference on Computer Graphics*, pages 119–128, New Orleans, LA, USA, August 1996. ACM.

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In Lynn Valastyan and Laura Walsh, editors, *Proceedings of the Annual Conference on Computer Graphics*, pages 279–288, Anaheim, CA, USA, August 1993. ACM Press. ISBN 0-89791-601-8.

[DCV97] Lucia Darsa, Bruno Costa Silva, and Amitabh Varshney. Navigating static environments using image-space simplification and morphing. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 25–ff. ACM Press, 1997.

[DDSD03] Xavier Decoret, Frédo Durand, François X. Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In John C. Hart, editor, *Proceedings of the 2003 ACM SIGGRAPH Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH-03)*, volume 22, 3 of *acm Transactions on Graphics*, pages 689–696, San Diego, CA, USA, july 2003. ACM Press.

[Dis98] J.-M. Dischler. Efficiently rendering macro-geometric surface structures using bi-directional texture functions. *Rendering Techniques98 (proceedings of EGRW)*, pages 169 – 180, 1998.

[DSSD99] Xavier Décoret, Gernot Schaufler, François Sillion, and Julie Dorsey. Multi-layered impostors for accelerated rendering. In *Computer Graphics Forum (Proc. of Eurographics '99)*, Sep 1999.

[DTM96] Patti E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs : A hybrid geometry- and image-based approach. In *Proceedings of the ACM Conference on Computer Graphics*, pages 11–20, New Orleans, LA, USA, august 1996. ACM.

[DWS+97] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. Roaming terrain : real-time optimally adapting meshes. In *Proceedings of the 8th conference on Visualization '97*, pages 81–88. IEEE Computer Society Press, 1997.

[DYB98] Paul Debevec, Yizhou Yu, and George Boshokov. Efficient view-dependent image-based rendering with projective texture-mapping. In *9th Eurographics Rendering Workshop*, Vienna, Austria, June 1998.

[EM99] Carl Erikson and Dinesh Manocha. Gaps : general and automatic polygonal simplification. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 79–88. ACM Press, 1999.

[GBP04] Gael Guennebaud, Loïc Barthe, and Mathias Paulin. Deferred Splatting. In *Computer Graphics Forum*, volume 23, pages 1–11 (To appear). European Association for Computer Graphics and Blackwell Publishing, PO Box 805, 108 Cowley Road, Oxford, United Kingdom, septembre 2004. (EG2004 Proceedings).

[Ger39] A. Gershun. Svetovoe Pole (The Light Field, in English). *Journal of Mathematics and Physics*, XVIII :51–151, 1939.

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM Press, 1996.

[GH95] Michael Garland and Paul S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Computer Science Department, Carnegie Mellon University, 1995.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

[GP03] Gael Guennebaud and Mathias Paulin. Efficient screen space approach for Hardware Accelerated Surfel Rendering. In *Vision, Modeling and Visualization, Munich*, pages 1–10. IEEE Signal Processing Society, 19-21 novembre 2003.

[Gro98] J.P. Grossman. Point Sample Rendering. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, August 1998.

[GW92] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley, Reading, MA, USA, 3rd edition, 1992. ISBN 0-201-50803-6.

[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 *Proceedings of the ACM SIGGRAPH Conference (SIGGRAPH-97)*, pages 225–232, Losa Angeles, CA, USA, august 1997. ACM Press.

[Hal98] Michael Halle. Multiple viewpoint rendering. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 243–254. ACM Press, 1998.

[Hec82] P. S. Heckbert. Color Image Quantization for Frame Buffer Display. *ACM Computer Graphics (ACM SIGGRAPH '82 Proceedings)*, 16(3) :297–307, 1982.

[Hec86] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics & Applications*, (11) :56–67, November 1986.

[Hec89] P. Heckbert. Fundamentals of texture mapping and image warping. Master thesis (technical report no. ucb/csd 89/516), University of California, Berkeley, 1989.

[HEGD04] Johannes Hirche, Alexander Ehlert, Stefan Guthe, and Michael Doggett. Hardware accelerated per-pixel displacement mapping. In *Proceedings of the 2004 conference on Graphics interface*, pages 153–158. Canadian Human-Computer Communications Society, 2004.

[Her96] Julio A. Hernandez. Color image compression using vector quantization and neural networks. Master's thesis, University of Puerto Rico, Mayaguez Campus, 1996.

[HHK+95] Taosong He, Lichan Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel based object simplification. In *Proceedings of the 6th conference on Visualization '95*, page 296. IEEE Computer Society, 1995.

[Hof92] Miller-G. Chen E.S. Patterson E. Blackketter D. Rubin S. Applin S. et. al. Hoffert, E.M. The virtual museum. In *Proceedings of the Imagina '92 Conference (Monte Carlo, France)*, February 1992.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM Press, 1996.

[Hop97] Hugues Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 189–198. ACM Press/Addison-Wesley Publishing Co., 1997.

[Hop98] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization '98*, pages 35–42, 1998.

[HS99] Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 171–178. ACM Press/Addison-Wesley Publishing Co., 1999.

[KHD00] Jan Kautz, Wolfgang Heidrich, and Katja Daubert. Bump map shadows for OpenGL rendering. Technical report MPI-I-2000-4-001, Max-Plank-Institut für Informatik, Saarbrüken, Germany, 2000.

[KK89] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. pages 271–280, Boston, MA, USA, July 1989. ACM Press. ISBN 0-201-50434-0.

[KS01] Jan Kautz and Hans-Peter Seidel. Hardware accelerated displacement mapping for image based rendering. In *No description on Graphics interface 2001*, pages 61–70. Canadian Information Processing Society, 2001.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes : A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.

[LE97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 199–208. ACM Press/Addison-Wesley Publishing Co., 1997.

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the ACM Conference on Computer Graphics*, pages 31–42, New Orleans, LA, USA, August 4–9 1996. ACM. ISBN 0-201-94800-1.

[Lip80] Andrew Lippman. Movie-maps : An application of the optical videodisc to computer graphics. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 32–42. ACM Press, 1980.

[LKR+96] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118. ACM Press, 1996.

[LPC+00] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project : 3d scanning of large statues. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144. ACM Press/Addison-Wesley Publishing Co., 2000.

[LT97] Kok-Lim Low and Tiow-Seng Tan. Model simplification using vertex-clustering. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 75–ff. ACM Press, 1997.

[LT00] Peter Lindstrom and Greg Turk. Image-driven simplification. *ACM Trans. Graph.*, 19(3) :204–241, 2000.

[Lue01] David P. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, 21(3) :24–35, 2001.

[LW85] M. Levoy and T. Whitted. The Use of Points as a Display Primitive. Technical Report TR85-022, Dept. of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1985.

[Max88] Nelson L. Max. Horizon mapping : shadows for bump-mapped surfaces. *The Visual Computer*, 4(2) :109–117, July 1988.

[Max96] N. Max. Hierarchical Rendering of Trees from Precomputed Multi-Layer Z-Buffers. In X. Pueyo and P. Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 165–174, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

[MB95] Leonard McMillan and Gary Bishop. Plenoptic modeling : An image-based rendering system. *Proceedings of SIGGRAPH 95*, pages 39–46, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California.

[McM95a] Leonard McMillan. Computing Visibility Without Depth. Technical Report TR95-047, Department of Computer Science, University of North Carolina - Chapel Hill, October 1995.

[McM95b] Leonard McMillan. Shape as a perturbation to projective mapping. Technical Report TR95-046, University of North Carolina, july 1995.

[McM97] Leonard McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, April 1997.

[MF02] Takashi Kanai Masahiro Fujita. Hardware-assisted relief texture mapping. *Eurographics 2002 short paper presentation*, 21(3) :257–262, September 2002.

[MH84] Gene S. Miller and C. Robert Hoffman. Illumination and reflection maps : Simulated objects in simulated and real environments. In *SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*. july 1984.

[MMB97] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–ff. ACM Press, 1997.

[MO95] N. Max and K. Ohsaki. Rendering Trees from Precomputed Z-Buffer Views. In *Proceedings of the 6th Eurographics Workshop on Rendering*, June 1995.

[MS95] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *ACM Siggraph Symposium on Interactive 3D Graphics*, 1995.

[Ney95] Fabrice Neyret. A general and multiscale model for volumetric textures. In Wayne A. Davis and Przemyslaw Prusinkiewicz, editors, *Graphics Interface '95*, pages 83–91. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1995. ISBN 0-9695338-4-5.

[OBM00] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief texture mapping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 359–368. ACM Press/Addison-Wesley Publishing Co., 2000.

[Oos98] C. Oosterbaan. Motion and deformation of surfels objects. Master's thesis, Delft University of Technology and MERL, 1998.

[PCD+97] Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. View-based rendering : Visualizing real objects from scanned range and color data. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 23–34, New York City, NY, June 1997. Eurographics, Springer Wein. ISBN 3-211-83001-4.

[PDG04] Damien Porquet, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Real-Time Image-Based Geometry Mapping. Technical Report MSI-0401, Université de Limoges, Laboratoire MSI, 2004. Soumis à une conférence internationale.

[PGD04] Damien Porquet, Djamchid Ghazanfarpour, and Jean-Michel Dischler. Real-time per-pixel viewpoint interpolation. *Proceedings of Theory and Practice of Computer Graphics'2004, IEEE, Bournemouth, England*, pages 219 – 225, June 2004.

[PHL+98] Frédéric Pighin, Jamie Hecker, Dani Lischinski, Richard Szeliski, and David H. Salesin. Synthesizing realistic facial expressions from photographs. In *Proceedings of the ACM Conference on Computer Graphics (SIGGRAPH-98)*, pages 75–84, Orlando, FL, USA, July 1998. ACM Press. ISBN 0-89791-999-8.

[PM88] J. G. Proakis and D. G. Manolakis. *Introduction to Digital Signal Processing*. Macmillan, New York, 1988.

[PSM04] R. Pajarola, M. Sainz, and Y. Meng. Dmesh : Fast depth-image meshing and warping. *International Journal of Image and Graphics (IJIG)*, 4(4) :1 – 29, 2004.

[PZvG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels : surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342. ACM Press/Addison-Wesley Publishing Co., 2000.

[RB93] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. In B. Falcidieno and T. L. Kunii, editors, *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, June–July 1993.

[RB98] Paul Rademacher and Gary Bishop. Multiple-center-of-projection images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 199–206. ACM Press, 1998.

[RL00] Szymon Rusinkiewicz and Marc Levoy. Qsplat : a multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[RPZ02] L. Ren, H. Pfister, and M. Zwicker. Object space ewa surface splatting : A hardware accelerated approach to high quality point rendering. *EUROGRAPHICS 2002*, pages 461 – – 470, 2002.

[SC00] Peter-Pike J. Sloan and Michael F. Cohen. Hardware accelerated horizon mapping. In B. Péroche and H. Rushmeier, editors, *Proceedings of the Eurographics Workshop on*

Rendering Techniques 2000 (RENDERING TECHNIQUES-00), pages 281–286, Brno, Czech Republic, june 2000. Springer-Verlag.

[Sch94] John Schlag. Fast embossing effects on raster image data. In Paul Heckbert, editor, *Graphics Gems IV*, pages 433–437. Academic Press, Boston, 1994.

[Sch95] Gernot Schaufler. Dynamically generated impostors. *GI Workshop on Modeling, Virtual Worlds, Distributed Graphics, Bonn, Germany*, pages 129–139, 1995.

[Sch97] Gernot Schaufler. Nailboards : A Rendering Primitive for Image Caching in Dynamic Scenes. In *Proc. 8th Eurographics Workshop on Rendering*, pages 151–162. Springer Verlag, 1997.

[Sch98] Gernot Schaufler. Per-object image warping with layered impostors. *Computer Graphics Forum (Proceedings of Eurographics '98)*, (3) :145–156, 1998.

[SD96] Steven M. Seitz and Charles R. Dyer. View morphing. In *Proceedings of the ACM Conference on Computer Graphics*, pages 21–30, New Orleans, LA, USA, August 4–9 1996. ACM. ISBN 0-201-94800-1.

[SDB97] Francois Sillion, George Drettakis, and Benoit Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum, (Proc. Eurographics '97)*, 16(3) :207–218, 1997.

[SGwHS98] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242. ACM Press, 1998.

[SH99] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 299–306. ACM Press/Addison-Wesley Publishing Co., 1999.

[SLS+96] Jonathan Shade, Dani Lischinski, David H. Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 75–82. ACM Press, 1996.

[Smi90] D. B. Smithe. A two-pass mesh warping algorithm for object transformation and image interpolation. Technical memo #1030, Industrial Light and Magic, 1990.

[SP99] G. Schaufler and M. Priglinger. Efficient Displacement Mapping by Image Warping. In *Proc. 10th Eurographics Workshop on Rendering*. Springer Verlag, 1999.

[SZL92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *Proceedings of the 19th Annual ACM Conference on Computer Graphics and Interactive Techniques*, pages 65–70, New York, NY, USA, July 1992. ACM Press.

[TKT01] Masahiko Inami Naoki Kawakami Yasuyuki Yanagida Taro Maeda Tomomichi Kaneko, Toshiyuki Takahei and Susumu Tachi. Detailed shape representation with parallax mapping. In *Proceedings of the ICAT 2001 (The 11th International Conference on Artificial Reality and Telexistence)*, pages 205–208, December 2001.

[vB98] J. van Baar. Generation and rendering of surface elements. Master's thesis, Delft University of Technology and MERL, 1998.

[VFSH03] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. Automatic view selection using viewpoint entropy and its application to image-based modelling. *Computer Graphics Forum*, 22(4), 2003.

[WAA+00] 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 *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 287–296. ACM Press/Addison-Wesley Publishing Co., 2000.

[Wel04] Terry Welsh. Parallax mapping with offset limiting : A per-pixel approximation of uneven surfaces. Technical report, Infiscape Corporation, January 2004.

[Wes90] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4) :367–376, August 1990.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274. ACM Press, 1978.

[WWT+03] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. View-dependent displacement mapping. *ACM Trans. Graph.*, 22(3) :334–339, 2003.

[YSK+02] Shuntaro Yamazaki, Ryusuke Sagawa, Hiroshi Kawasaki, Katsushi Ikeuchi, and Masao Sakauchi. Microfacet billboarding. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 169–180. Eurographics Association, 2002.

[ZGPvB01] Matthias Zwicker, Markus Gross, Hanspeter Pfister, and Jeroen van Baar. Surface splatting. pages 371–378, Los Angeles, CA, 12–17 2001. ACM Press.