Université Sidi Mohammed Ben Abdellah
Faculté des Sciences Dhar El Mahraz – Fès

Département d'Informatique - UFR Informatique Avancée
Laboratoire d'Informatique, Imagerie et Analyse Numérique (LIIAN)

THESE DE DOCTORAT NATIONAL

Discipline : Informatique
Spécialité : Informatique
N° d'ordre : _____

Placage de Déplacement Tridimensionnel par Pixel pour le Rendu Temps Réel

Présentée et soutenue publiquement par

Akram HALLI

le 6 Juin 2009

Directeur de thèse : Pr. Khalid SATORI
Co-encadrant : Pr. Hamid TAIRI

Composition du jury

Président	Pr. Driss ABOUTAJDINE	Faculté des Sciences de Rabat Académie Hassan II des Sciences et Techniques
Directeur	Pr. Khalid SATORI	Faculté des Sciences Dhar El Mahraz – Fès
Rapporteurs	Pr. Abdellah AARAB	Faculté des Sciences Dhar El Mahraz – Fès
	Pr. Ahmed ROUKHE	Faculté des Sciences de Meknès
Examinateurs	Pr. Nasser ASSEM	Al Akhawayn University – Ifrane
	Pr. Noureddine CHENFOUR	Faculté des Sciences Dhar El Mahraz – Fès
	Pr. Saïd OUATIK EL ALAOUI	Faculté des Sciences Dhar El Mahraz – Fès

REMERCIEMENTS

Je voudrais tout d'abord remercier vivement le Pr. Khalid SATORI, mon directeur de thèse, pour m'avoir initié au monde de la recherche, et pour m'avoir proposé de travailler sur un sujet très porteur et passionnant. Je tiens à le remercier également pour ses encouragements et sa disponibilité, ainsi que pour les nombreux échanges et les conseils pertinents qui ont permis à ce travail d'aboutir.

Je tiens également à remercier chaleureusement le Pr. Hamid TAIRI, mon codirecteur de thèse, d'une part, pour le soutien permanent et la disponibilité qu'il m'a accordés, et d'autre part pour les nombreux échanges et discussions qui ont énormément apporté à ce travail.

Mes plus sincères remerciements vont à tous les professeurs qui m'ont honoré de leur participation au jury de ma thèse, et qui ont accepté de évaluer mon travail. Tout d'abord, le Pr. Driss ABOUTAJDINE qui m'a fait l'honneur de présider mon jury de thèse. Je remercie chaleureusement mes rapporteurs de thèse, le Pr. Ahmed ROUKH et le Pr. Abdellah AARAB pour le temps consacré à la lecture de ce manuscrit, et pour leurs remarques et suggestions pertinentes. J'adresse un très grand merci aux membres examinateurs, le Pr. Nasser ASSEM, le Pr. Noureddine CHENFOUR, et le Pr. Saïd OUATIK EL ALAOUI, pour le temps, l'effort et l'intérêt qu'ils ont porté à mon travail.

J'aimerais aussi remercier chaleureusement tous les professeurs et les doctorants de la Faculté des Sciences Dhar El Mahraz, pour les échanges constructifs que nous avons pu avoir, et pour les moments agréables que nous avons passés ensemble. Je remercie spécialement mon ami Abderrahim Saaidi, pour notre étroite collaboration tout au long de nos études doctorales.

Enfin, je dédie cette thèse à toute ma famille, à ma chère mère en premier, à mes deux sœurs Sabah et Sabrina, à mon petit neveu Sami, à Driss, à Mehdi, à ma fiancée Ouiame et sa famille, ainsi qu'à tous mes amis, spécialement Houcine et Laraqui.

RESUME

De nos jours, les images de synthèse occupent une place de plus en plus importante dans des domaines aussi variés que l'ingénierie ou le divertissement. Le matériel informatique a énormément évolué ces dernières années afin de pouvoir suivre cette tendance. Cependant, le matériel ne peut satisfaire seul aux exigences, en terme de qualité et d'interactivité, que requièrent les nouvelles applications. D'où l'intérêt d'algorithmes permettant d'optimiser le temps d'exécution et l'occupation mémoire de telles applications, et notamment celles qui nécessitent une interactivité avec l'utilisateur.

Le placage de déplacement tridimensionnel par pixel est une nouvelle technique qui permet d'enrichir visuellement les scènes 3D, en simulant les mesostructures (microreliefs) présentes sur de nombreux types de surface. Cette amélioration s'effectue sans modification de la définition géométrique des objets 3D, et s'appuie uniquement sur le placage de textures dédiées à cet effet. Cette technique contribue ainsi à éviter la saturation du pipeline graphique que peut provoquer le traitement d'un très grand nombre de primitives graphiques. Dans le même esprit, la technique de *modélisation et rendu à base d'images*, qui est une extension du placage de déplacement 3D par pixel, permet de créer et d'afficher des objets 3D dans leur intégralité, sans passer par le traditionnel maillage polygonal.

Au cours de ce travail, nous nous sommes intéressés à l'une des meilleures techniques pour le placage de déplacement 3D par pixel. Il s'agit de la technique de *traçage de cônes*, dont nous avons pu faire passer les algorithmes de prétraitement des textures d'une complexité quadratique à une complexité linéaire. Nous avons également proposé un nouvel algorithme permettant la prise en charge des textures rectangulaires, car elles étaient très mal gérées par les méthodes existantes. Nous avons aussi développé une nouvelle approche qui rend possible la synchronisation temps réel entre la modification de l'échelle du microrelief et son ombrage.

Cette thèse introduit notamment le *placage d'extrusion et de révolution*. Il s'agit d'une nouvelle technique de modélisation et rendu à base d'images. L'approche que nous avons adoptée permet de créer des géométries sans maillage, par extrusion ou par révolution d'une forme de base, stockée sous forme d'une image 2D binaire. Ce type de géométrie est très utilisé en modélisation 3D, et particulièrement dans la création d'environnements architecturaux. La technique proposée est très performante, que ce soit en terme d'interactivité, ou en terme de qualité visuelle.

Mots-clés : *rendu temps réel, mesostructures, placage de déplacement par pixel, traçage de cônes, modélisation et rendu à base d'images, surfaces extrudées, surfaces de révolution.*

ABSTRACT

PER-PIXEL THREE-DIMENSIONAL DISPLACEMENT MAPPING FOR REAL-TIME RENDERING

Today's computer graphics plays an increasingly important role in fields as diverse as engineering or entertainment. The electronic components have evolved tremendously in recent years in order to follow that trend. However, the hardware alone cannot meet the requirements in terms of quality and interactivity, expected by the new graphics applications. Hence the importance of algorithms that optimize the running time and the memory occupation of such applications, particularly those requiring interactivity with the user.

Per-pixel three-dimensional displacement mapping is a new technique that enhances the rendering of 3D scenes, by simulating the micro-relief present on various types of surface. This improvement is made without changing the geometric definition of the 3D models, and relies only on the mapping of textures dedicated for this purpose. This technique helps to avoid the bottleneck problem caused by the huge number of graphics primitives to be submitted to the graphics card. In the same way, *Image-Based Modeling and Rendering technique,* which is an extension of per-pixel displacement mapping, can create and display full 3D models, without using the traditional polygonal mesh.

During this work, we are interested in one of the best techniques for Per-pixel 3D displacement mapping, namely *the cone tracing technique*, we were able to improve the pre-processing algorithms (from quadratic time to linear time). We also proposed a new algorithm for taking into account rectangular textures, because they were poorly managed by existing methods. We have also developed a new approach that enables real-time synchronization between the modulation of the scale of the micro-relief and the resulting shading.

This thesis introduces in particular *the extrusion and revolution mapping*, which is a novel image-based modeling and rendering technique. The approach we have adopted can create geometries without meshes, by the extrusion or the revolution of a main shape, stored as a binary 2D image. This kind of geometry is widely used in 3D modeling, especially for creating architectural environments. The proposed technique is very powerful, both in terms of interactivity and visual quality.

Keywords : *real-time rendering, mesostructures, per-pixel displacement mapping, cone tracing, image-based modeling and rendering, extruded surface, surfaces of revolution.*

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

Introduction Générale

La technique de placage de déplacement tridimensionnel par pixel est un algorithme utilisé essentiellement dans le cadre du rendu temps réel d'environnements tridimensionnels. Cette technique a pour objectif de restituer les surfaces présentant des microreliefs, sans procéder à la densification de leur maillage de base, en se basant uniquement sur la technique de placage de texture et sur une carte de profondeurs. Dans cette introduction générale, nous allons commencer par situer le cadre scientifique dans lequel évolue cette thèse. Nous évoquerons les motivations qui nous ont poussées à adopter le thème du placage de déplacement tridimensionnel par pixel, ainsi que l'objectif que nous nous sommes fixés. Enfin, nous présenterons brièvement nos différentes contributions concernant le sujet traité.

Cadre de la thèse

Depuis que les ordinateurs ont acquis la puissance nécessaire au traitement de l'imagerie numérique, les images de synthèse ont investie quasiment tous les secteurs de la technologie et de la communication, où elles jouent souvent un rôle central. Cela est dû notamment au rapport privilégié qu'entretient le cerveau humain avec les images, à la différence du texte, du langage ou des modélisations abstraites.

L'infographie 3D constitue désormais une composante essentielle dans de nombreux secteurs d'activités tels que : la médecine, l'ingénierie ou le divertissement. Cependant, l'usage de l'imagerie de synthèse varie significativement d'un secteur à l'autre, suivant les moyens matériels et les objectifs de chaque spécialité. D'une manière générale, l'usage de l'infographie est divisé en deux catégories distinctes : Le rendu pré-calculé et le rendu temps réel. Le premier produit des images réalistes de très haute qualité, tandis que le deuxième est axé principalement sur la vitesse du rendu, ce qui lui permet de produire des animations interactives fluides.

L'apparence des modèles 3D est constituée généralement d'une structure à trois niveaux : Le niveau macrostructure définit la structure globale de l'objet qui est créé à l'aide d'un maillage. Le niveau microstructure désigne la micro-géométrie qui est simulée par le calcul d'ombrage et par des textures. Enfin, le niveau mesostructure est une structure intermédiaire représentant les microreliefs qui couvrent la surface d'un objet. Ces derniers sont très petits pour être créés par un maillage, et ils sont nettement visibles pour être simulés à l'aide d'une simple texture ombrée.

Le cas des mesostructures a constitué un problème majeur, notamment pour le rendu temps réel. Le *placage de bosselures* a été l'une des premières solutions. Cette technique, qui est basée uniquement sur l'ombrage, effectue une perturbation des normales à la surface 3D, afin d'induire une impression de relief. Toutefois, même si elle est toujours la technique la plus utilisée, elle demeure néanmoins une solution assez basique. Une meilleure technique, appelée *placage de déplacement,* a permis de masquer presque tous les défauts du placage de bosselures, car elle procède à une densification du maillage. Cependant, vu la taille du maillage à traiter, cette technique n'est utilisée que dans le cadre du rendu pré-calculé.

La technique du *placage de déplacement tridimensionnel par pixel* représente une solution plus intéressante pour le rendu temps réel. Elle permet notamment de contourner la densification du maillage. Cette solution repose à la fois sur l'ombrage du placage de bosselures, et sur le

déplacement de la surface de base en fonction d'une une carte de profondeurs. Cependant, le déplacement ici se fait au niveau des pixels, et non plus au niveau des sommets.

En s'inspirant du placage de déplacement tridimensionnel par pixel, certaines méthodes ont été introduites pour pouvoir créer des objets 3D entiers sans maillage (sauf sous une forme basique comme des plans ou des boites). La forme géométrique de l'objet 3D est déduite à partir de textures 2D ou 3D. Ces techniques, dites de *modélisation et rendu à base d'images*, sont très prometteuses, notamment dans le cas où les scènes sont très détaillées.

C'est dans le cadre du placage de déplacement tridimensionnel par pixel, ainsi que la modélisation et rendu à base d'images, que se situe cette thèse. Nous avons pu améliorer significativement l'une des meilleures techniques concernant le placage de déplacement 3D par pixel, et nous avons notamment introduit une nouvelle technique de modélisation et rendu à base d'images.

Motivations et But

Cette thèse, constitue une suite naturelle d'un sujet de recherche entamé pendant le cycle du DESA, et portant sur la modélisation 3D et les visites virtuelles d'édifices. Au départ, l'objectif de la thèse était de procéder à la reconstruction 3D de monuments historiques, afin de constituer une base de données du patrimoine bâti. Cependant, un tel projet nécessitait un matériel professionnel onéreux, et par conséquent, un budget assez considérable. Nous nous sommes alors concentrés sur la partie théorique de ce projet, en nous penchant spécialement sur un problème que nous avons rencontré pendant la modélisation et les visites virtuelles de bâtiments. En effet, au cours de ce travail, nous avons particulièrement été confronté à la problématique de l'interactivité que nécessitaient les visites virtuelles. Les scènes que nous avons créées étaient complexes et très riches en détails (voir la figure 0.1). Cependant, comme ces derniers sont créés à partir de maillages, le nombre de polygones et de sommets était tellement important que la carte graphique n'arrivait plus à calculer le nombre d'images nécessaires permettant d'avoir l'impression d'une animation. Nous avons alors été obligés de simplifier énormément le maillage et de supprimer quelques éléments de la scène. Cela a résolu le problème de l'interactivité, mais au prix d'une scène assez pauvre et beaucoup moins réaliste. Nos nous sommes alors orienté vers l'étude d'algorithmes permettant d'avoir la richesse et le réalisme des scènes 3D, tout en conservant l'interactivité pendant la navigation.

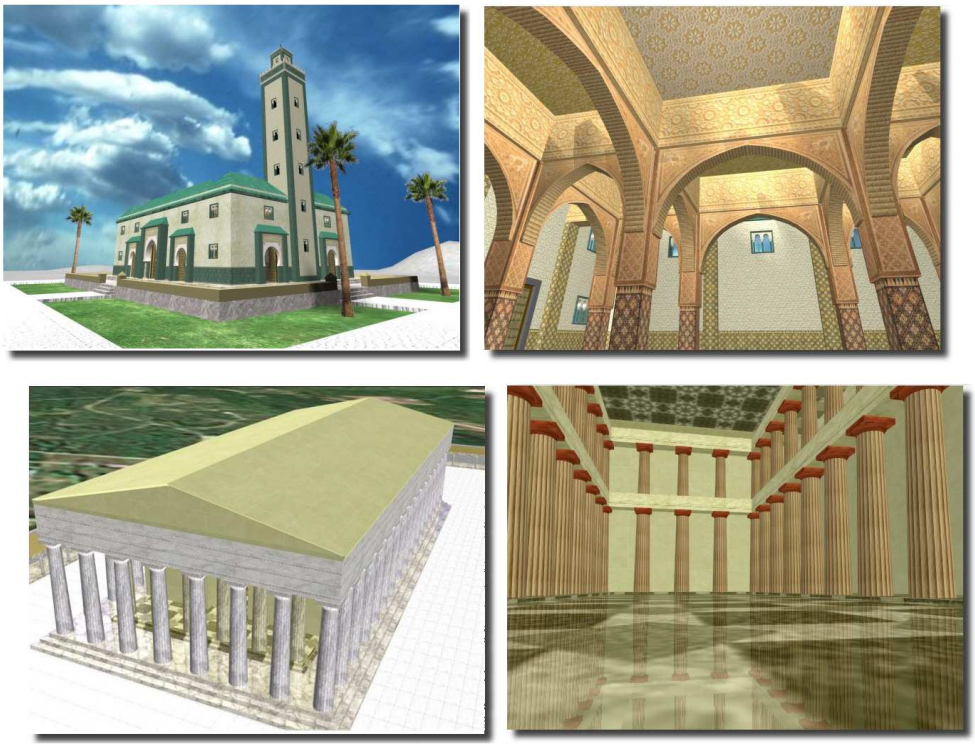


Fig. 0.1 - Modélisation d'une mosquée d'architecture marocaine (en haut), et du Parthénon d'Athènes (en bas)

Contributions

Nos contributions sont directement liées à l'objectif que nous nous sommes fixés, à savoir : Conserver la richesse visuelle des scènes 3D complexes, sans diminuer la vitesse du rendu permettant l'interactivité avec la scène.

Notre première contribution consiste à synchroniser en temps réel le changement d'échelle de la profondeur du microrelief avec l'ombrage de ce dernier. En effet, les techniques existantes ne permettent pas cette synchronisation, car elles supposent que l'échelle est fixée d'avance. Or, la variation d'échelle est parfois très utile, comme dans le cas où une même carte de déplacement doit être utilisée avec des échelles différentes, ou lorsque la profondeur doit être animée et constamment mise à jour. Nous avons proposé deux solutions à ce problème : La première consiste à recalculer la normale en fonction de celle déjà stockée dans la carte de déplacement. La deuxième méthode stocke les dérivées partielles de la carte de profondeurs au lieu de ses normales, puis calcule celles-ci pendant la phase de rendu en fonction de l'échelle de profondeur souhaitée.

Notre seconde contribution concerne le placage de déplacement par traçage de cônes. Cette technique, qui permet d'ajouter des microreliefs à une surface 3D, est l'une des meilleures

solutions concernant les mesostructures. Cependant, elle souffre d'un lourd prétraitement qui la pénalise par rapport aux autres approches, en plus, elle gère mal les textures non-carrées. Nous avons ainsi proposé des algorithmes de prétraitement à complexité linéaire. Nous avons également introduit une nouvelle méthode permettant une meilleure gestion des textures rectangulaires. Enfin, nous avons adopté une nouvelle approche permettant de faire passer le demi-angle des cônes traités de $\pi/4$ à $\pi/2$.

Notre troisième apport consiste en l'introduction d'une nouvelle technique, que nous avons appelée : *Placage d'extrusion et de révolution*. Cette technique permet d'appliquer des motifs extrudés ou biseautés sur une surface 3D. Dans cette approche, nous nous sommes basés uniquement sur une texture RVBA, où l'on stocke : La forme de base, sa transformation distance et son gradient. Notre technique a l'avantage d'être très rapide et efficace à la différence des méthodes traditionnelles qui ne sont guère adaptées à l'extrusion de formes. De plus, La technique proposée ne se limite pas à l'extrusion de petits motifs, mais permet également de créer et d'afficher correctement des objets 3D entiers obtenus avec l'extrusion ou la révolution d'une forme de base. Nous pouvons ainsi générer des objets complexes à l'aide d'un maillage très basique, une simple boite en l'occurrence.

Organisation du document

Ce mémoire est organisé de la manière suivante :

Dans le premier chapitre, après une introduction détaillée, nous commencerons par un rappel de quelques notions fondamentales concernant le rendu, et en particulier, le rendu temps réel. Nous allons ainsi définir le pipeline graphique qui désigne l'ensemble des étapes permettant de passer d'une description abstraite de la scène à une image discrète sur l'écran. Nous aborderons ensuite les techniques d'ombrage et de placage de textures, car elles jouent un rôle majeur dans le placage de déplacement 3D par pixel. Dans la seconde partie du premier chapitre, nous aborderons le sujet principal de cette thèse, à savoir : Le placage de déplacement tridimensionnel par pixel. Le principe général et le fondement mathématique seront clairement définis.

La troisième partie du premier chapitre sera consacrée à notre approche qui permet la synchronisation de l'ombrage avec le changement d'échelle de la profondeur des mesostructures. Nous commencerons par évoquer les limitations des techniques existantes, avant de proposer deux solutions permettant de les surmonter.

Le deuxième chapitre sera consacré à l'état de l'art concernant le placage de déplacement 3D par pixel. Nous allons évoquer la majorité des solutions qui ont été proposées pour calculer le point d'intersection entre le rayon de vue et le microrelief généré à partir de la carte de profondeurs. Nous discuterons également des techniques permettant la prise en charge de l'auto-ombrage et de la silhouette. Nous présenterons ensuite les techniques de modélisation et rendu à base d'images qui permettent la représentation de modèles 3D entiers sans maillage.

Le troisième chapitre sera consacré à notre contribution relative à la technique de traçage de cônes, qui reste actuellement l'une des meilleures techniques pour le placage de déplacement par pixel. Nous commencerons par une description détaillée des algorithmes de rendu relatifs aux deux versions de cette technique (conservative et relaxée). Dans la deuxième partie, nous présenterons les algorithmes de la phase du prétraitement où s'effectue le calcul de la carte de déplacement (ou carte de cônes dans ce cas). La dernière partie de ce chapitre sera consacrée à la présentation des résultats de nos méthodes, notamment en les comparant aux techniques de référence.

Dans le quatrième chapitre, nous présenterons une nouvelle technique dans le domaine du placage de déplacement 3D par pixel. Il s'agit du placage d'extrusion et de révolution. Nous commencerons par une description générale de cette technique, avant de présenter la phase de prétraitement qui consiste à calculer la carte de forme. Ensuite, nous aborderons les différents algorithmes de rendu (placage d'extrusion, de biseau, de chanfrein et de révolution). La section suivante sera consacrée à la création d'objets complets, extrudés ou révolus, à l'aide d'une boite de forme. Enfin, dans la dernière partie de ce chapitre, nous présenterons les résultats obtenus, de même que les temps d'exécutions des différents algorithmes.

Chapitre 1

Placage de Déplacement 3D Par Pixel : Principe et Généralités

Le placage de déplacement tridimensionnel par pixel est une technique de rendu qui consiste à réduire le nombre de primitives graphiques (polygones et sommets) constituant une scène 3D tout en conservant la qualité visuelle globale de la scène. Pour réaliser cet objectif, cette technique se base sur des textures et sur un algorithme de type "lancer de rayons" qui s'exécute sur les unités programmables du processeur graphique. Dans ce chapitre, nous allons commencer par évoquer quelques prérequis nécessaires à la réalisation de cette technique, à savoir : Le pipeline graphique, les modèles d'ombrage et le placage de textures. Ensuite, nous décrirons brièvement le principe du placage de déplacement 3D par pixel, avant de présenter le fondement mathématique sur lequel repose cette technique. Finalement, nous présenterons notre approche concernant la synchronisation interactive entre le changement d'échelle de la profondeur du microrelief et l'ombrage de ce dernier.

1.1 Introduction

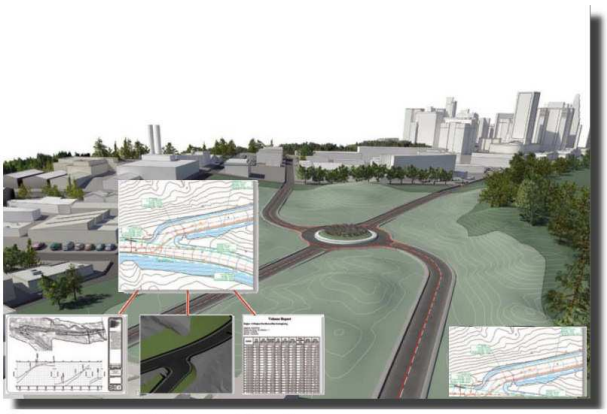
L'infographie a connu ces dernières années un essor incroyable grâce à l'évolution exponentielle du matériel informatique et graphique, mais aussi grâce à l'engouement que portent les professionnels et le grand public envers les images de synthèse et la 3D en particulier. Dans ce contexte, les champs d'application de la 3D se sont constamment diversifiés pour atteindre presque tous les domaines d'activité (voir figure 1.1). Parmi ces disciplines, nous pouvons citer :

- **La conception et modélisation 3D assistées par ordinateur** : L'époque des croquis et des maquettes en bois, en miniatures ou grandeur nature, est définitivement révolue. Actuellement, tous les engins (voitures, Trains, bateau, avions) sont conçus à l'aide d'applications de gestion du cycle de vie d'un produit ou PLM (*Product Lifecycle Management*). Ce type d'application intervient dans tous les stades de la production. De la conception à la maintenance, en passant par la fabrication et la simulation. La conception ne se limite pas seulement aux grosses machines, mais concerne évidemment tous les produits manufacturés, que ce soit dans le domaine de l'ingénierie ou du design.
- **Génie civil et Architecture** : Les architectes et les professionnels du bâtiment étaient parmi les premiers à adopter l'infographie 3D. Ils ont tout de suite compris les avantages immenses qu'offrait cette nouvelle technologie. Les logiciels actuels permettent de concevoir un bâtiment ou un ouvrage dans son intégralité. Les différentes étapes de modélisation aboutissent à une maquette virtuelle du projet et qui, contrairement à une maquette en bois, constitue une véritable miniature numérique de l'édifice (architecture, structure d'acier, structure du béton armé, tuyauterie, espaces verts, etc.). Cette maquette est personnalisable à volonté et les erreurs de conception peuvent être repérées et corrigées avant la phase de réalisation.
- **Le domaine de la santé** : Les techniques d'imagerie médicale sont de plus en plus performantes, rapides et précises. Les images 2D et 3D de l'organisme humain et de son fonctionnement sont acquises via plusieurs types de scanner (résonance magnétique, rayons x, rayons gamma, ultrasons). Par la suite, des applications dédiées permettent la visualisation des données, leur analyse et leur interprétation. Elles permettent, entre autres, l'aide au diagnostic, la préparation d'opérations chirurgicales et le suivi thérapeutique. La possibilité d'effectuer des interventions chirurgicales virtuelles est un outil formidable,

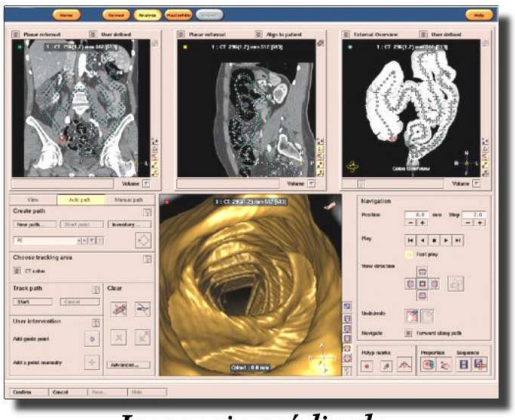
permettant aux étudiants en médecine de s'exercer plus fréquemment, et sans risque pour les patients.

- **Communication commerciale, culturelle et pédagogique** : Les animations 3D attirent très facilement l'attention. Les publicitaires l'ont bien compris car la 3D fait partie intégrante des spots publicitaires ou des communications d'entreprises surtout depuis l'avènement de l'Internet. Le domaine culturel a su également tirer profit de cette nouvelle technologie, ainsi nous commençons à voir des musées virtuels interactifs. Nous pouvons également effectuer des visites virtuelles de sites historiques ou de monuments disparus. L'enseignement et l'apprentissage par le biais de la 3D sont aussi très prometteurs. En effet, la simulation 3D est probablement la manière la plus simple pour comprendre la structure du système solaire ou le fonctionnement d'un moteur à combustion par exemple
- **Divertissement** : Le cinéma, les films et série télévisés, les documentaires ou les dessins animés sont de plus en plus consommateurs d'images de synthèse, que ce soit pour des séquences entièrement réalisées en 3D (film d'animations par exemple), ou pour les effets spéciaux (animation et simulation de foule, destruction de bâtiments, simulation de fluides…). Quant aux jeux vidéo, ils reposent entièrement sur l'infographie 3D. Cette industrie, longtemps marginale, est en train de bouleverser les traditionnelles industries du divertissement. En effet, le jeu *GTA IV* des studios *Rockstar*® par exemple, est devenu en 2008 le produit culturel le plus vendu au monde.

Cependant, tous ces domaines d'activité n'utilisent pas l'infographie 3D de la même manière. En effet, chaque discipline a ses propres contraintes, ses propres moyens matériels et ses propres objectifs. L'infographie 3D a du adapter ses techniques et ses algorithmes afin de satisfaire d'une manière optimisée toutes ces disciplines. Ainsi, la médecine s'appuie par exemple sur l'imagerie 3D. Le Cinéma, dont le réalisme est l'objectif majeur, adopte des techniques très avancées comme le lancer de rayons ou la radiosité. Quant à la simulation, l'immersion virtuelle ou les jeux vidéo, leur contrainte d'interactivité et de rapidité les a poussé à adopter un ensemble de techniques très optimisées dites *techniques de rendu temps réel*.

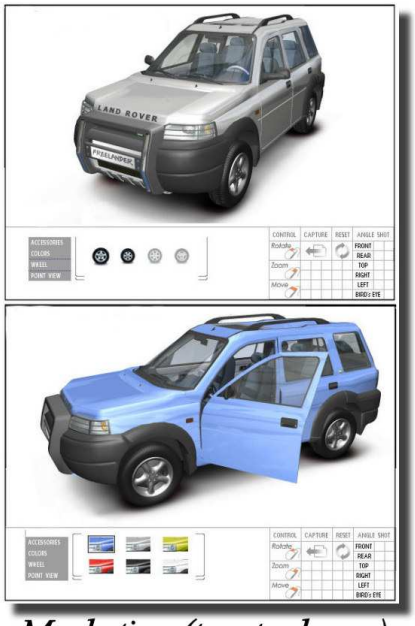


Génie Civil (autodesk.fr)



Imagerie médicale (medical.philips.com)

PLM - Aviation (3ds.com)



Marketing (turntool.com)

Jeux Vidéo (ea.com)

Cinéma (massivesoftware.com)

Fig. 1.1 - Exemple de quelques secteurs d'activité qui s'appuient fortement sur l'infographie 3D

D'une manière générale, l'imagerie de synthèse 3D est déclinée en deux grandes spécialités : le rendu pré-calculé et le rendu temps réel.

- **Le rendu pré-calculé**: Appelé également rendu différé, rendu post-production ou rendu réaliste. Il a comme principal objectif, la création d'images et de vidéos de très haute qualité. Le temps de calcul n'est pas le facteur déterminant, et tous les algorithmes utilisés par ce type de rendu mettent plutôt l'accent sur le réalisme, que ce soit au niveau des détails géométriques (maillages denses, souvent générés à partir de formules mathématique comme les surfaces NURBS), mais aussi au niveau de la réflexion/réfraction (par le biais du lancer de rayon) ou encore au niveau de l'ambiance globale (algorithme de la radiosité). Tous ces algorithmes sont très complexes et nécessitent plusieurs passes de traitement. Par conséquent, le temps de calcul d'une seule image peut prendre plusieurs minutes voir des heures. C'est pour cette raison que le rendu pré-calculé n'est utilisé que dans les domaines qui ne nécessitent pas d'interaction avec l'utilisateur, comme le cinéma, la publicité, les communications d'entreprises ou les modélisations scientifiques. Il faut noter que le calcul du rendu pré-calculé est effectué entièrement par les processeurs centraux, car les cartes graphiques ne sont guère adaptées à ce type de rendu.

- **Le rendu temps réel** : Appelé parfois rendu interactif, il est à l'opposé du rendu pré-calculé, car le facteur déterminant dans ce cas est la vitesse de calcul. Le rendu interactif ne génère pas de séquences vidéo, mais des animations en temps réel sur l'écran. Or, nous savons que pour pouvoir percevoir une animation fluide, il faut que le rendu calcule au moins 16 images par seconde. Cette contrainte limite sérieusement la qualité des images obtenues car tous les algorithmes sont optimisés pour la vitesse aux dépens de la qualité. Pour cette raison, le calcul du rendu est délégué presque entièrement au processeur graphique. Parmi les domaines faisant appel au rendu temps réel, nous pouvons citer : la conception graphique, la simulation 3D, les visites virtuelles et les jeux vidéo.

Néanmoins, les frontières entre le rendu pré-calculé et le rendu interactif ne sont pas aussi nettes. En effet, ces deux spécialités partagent plusieurs algorithmes et techniques de programmation. D'un autre coté, les domaines d'application ne se limitent souvent pas à une seule forme de rendu, comme le cinéma par exemple où les infographistes ont besoin d'outils de rendu temps réel pour concevoir et prévisualiser les scènes avant de lancer le calcul du rendu final qui peut nécessiter plusieurs jours !

L'apparence des modèles 3D est constituée généralement d'une structure à trois niveaux, ces derniers peuvent être définis comme suit:

- **Macrostructure** : Elle définit la structure globale de l'objet. Elle est créée à l'aide d'un maillage constitué d'un ensemble de polygones (généralement triangulaires) connectés par des sommets 3D.
- **Microstructure** : Elle ne participe pas à la définition géométrique de l'objet mais donne uniquement sa couleur, celle-ci est influencée dans le monde réel par le type de surface (qui diffère suivant la microgéométrie). La couleur est calculée à partir d'une texture 2D, 3D ou procédurale, à l'aide d'une formule d'ombrage faisant appel à différents paramètres liés à la fois à l'objet et à la scène. Le bois et le plastique par exemple, ne réfléchissent pas la lumière de la même manière, même si nous leur appliquons la même texture.
- **Mesostructure** : Le mot latin *meso* signifiant *milieu* ou *entre*, nous comprenons qu'il s'agit d'une structure intermédiaire entre la macrostructure et la microstructure. En effet, il s'agit des microreliefs qui couvrent la surface d'un objet. Ces derniers sont très petits pour être incorporés au maillage, car un maillage trop dense est très lourd à gérer, mais d'un autre coté, ils sont nettement visibles pour être simplement simulés à l'aide d'une formule d'ombrage.

Le cas des mesostructures a donc constitué un problème majeur, particulièrement pour le rendu temps réel. Blinn a proposé en 1978 une solution basée sur l'ombrage [Bli78]. Cette technique, appelée *placage de bosselures*, effectue une perturbation des normales de la surface 3D afin d'induire une impression de relief. Cette perturbation (par rotation des normales) est calculée à partir d'un microrelief stocké sous forme d'une image monochrome appelée *carte de hauteurs* (ou *profondeurs* suivant l'interprétation[1]). Des méthodes ultérieures ont pré-calculé les normales et les ont stockées sous forme d'une image RVB, d'où le terme *placage de normales* utilisé parfois pour désigner le placage de bosselures.

Même si le placage de bosselures est toujours la technique la plus utilisée pour le rendu des mesostructures, elle demeure néanmoins une solution basique, s'appuyant uniquement sur l'ombrage. La géométrie reste alors invisible, de même que l'effet parallaxe qui simule les distorsions de la texture dues au microrelief. Le placage de bosselures ne gère pas non plus l'auto-ombrage ni la silhouette du microrelief.

[1] Dans ce mémoire, nous utiliserons plutôt la notion de profondeur car la majorité des techniques actuelles placent le relief sous la surface des polygones.

Partant de ce constat, Cook a introduit une technique appelée *placage de déplacement* [Coo84]. Cette technique a permis de masquer quasiment toutes les limitations du placage de bosselures, car elle procède à une densification du maillage. Ce dernier est subdivisé puis extrudé suivant les normales aux sommets en s'appuyant sur la carte de hauteurs. La création du maillage additionnel n'est effectuée qu'au moment de l'affichage. Ainsi, la géométrie créée est aussitôt supprimée après son traitement par le pipeline graphique. Cependant, vu la taille du maillage à traiter, cette technique n'a été utilisée que dans le cadre du rendu pré-calculé. Dernièrement, les cartes graphiques ont intégré cette fonctionnalité au processeur graphique[1], mais son usage reste relatif.

Afin de contourner la densification du maillage effectuée par le placage de déplacement, Patterson et al. [PHL91] ont proposé une solution plus intéressante pour le rendu temps réel appelée *placage de déplacement inverse* ou *placage de déplacement par pixel*. Cette technique repose à la fois sur l'ombrage du placage de bosselures, et sur le déplacement de la surface de base en fonction d'une carte de profondeurs. Cependant, le déplacement dans ce cas se fait au niveau des pixels et non plus au niveau des sommets. Par conséquent, aucune densification du maillage n'est effectuée, sauf dans certaines méthodes qui créent une coquille autour du maillage de base. Le placage de déplacement 3D par pixel se base également sur le traçage de rayon ou lancer de rayon, mais seulement sous une forme simplifiée, car le traçage se fait au niveau de la carte de profondeurs et non au niveau de la scène globale.

La figure 1.2 et le tableau 1.1 donnent une comparaison entre le rendu d'un modèle avec mesostructures, en s'appuyant d'abord sur un maillage très dense, puis en utilisant une version simplifié de celui-ci en combinaison avec l'une des techniques de placage de déplacement par pixel. Le goulot d'étranglement, causé par le nombre important de polygones dans le premier cas (1.5 million), fait chuter considérablement la vitesse du rendu: 32 images par secondes seulement contre 235 dans le deuxième cas.

Sur la base du placage de déplacement par pixel, certaines méthodes ont été introduites pour pouvoir créer des objets 3D en intégralité, c-à-d sans maillage sauf sous une forme basique comme des plans ou des boites. La forme géométrique de l'objet est générée à partir de textures 2D ou 3D. Ces techniques, dites de *modélisation et rendu à base d'images*, sont très prometteuses, notamment dans le cas où les scènes sont très détaillées, à tel point qu'une représentation avec maillages pourrait provoquer une saturation du matériel graphique.

[1] Introduite d'abord par Matrox® via les N-Patch, elle est actuellement assurée par le nuanceur de géométrie introduit dans les spécifications de DirectX® 10.

Cependant, ces techniques sont encore dans un stade de développement, et nécessitent davantage d'optimisation.

Fig. 1.2 - Comparaison entre la modélisation par un maillage polygonal dense (à gauche), et la modélisation à l'aide d'un maillage simplifié combiné à une technique de placage de déplacement 3D par pixel (à droite). Les modèles réduits sont combinés avec le rendu en fil de fer afin de mettre en évidence la densité du maillage [Tat06b].

	Maillage polygonal dense	**Maillage polygonal simplifié avec placage de déplacement par pixel**
Nombre de polygones	**1 500 000**	**1 100**
Occupation mémoire	Tableau des sommets : 31 Mo Tableau des polygones : 14 Mo **Total : 45 Mo**	Tableau des sommets : 79 Ko Tableau des polygones : 6 Ko Texture de déplacement : 13 Mo **Total : 13,1 Mo**
Vitesse d'affichage	**32** images par secondes	**255** images par secondes (235 avec placage de texture)

Tab. 1.1 - Comparaison entre les performances de la modélisation par un maillage polygonal dense, et la modélisation par un maillage simplifié combiné à une technique de placage de déplacement par pixel. Cette dernière est beaucoup moins gourmande en terme d'occupation mémoire, en plus, elle est nettement plus rapide tout en conservant une qualité de rendu assez proche de celle du maillage dense [Tat06b].

1.2 Pipeline graphique

Avant de pouvoir étudier et implémenter les différentes techniques de placage de déplacement tridimensionnel par pixel, il est nécessaire d'étudier l'architecture du pipeline graphique adopté conjointement par les développeurs des bibliothèques graphiques (OpenGL de Silicon Graphics et Direct3D de Microsoft) et les fabricants des puces graphiques (NVIDIA, AMD/ATI, Intel, S3 Graphics). Il faut aussi étudier en détail la bibliothèque graphique OpenGL 2.0 ainsi que son langage de programmation à traitement parallèle GLSL, celui-ci permet d'écrire des programmes destinés au processeur graphique. La solution OpenGL/GLSL a été retenue parce qu'elle est

multi-plateforme et non commerciale. Il faut noter qu'une carte graphique de nouvelle génération est indispensable.

Le pipeline graphique désigne l'ensemble des étapes permettant de passer d'une représentation structurée de données tridimensionnelles, stockées en mémoire, à une image 2D fixe ou animée affichée sur un écran[1]. Dans le cas du rendu temps réel, qui nécessite une vitesse de traitement très rapide[2], le pipeline graphique est assuré par différents éléments logiciels et matériels, dont le plus important est la carte graphique, ou plus précisément, le processeur graphique GPU (*Graphics Processing Unit*). Ce dernier adopte actuellement une architecture massivement parallèle dédiée principalement au traitement de données 3D (voir figure 1.3).

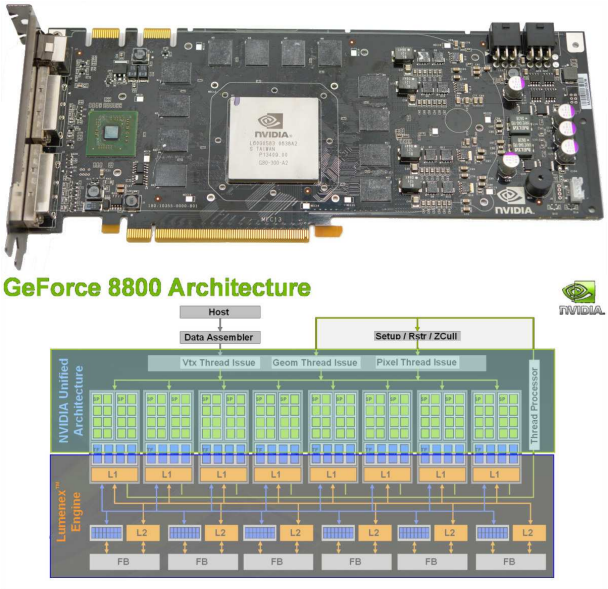


Fig. 1.3 - Architecture des cartes graphiques. En haut : la carte graphique NVIDIA GeForce 8800 avec au centre, le GPU. En bas : Schéma de l'architecture massivement parallèle du GPU (ce modèle se compose de 128 unités d'exécution parallèles !)

Une scène 3D comporte plusieurs objets différents. Ces objets sont constitués d'un certain nombre de sommets et d'un ensemble de primitives graphiques indiquant la façon dont ces sommets sont connectés entre eux. La figure 1.4 illustre les dix types de primitives définies dans la bibliothèque graphique OpenGL, où l'on constate que le matériel graphique peut rendre des sommets isolés, des segments ou un groupe de polygones. Néanmoins, la grande majorité des modèles 3D sont définis par un maillage triangulaire, où chaque triangle (ou facette) fait référence à trois sommets dans une liste regroupant tous les sommets du maillage.

[1] Pour plus de détails sur les algorithmes de base, le lecteur pourra consulter l'ouvrage [Len04]

[2] 16 IPS (Images Par Seconde) étant un minimum, le standard PAL/SECAM est à 25 IPS et le NTSC est à 30 IPS.

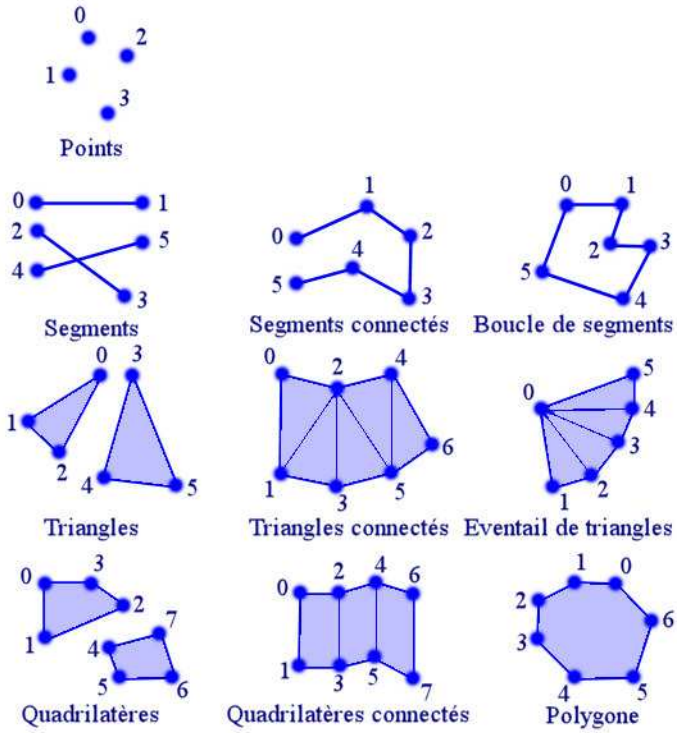


Fig. 1.4 - Les dix types de primitives définies dans OpenGL. Les nombres indiquent l'ordre dans lequel chaque primitive doit être spécifiée.

Les cartes graphiques actuelles possèdent une unité de traitement graphique dédiée (le GPU) qui exécute des instructions indépendamment de l'unité de traitement centrale CPU (*Central Processing Unit*). Le CPU envoie les commandes du rendu graphique au GPU qui exécute les instructions associées. Pendant ce temps, le processeur est libre et peut effectuer d'autres taches. Ce mode de fonctionnement s'appelle *le traitement asynchrone*.

Les applications communiquent avec le GPU en envoyant des commandes à des bibliothèques graphiques ou API 3D (*Application Programming Interface*), telles que OpenGL ou Direct3D. Dans la suite de cette description, nous nous intéresserons plutôt à OpenGL qui est une bibliothèque libre et multi-plateforme. OpenGL envoie à son tour des commandes au pilote de la carte graphique qui est le seul à savoir communiquer avec le GPU dans son langage natif (instructions machine). L'interface d'OpenGL est appelée *couche d'abstraction matériel* ou HAL (*Hardware Abstraction Layer*), parce qu'elle dispose d'un certain nombre de fonctions graphiques qui peuvent être exécutées qu'elle que soit l'architecture du matériel graphique. Le pilote de la carte graphique se charge lui, de traduire ces fonctions en instructions exécutables par le GPU. Le diagramme de la figure 1.5 illustre la communication qui s'établit entre le CPU, le GPU, la mémoire centrale et la mémoire vidéo.

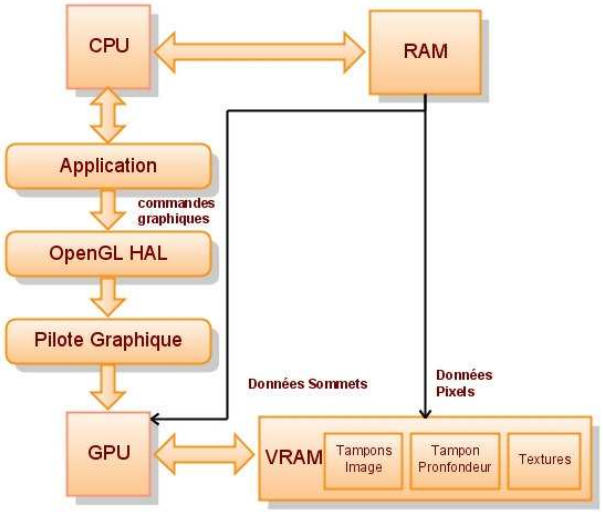


Fig. 1.5 - Communication entre le CPU, le GPU et les mémoires associées

La mémoire vidéo, plus connue sous le nom de VRAM (*Video Random Access Memory*) est soit intégrée aux cartes graphiques indépendantes, soit partagée avec la mémoire centrale, notamment dans le cas des cartes graphiques intégrées à la carte mère. La VRAM permet de stocker différents éléments relatifs au rendu graphique. Parmi ces éléments, il y a plusieurs tampons d'images. Le tampon couleur est sans doute le plus important puisqu'il contient l'image finale à afficher. Ce tampon est doublé dans le cas d'une animation : Le tampon avant (*Front Buffer*) abrite l'image calculée qui est affichée actuellement sur l'écran, tandis que le tampon arrière (*Back Buffer*) reçoit les pixels qui sont en train d'être calculés pour la prochaine image. Sans ce mode de fonctionnement, l'animation sera parasitée par des scintillements très gênants. La VRAM abrite également un tampon de profondeurs (*Z-Buffer*). Ce tampon stocke, pour chaque pixel de l'image, la profondeur vis-à-vis de la caméra. Il intervient dans le processus d'élimination des parties cachées. Ainsi, un pixel ne peut écraser un autre sauf s'il a une profondeur plus petite. Il existe un autre tampon appelé *Stencil* qui sert parfois à masquer des portions de l'image ou tout autre traitement particulier, comme la gestion des ombres par exemple.

Enfin, la majeure partie de la VRAM est occupée par les textures. Les textures sont en général plaquées sur les facettes du maillage en utilisant des coordonnées spécifiques à cette fonction, appelées *coordonnées de texture*. Le placage de texture permet d'avoir des détails visuels très riches (bois, marbre, gazon...), mais également des effets plus complexes comme l'ajout de microreliefs aux maillages sans en augmenter la densité. Cette fonctionnalité est bien entendu le sujet de la présente thèse.

1.2.1 Transformation des sommets

La transformation des sommets est l'étape permettant de transformer les données tridimensionnelles et continues passées au GPU, en données discrètes en deux dimensions pouvant être affichées sur l'écran. Le pipeline graphique est associé à plusieurs systèmes de coordonnées. La figure 1.6 illustre la relation entre ces différents espaces. Un modèle 3D est le plus souvent défini dans un espace qui lui est propre, appelé *l'espace objet*[1]. Par exemple, les coordonnées des sommets d'une sphère sont définies par rapport à son centre. Cependant, la position, l'orientation et l'échelle de chaque objet sont définies par rapport à un espace commun appelé *espace global*[2].

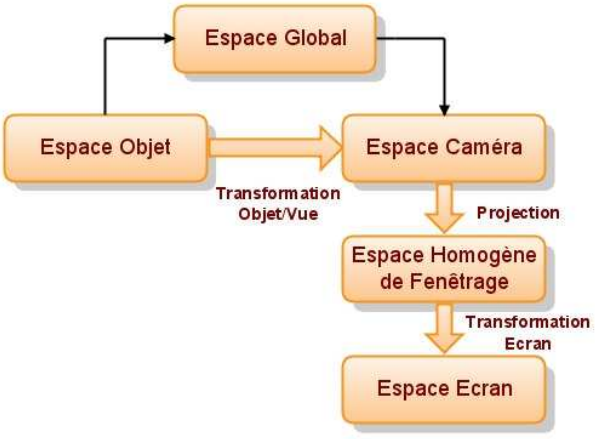


Fig. 1.6 - Les différents espaces intervenant dans le pipeline graphique. Le matériel graphique reçoit des sommets exprimés dans l'espace objet, puis les transforme dans l'espace écran.

Après la transformation des sommets dans l'espace global, ils subissent une autre transformation. Celle du passage vers l'*espace caméra* (ou *espace vue*). Cette étape à pour objectif d'aligner l'axe *Z* du repère global avec celui de la caméra. Il faut noter qu'il est possible de transformer les sommets directement de l'espace objet vers l'espace caméra en concaténant les matrices de transformations correspondantes. Cette transformation directe est appelée *transformation objet/vue*. D'ailleurs, OpenGL adopte cette dernière solution.

Une fois que les sommets de l'objet sont transformés dans le repère caméra, ils subissent une transformation de projection perspective[3]. La géométrie apparaît alors de plus en plus petite au fur et à mesure qu'elle s'éloigne de la caméra. Contrairement à ce qu'on pourrait penser, cette

[1] Des fois il est désigné par : espace local ou espace du modèle

[2] On parle aussi d'espace monde ou espace de la scène

[3] Une projection orthographique est également prise en compte, car elle s'avère parfois très utile, notamment pendant la modélisation, ou en dessin industrielle et architecturale.

transformation ne génère pas des coordonnées 2D, car les sommets projetés garde leur profondeur (coordonnées *z*). Cette dernière sera utilisée dans le processus de l'élimination des faces cachées. La projection est effectuée en coordonnées homogènes. L'espace dans lequel seront définis les sommets après cette projection est appelé *espace homogène de fenêtrage*. Il est appelé ainsi, parce que les primitives géométriques subiront un fenêtrage 3D, afin de ne garder que les primitives se trouvant à l'intérieur de la zone visée par la caméra. Celle-ci est représentée par une pyramide tronquée (figure 1.7). Finalement, les sommets subissent une dernière transformation, qui est le passage vers le repère discret de l'écran.

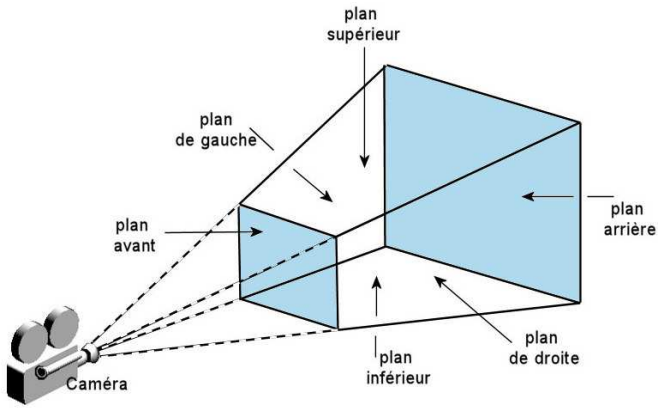


Fig. 1.7 - Pyramide tronquée représentant la zone visible qui est limitée par six plans.

1.2.2 Transformation de pixel

Une fois les sommets projetés sur l'écran, les primitives constituées par ces sommets doivent être restituées. L'étape qui consiste à procéder au remplissage des primitives, d'une manière horizontale ligne par ligne, est appelée *rastérisation*. Pendant le remplissage, le GPU interpole les données associées aux sommets, comme la profondeur, la couleur, la normale, la binormale, la tangente, ou encore les coordonnées de texture. L'ensemble de ces données, plus la position du remplissage courante, forment une entité appelée *pixel* ou *fragment*[1].

Le processus au travers duquel une primitive est convertie en un ensemble de pixels est illustré sur la figure 1.8. D'abord, si l'application le demande, une sélection de facettes est appliquée aux primitives tournant le dos à la caméra. La sélection s'appuie sur les normales des facettes. Cette étape est une optimisation importante, puisqu'elle permet d'éliminer un très grand nombre de primitives.

[1] Parfois cette entité est appelée fragment afin de la différencier d'un pixel de l'écran ou d'une image.

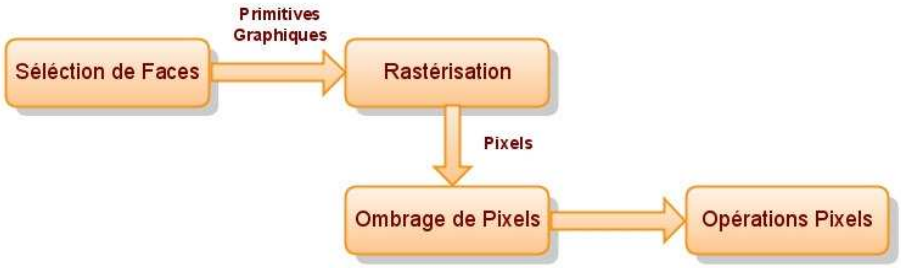


Fig. 1.8 - Les primitives graphiques sont converties en un ensemble de pixels durant la rastérisation. Après l'étape d'ombrage, les pixels subissent un certain nombre d'opérations qui déterminent principalement leur visibilité.

L'application doit également spécifier comment les données des pixels doivent être utilisées pour déterminer la couleur finale. Ce processus est appelé *ombrage de pixels*. Dans sa forme la plus utilisée (ombrage de Gouraud), il s'agit simplement de combiner la couleur interpolée à partir de celle des sommets, avec celle obtenue à partir de la texture appliquée à la primitive. L'accès à la texture s'effectue via les coordonnées de texture interpolées également à partir de celles des sommets. Néanmoins, les GPU récents permettent un ombrage plus complexe comme l'ombrage de Phong.

La figure 1.9 montre les opérations effectuées pour chaque pixel généré pendant la rastérisation. La plupart de ces opérations consistent à déterminer si un pixel doit être affiché ou ignoré. Logiquement, ces opérations interviennent après le calcul d'ombrage, mais ce dernier étant devenu de plus en plus complexe, les récents GPU tendent à les implémenter en premier. Ainsi, ils permettent d'éviter des calculs complexes concernant l'ombrage de pixels qui seront de toute façon éliminés plus tard.

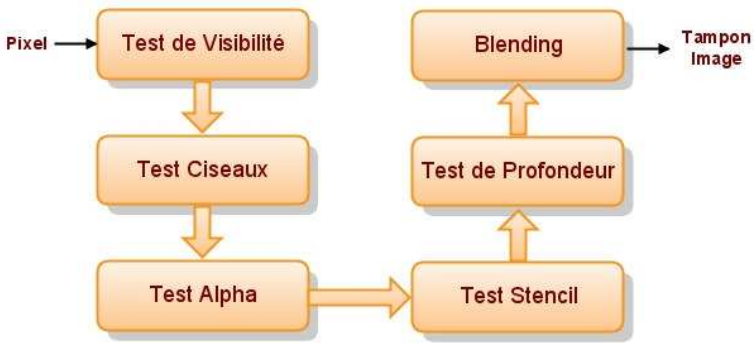


Fig. 1.9 - Opérations effectuées avant qu'un pixel ne puisse être affiché sur l'écran.

Un pixel subit tout d'abord un *test de visibilité système*. Cela signifie que si une autre fenêtre du système est devant lui, il sera ignoré. Ensuite, le *test ciseaux* permet d'éliminer les pixels se trouvant hors d'un rectangle défini par l'application. Le *test alpha* permet quant à lui d'éliminer certains pixels suivant leur valeur de transparence. Le *test stencil* est effectué en éliminant les

pixels non désirés repérés par un bit 0 dans le tampon stencil. Arrive ensuite le *test de profondeur* qui permet de supprimer les pixels dont la profondeur est supérieure à celle déjà présente dans le tampon de profondeur. Enfin l'étape du *Blending* (mélange) permet de définir la façon avec laquelle se fera le mélange entre la couleur du nouveau pixel, et celle déjà calculée à la même position dans le tampon couleur. C'est à ce niveau également que la transparence est appliquée en fonction des valeurs alpha.

1.2.3 Pipeline programmable

Comme nous venons de décrire, le pipeline graphique est implémenté par le processeur graphique suivant un enchaînement d'étapes clairement établi et rigide. Il n'est donc pas possible de le modifier, sauf par la manipulation de quelques variables d'états via les bibliothèques 3D. Ce pipeline fixe offre certes des performances remarquables, mais la personnalisation du rendu 3D reste très limitée. Pour cette raison, le matériel de nouvelle génération a introduit la possibilité d'intervenir et de programmer certaines étapes du pipeline graphique. Il s'agit en l'occurrence, de l'étape de transformation des sommets, et celle du calcul de l'ombrage des pixels après la rastérisation.

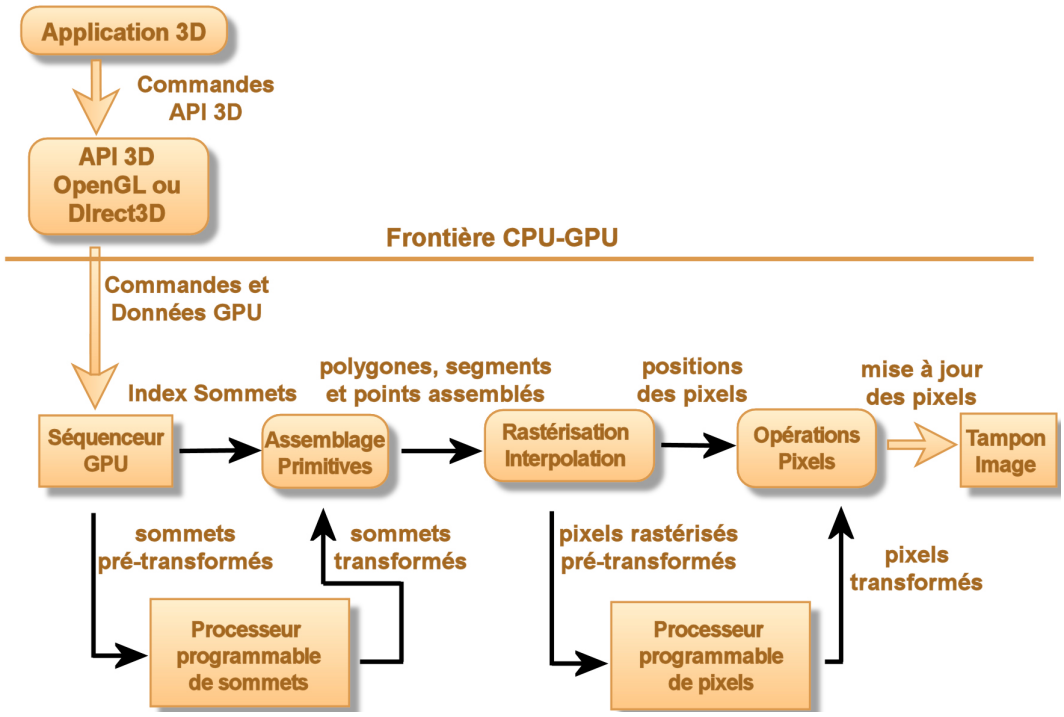


Fig. 1.10 - Positionnement des unités programmables dans le pipeline graphique. La première unité est le processeur de sommets qui permet d'effectuer des opérations sur les sommets. La deuxième unité est le processeur de pixels qui est utilisé pour le traitement des pixels résultants de la rastérisation.

Le GPU dispose désormais de deux ensembles d'unités de traitement supplémentaires[1], appelées *processeur de sommets* et *processeur de pixels* (voir la figure 1.10). Chaque ensemble dispose de plusieurs unités à traitement parallèle. Ces deux processeurs sont chargés d'exécuter des programmes spéciaux appelés *nuanceurs*. Le processeur de sommets exécute le nuanceur de sommets, qui opère sur les sommets pour exécuter des opérations comme les transformations, la projection ou l'interpolation de données. Quant au processeur de pixels, il se charge de l'exécution du nuanceur de pixels qui s'exécute après la rastérisation et l'interpolation associée. Il permet notamment le calcul de la couleur d'un pixel en fonction des données interpolées.

Au début, les nuanceurs étaient écrits à l'aide d'un langage machine, semblable au langage assembleur du CPU. Mais avec les évolutions récurrentes des GPU, les nuanceurs devenaient de plus en plus complexes. Des langages de haut niveau devenaient alors indispensables, parmi lesquels nous pouvons citer :

- RSL (*RenderMan Shading Language*) : il est le précurseur mais il est spécifique au lancer de rayons. Il ne tire donc aucun bénéfice du matériel graphique. Cependant, il a fortement influencé la programmation du rendu temps réel, ainsi que le développement des GPU.
- SSL (*Stanford Shading Language*) : Développé à l'université de Stanford, il est le premier langage de haut niveau pour le temps réel. Il s'intègre à OpenGL en ajoutant une couche supérieure d'abstraction. Par contre, il ne fait pas de distinction entre le nuanceur de sommets et le nuanceur de pixels.
- Cg (*C for graphics*) : Langage propriétaire du fabriquant des puces graphiques NVIDIA, c'est un langage générique et multi-nuanceurs (sommets et pixels). Il est surtout indépendant des bibliothèques graphiques (OpenGL et DirectX).
- HLSL (*High level Shading Language*) : Langage propriétaire de Microsoft. Il est quasi identique au langage Cg (collaboration avec NVIDIA). Il est toutefois lié exclusivement à l'API DirectX.
- GLSL (Open**GL** **S**hading **L**anguage) : Ce langage est également basé sur la syntaxe du langage C. Il a été développé par le consortium *OpenGL Architecture Review Board.* Il est complètement intégré à OpenGL depuis sa version 2.0. Il a l'avantage d'être indépendant des cartes graphiques et des systèmes d'exploitation à l'instar d'OpenGL. C'est ce langage que nous avons retenu pour l'implémentation de nos nuanceurs.

[1] Une troisième unité a été récemment introduite dans les spécifications de la bibliothèque DirectX® 10. Il s'agit du processeur de géométrie qui permet (entre autres) d'effectuer un placage de déplacement matériel.

1.3 Modèles d'ombrage et placage de textures

Avant de pouvoir aborder la technique du placage de déplacement 3D par pixel, il est impératif de s'attarder sur le calcul d'ombrage et le placage de textures qui s'effectuent pendant la phase de rastérisation.

1.3.1 Calcul d'ombrage

Le calcul d'ombrage[1] est l'opération qui consiste à calculer la couleur de chaque pixel généré pendant la phase de rastérisation, en utilisant un modèle d'illumination local. Le modèle le plus simple est le modèle de Lambert, dont la formule est donnée par :

$$I = I_{ambiante} + I_{diffuse}$$
$$I = C_a K_a + C_d K_d \cdot \max(\vec{L} \cdot \vec{N}, 0)$$

où $\vec{N}$ est la normale à la surface. $\vec{L}$ est la direction de la lumière. C_a et K_a sont respectivement, les composantes ambiantes de la lumière et du matériau (où du texel). De même, C_d et K_d sont les composantes diffuses.

Le modèle de Lambert ne prenant pas en considération la lumière spéculaire, d'autres modèles ont été proposés, dont celui de Phong, défini par la formule suivante:

$$I = I_{ambiante} + I_{diffuse} + I_{spéculaire}$$
$$I = C_a K_a + C_d K_d \cdot \max(\vec{L} \cdot \vec{N}, 0) + K_s C_s \max((\vec{L} \cdot \frac{\vec{R}}{\|\vec{R}\|})^n, 0)$$

où $\vec{V}$ désigne le vecteur de vue, $\vec{R}$ est la direction de la réflexion de $\vec{V}$: $\vec{R} = \vec{V} - 2\vec{N}(\vec{N} \cdot \vec{V})$. C_s et K_s sont les composantes spéculaires de la lumière et du matériau. L'exposant n désigne la réflectivité (ou lustre spéculaire).

Le modèle de Blinn-Phong est également très utilisé, il est donné par :

$$I = I_{ambiante} + I_{diffuse} + I_{spéculaire}$$
$$I = C_a K_a + C_d K_d \cdot \max(\vec{L} \cdot \vec{N}, 0) + K_s C_s \max((\vec{H} \cdot \vec{N})^n, 0)$$

où $\vec{H}$ est le vecteur demi-angle, calculé par la relation suivante :

$$\vec{H} = \frac{\vec{L} + \vec{V}}{\|\vec{L} + \vec{V}\|}$$

[1] A ne pas confondre avec le calcul des ombres portées

Les modèles d'ombrage se différencient par le niveau dans lequel s'effectue l'évaluation de la formule d'illumination pendant la phase de rastérisation (voir la figure 1.11). Il existe trois modèles d'ombrage :

- **Modèle de Lambert** : Appelé parfois *modèle plat*, il commence par calculer la couleur de la facette, supposée plate, en fonction de sa normale. Puis, applique cette même couleur à tous les pixels de la facette.
- **Modèle de Gouraud** : C'est un modèle plus évolué, car il calcule la couleur des trois sommets de la facette en fonction de leurs normales. Ensuite, il effectue une interpolation bilinéaire des trois couleurs pendant la rastérisation. Toutefois, ce modèle gère très mal la lumière spéculaire.
- **Modèle de Phong** : C'est le modèle le plus réaliste, car il calcule la couleur au niveau de chaque pixel en interpolant les normales des trois sommets. Cependant, ce modèle est plus coûteux en terme de ressource système. C'est pour cette raison qu'il n'est pas pris en charge nativement par le matériel graphique.

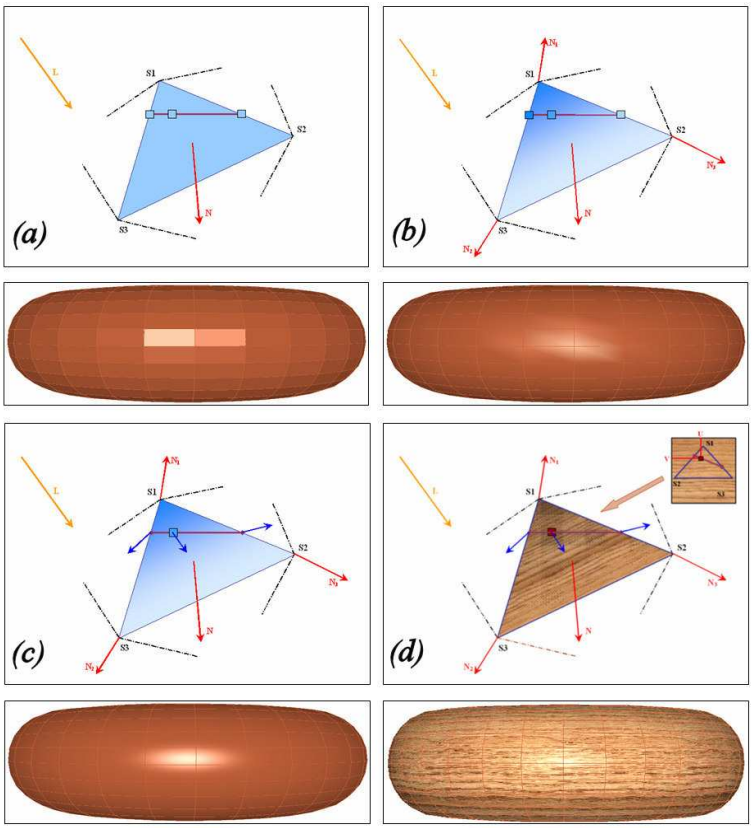


Fig. 1.11 - Modèles d'ombrage usuels. (a) Modèle de Lambert. (b) Modèle de Gouraud. (c) Modèle de Phong. (d) Modèle de Phong avec placage de texture.

1.3.2 Principe du placage de textures

La technique du placage de textures a été introduite par Catmull [Cat74], puis étendue par Blinn et Newell [BN76]. Elle est largement utilisée en imagerie de synthèse, car elle permet d'habiller les modèles 3D.

L'idée principale du placage de texture, est d'associer une image en deux dimensions à une surface en trois dimensions à l'aide d'une fonction appelée *paramétrisation*. Cette fonction met en correspondance chaque sommet (x,y,z) de la surface maillée, avec une paire de coordonnées (u,v) représentant un pixel de la texture. Étant donnée une surface S de R^3, une paramétrisation $\sigma(u,v)$ de S, est une fonction bijective mettant un sous-ensemble M de R^2 en correspondance avec la surface S, comme il est schématisé sur la figure 1.12.

$$\begin{cases} \sigma : & M \subset R^2 & \rightarrow & R^3 \\ & (u,v) & \mapsto & (x,y,z) \end{cases}$$

où les notions suivantes peuvent être définies :

- L'ensemble M de R^2 est appelé *domaine* (u,v) ou encore *domaine paramétrique*, l'espace R^2 associé est appelé *espace texture*.
- Par définition, $\sigma(u,v)$ est bijective, elle a donc une fonction inverse $\sigma^{-1}(x,y,z)$ appelée *paramétrisation inverse*.

Lorsque le maillage de la surface 3D S est créé à partir d'une surface paramétrique (plan, sphère, tore, spline, surface de révolution ou d'extrusion…), les coordonnées de textures (u,v) découlent directement de l'équation paramétrique de S. Dans le cas d'un maillage quelconque, la stratégie la plus utilisée consiste à projeter les coordonnées de texture d'une surface déjà paramétrée, et dont la topologie est proche du maillage à traiter.

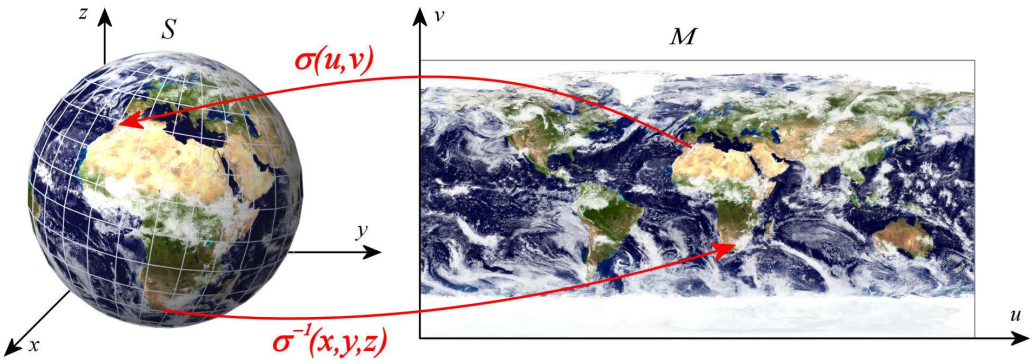


Fig. 1.12 - Paramétrisation $\sigma(u,v)$ et paramétrisation inverse $\sigma^{-1}(x,y,z)$ mettant en correspondance une surface S de R^3 avec le domaine M de R^2

1.3.3 Algorithme de rastérisation

Comme nous l'avons évoqué précédemment, la phase de rastérisation effectue le remplissage des polygones constitués par les sommets projetés. D'une manière plus pratique, l'algorithme du rendu d'un maillage texturé peut être effectuée par les étapes suivantes [HM91] :

1. Associer à chaque sommet du maillage, un certain nombre de paramètres d'intérêt (r_1,r_2,…,r_n). Parmi ces paramètres, on trouve bien évidemment les coordonnées de texture et la profondeur, et suivant la méthode d'ombrage, nous pouvons avoir également la couleur, la normale, la binormale et la tangente.
2. Pour chaque sommet d'une primitive constituant le maillage (généralement un triangle), effectuer la transformation de l'espace objet vers l'espace écran en utilisant des matrices de transformation homogène 4×4. on obtient alors les valeurs (wx,wy,wz,w).
3. Effectuer le fenêtrage de chaque primitive en utilisant les équations des six plans constituant le volume de vue. Les paramètres d'intérêt doivent être linéairement interpolés lorsque de nouveaux sommets sont créés. Nous obtenons ainsi un ensemble de polygones.
4. Pour chaque sommet, diviser les coordonnées écrans (homogènes), les paramètres d'intérêt r_i et le nombre 1 par w, afin de construire la liste (x,y,z,s_1,s_2,…,s_n,s_{n+1}) où $s_i=r_i/w$ pour $i\leq n$ et $s_{n+1}=1/w$.
5. Dans l'espace écran, effectuer le remplissage de chaque polygone obtenu après le fenêtrage en procédant ligne par ligne. Pour chaque pixel, effectuer une interpolation bilinéaire des paramètres d'intérêt. Finalement, utiliser les paramètres $r_i=s_i/s_{n+1}$ dans une formule d'ombrage.

1.4 Principe général du placage de déplacement 3D par pixel

Le placage de déplacement tridimensionnel par pixel [PHL91] est une méthode qui s'inspire à la fois du placage de bosselures [Bli78] qui procède au niveau des pixels, et du placage de déplacement [Coo84] qui effectue un traitement au niveau des sommets. Cette technique permet d'ajouter des détails très fins (mesostructures) aux modèles géométriques à base de maillages sans en augmenter la densité (contrairement au déplacement par sommet). Cela permet d'afficher des modèles très détaillés, tout en évitant le goulot d'étranglement pouvant être causé par un très grand nombre de sommets et de polygones (figure 1.13). Le placage de déplacement par pixel est basé sur trois principes fondamentaux : la carte de déplacement, l'espace tangent, et le tracé de rayons inverse.

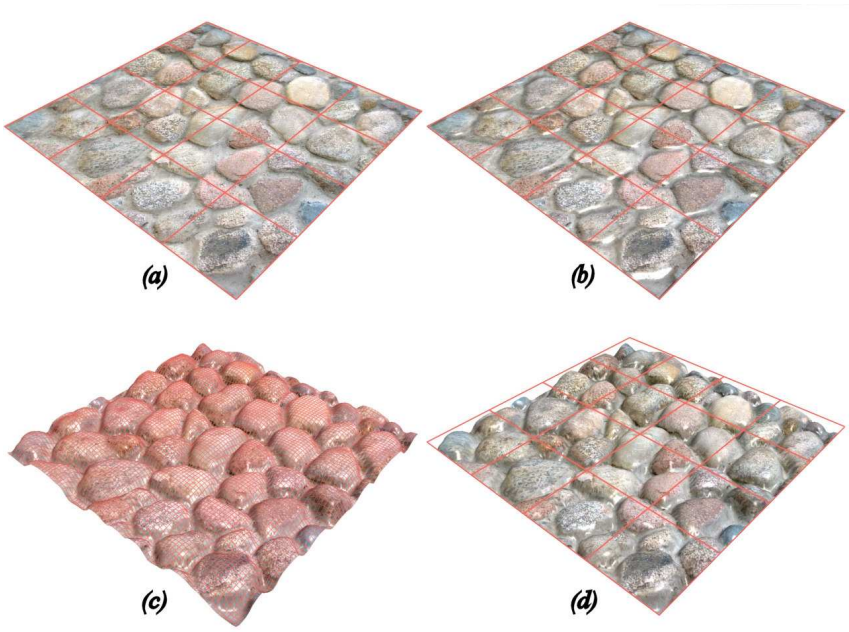


Fig. 1.13 - Comparaison entre les différents niveaux de la représentation des microreliefs. Les lignes rouges mettent en évidence le maillage polygonal. (a) Placage de texture seul. (b) Placage de bosselures. (c) Placage de déplacement par sommet. (d) Placage de déplacement par pixel.

1.4.1 La carte de déplacement

Parfois appelée *carte de relief*, cette carte est une image bidimensionnelle dont les pixels ne servent pas à stocker des couleurs, mais des données géométriques qui sont en l'occurrence, des profondeurs et des normales (figure 1.14). Dans le canal α, sont souvent placées les profondeurs associées au microrelief plaqué sur la surface 3D. Les trois autres canaux : *rouge*, *vert* et *bleu* servent à stocker les trois composantes *x*, *y* et *z* des normales qui sont calculées à partir des profondeurs. Comme la composante *z* de la normale peut être retrouvée en fonction des deux

autres, le canal *bleu* peut être libéré pour pouvoir stocker d'autres données utilisées par certaines techniques comme *le traçage de cônes*. Dans ce cas, la carte de déplacement peut être nommée en fonction de cette technique comme par exemple la *carte de cônes*.

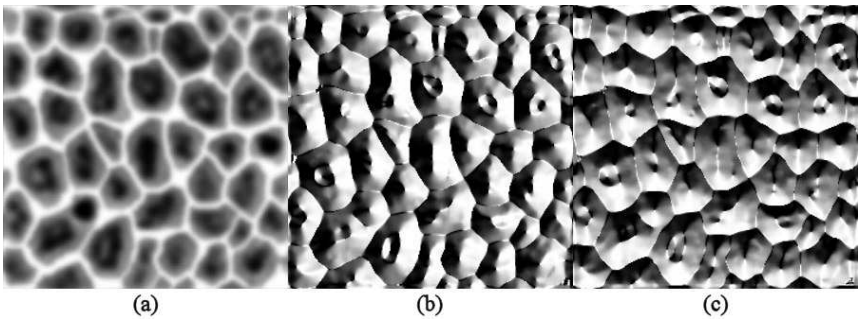


Fig. 1.14 - Carte de déplacement (a) Le canal α qui abrite les profondeurs. (b) le canal *rouge* où l'on stocke la composante *x* de la normale, et (c) le canal *vert* qui stocke la composante *y*.

Les cartes de déplacement peuvent être obtenues de différentes manières, suivant le type de relief recherché. Pour le rendu de terrains ou de surfaces liquides par exemple, le calcul de la carte de profondeurs et des normales s'effectue directement à partir d'une équation paramétrique $H(u,v)$ (de type ondulatoire ou fractal). Dans le cadre de la simplification de maillages pour le rendu temps réel, la carte de profondeurs s'obtient par l'encodage des distances représentant les différences entre la version dense du maillage et sa version simplifiée [CMRS98, LMH00, EBAB05, TI07]. Les profondeurs peuvent être extraites à partir du z-buffer après un rendu orthographique d'une surface avec mesostructure, ou à partir d'un rendu sur texture dans le cas des imposteurs [JWP05, MJW07]. Les cartes de déplacement peuvent également être créées à partir de photographies de mesostructures, en s'appuyant sur l'ombrage induit par le changement de la position de la source de lumière [ZTCS99, LKG+03]. Enfin, puisque les cartes de profondeurs ne sont que des images 2D en niveaux de gris, leur création peut s'effectuer à l'aide d'outils de dessin classiques, en dessinant des formes et en les remplissant d'une manière uniforme ou par un gradient de gris.

1.4.2 L'espace tangent

L'espace tangent [PAC97] ou espace TBN (Tangente, Binormale et Normale), est un espace associé à chaque sommet constituant un objet 3D (voir la figure 1.15). Cet espace est construit en fonction de la normale au sommet, ainsi que des coordonnées de texture qui lui sont associées. L'espace tangent est très utile du fait que les normales stockées dans la carte de déplacement sont naturellement exprimées dans cet espace. De ce fait, il serait très coûteux d'effectuer un changement de repère pour chaque normale. Cependant, les vecteurs de vue et de lumière

doivent être également exprimés dans cet espace. Ils seront par la suite interpolés pour chaque pixel de l'objet pendant la rastérisation.

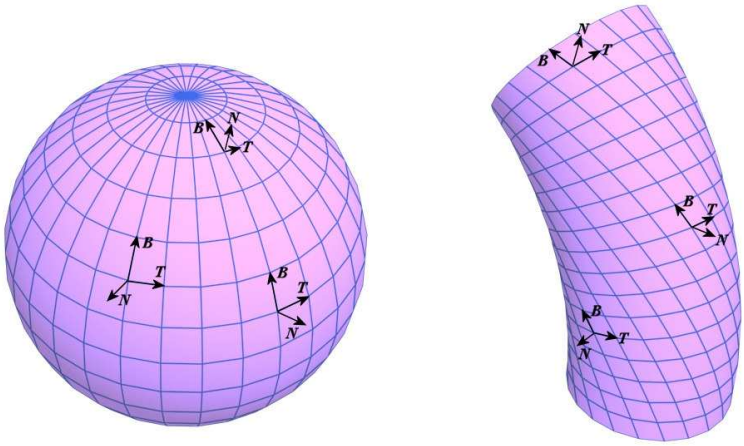


Fig. 1.15 - Espace tangent ou espace TBN. Il s'agit d'un espace local associé à chaque point de la surface.

1.4.3 Le traçage de rayon inverse

La figure 1.16 illustre la problématique principale de la technique du placage de déplacement par pixel. Il s'agit de la recherche de l'intersection de l'axe de vue avec le microrelief stocké dans la carte de déplacement. La recherche de l'intersection doit être effectuée pour chaque pixel issu de l'objet 3D qui est associé à cette carte de déplacement. Cette recherche s'effectue en utilisant un algorithme de traçage de rayons inverse. L'adjectif *inverse* désigne le fait qu'on part d'un pixel déjà projeté sur écran, alors que dans le cas du traçage direct, on part d'une position 2D de l'écran pour essayer de trouver le point de l'objet 3D qui s'y projette.

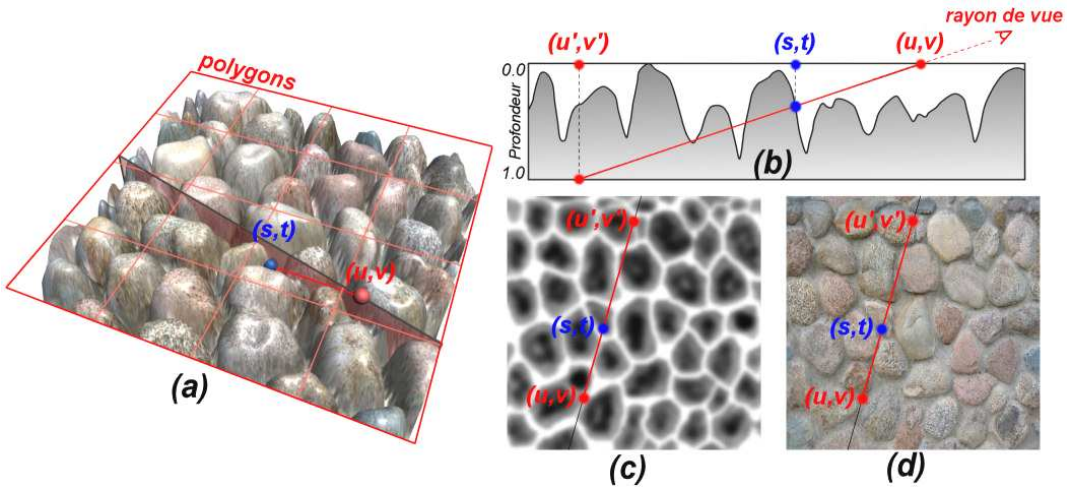


Fig. 1.16 - Calcul de l'intersection avec le relief. (*u*,*v*) est le pixel traité à ce stade. (*s*,*t*) est le pixel rectifié après la recherche d'intersection (a) Vue 3D de l'intersection. (b) La coupe du relief incluant le rayon de vue. (c) La carte de profondeurs. (d) La texture couleur correspondante. Le pixel à la position (u,v) sera texturé avec le texel (s,t) au lieu du texel (u,v).

La contrainte de rapidité ne permet pas une recherche exacte, ce qui oblige à trouver un point se rapprochant le plus possible du point de l'intersection, tout en utilisant un nombre restreint d'étapes. Plusieurs approches existent pour effectuer cette recherche. Szirmay-Kalos et Umenhoffer [SKU08], dans leur article sur l'état de l'art du placage de déplacement en temps réel, les ont classé en quatre catégories : les techniques non itératives, les techniques itératives non fiables, les techniques itératives fiables et les techniques itératives hybrides. La fiabilité dans ce contexte désigne la détection de la première intersection. C'est cette classification que nous avons retenue dans la section traitant des travaux antérieurs. En effet, un ordre chronologique des techniques pourrait prêter à confusion.

1.5 Fondement Mathématique

Le placage de déplacement par pixel repose à la fois sur le placage de bosselures, et sur le placage de déplacement par sommet. Dans cette section, nous proposons une re-formulation globale et cohérente de ces deux théories. Nous allons commencer par présenter la théorie du placage de déplacement, car elle intervient logiquement en premier.

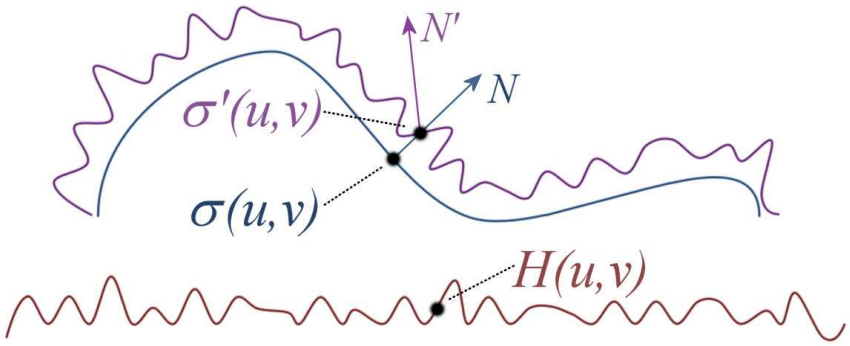


Fig. 1.17 - Principe du déplacement d'une surface paramétrique σ par une fonction de déplacement *H*. Le résultat étant la surface modulée σ'.

Soit σ la surface paramétrique qui va servir de support pour le déplacement 3D (figure 1.17). Cette surface est définie comme suit :

$$\begin{cases} \sigma : R^2 \to R^3 \\ (u,v) \mapsto (x,y,z) \end{cases}$$

u et *v* sont à la fois les coordonnées paramétriques de σ, et les coordonnées de texture de celle-ci.

Soit H la fonction[1] de R^2 qui définit la magnitude du déplacement de la surface σ. Le déplacement s'effectue suivant la direction de la normale à la surface σ. H est définie par :

$$\begin{cases} H : R^2 \to R \\ (u,v) \mapsto H(u,v) \end{cases}$$

La fonction H est usuellement représentée par une carte de hauteurs M, stockée sous forme d'une image à niveau de gris. Comme les textures sont souvent définies sous forme d'entiers positifs à 8 bits, nous devons définir la correspondance suivante entre H et M :

$$H(u,v) = b + e.M(u,v)$$

où b est un paramètre qui sert à biaiser les valeurs de M, car H peut avoir des valeurs négatives. e est un facteur d'échelle permettant de contrôler l'amplitude des déplacements.

[1] Cette fonction sera par la suite notée *D* lorsqu'elle sera utilisée dans un contexte de profondeurs.

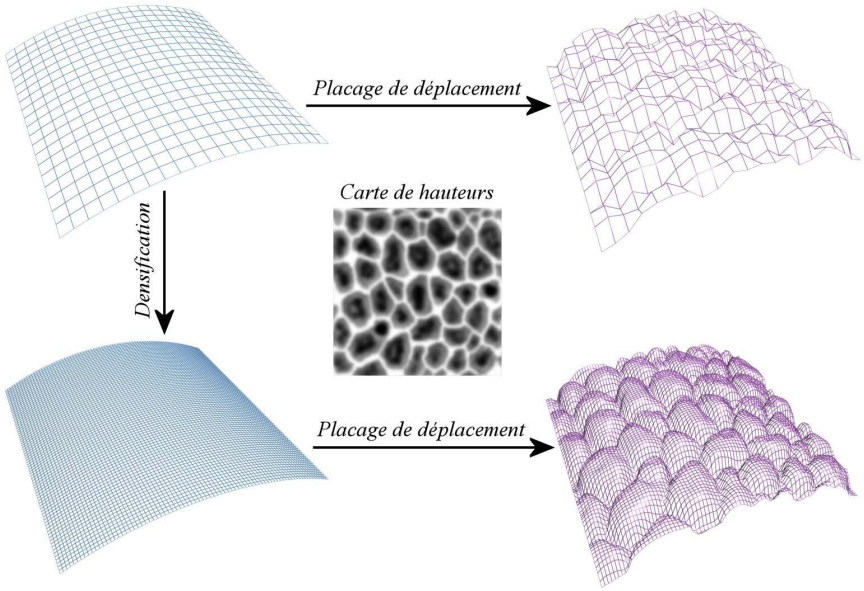


Fig. 1.18 - Placage de déplacement sur une surface maillée. Le déplacement direct ne permet pas d'avoir une surface avec un microrelief correct. Il faut impérativement effectuer une densification du maillage de base, avant de pouvoir procéder au déplacement.

1.5.1 Placage de déplacement

Le placage de déplacement consiste à déplacer chaque point de la surface σ en fonction de la valeur donnée par la fonction H. Ce déplacement est effectué suivant la normale $\vec{N}$ à la surface σ, que l'on retrouve à l'aide du produit vectoriel suivant :

$$\vec{N}(u,v) = \frac{\partial \sigma(u,v)}{\partial u} \wedge \frac{\partial \sigma(u,v)}{\partial v} \tag{1.1}$$

La surface σ' obtenue par déplacement de σ sera alors définie par la formule suivante :

$$\sigma'(u,v) = \sigma(u,v) + H(u,v)\frac{\vec{N}(u,v)}{\left\|\vec{N}(u,v)\right\|} \tag{1.2}$$

Comme le pipeline ne gère pas directement des surfaces paramétriques continues. σ est discrétisée pour correspondre à un maillage (polyèdre). Rappelons qu'un maillage $\boldsymbol{M}$ dans l'espace tridimensionnel R^3 est la donnée de :

- Une suite de points $P_1, P_2,\ldots,P_n$ de R^3 appelés sommets du maillage. avec $P_i = \sigma(u_j, v_k)$
- Un ensemble de polygones, chaque polygone étant une suite de numéros de sommets dans $[1,2,\ldots,n]$. Ces polygones sont généralement triangulaires.

La facettisation est soit régulière, c-à-d que les sommets P_i sont calculés à intervalles réguliers, soit irrégulière pour mieux correspondre à la courbure de la surface. Mais dans tous les cas, la densité du maillage résultant ne peut correspondre aux petites variations de H. Par

conséquent, un déplacement direct des points P_i ne produira pas une surface avec mesostructures. Il faudra alors subdiviser le maillage de base (chaque triangle est divisé en plusieurs sous triangles), avant de procéder aux déplacement des sommets (figure 1.18). Le placage de déplacement adaptatif, décrit dans [GH99, DH00], permet de moduler la densification (ou tesselation) du maillage en fonction des variations du microrelief, permettant ainsi de réduire considérablement le nombre de polygones.

Après avoir procédé au déplacement d'un sommet du maillage, il faudra ensuite définir sa normale. Dans le cas général d'un maillage arbitraire, cette normale est obtenue en effectuant la moyenne des normales aux facettes adjacentes au sommet. Dans le cas du placage de déplacement, la normale peut être obtenue directement par la formule (1.7), issue du placage de bosselures (voir la section suivante). Notons enfin que la raison pour laquelle on parle de placage de déplacement, plutôt que d'extrusion de surface, est que cette technique se base sur une image 2D et sur des coordonnées de texture, à l'instar du placage de texture classique.

1.5.2 Placage de bosselures

A la différence du placage de déplacement, qui modifie la géométrie, le placage de bosselures n'intervient qu'au niveau de l'ombrage. Ce dernier étant fonction de la normale, la perturbation des normales entraînera l'illusion d'un microrelief. Ainsi, au lieu de créer la surface σ', il suffit de calculer sa normale et de l'utiliser dans une formule d'ombrage. Cette normale est donnée par le produit vectoriel suivant :

$$\vec{N}'(u,v) = \frac{\partial \sigma'(u,v)}{\partial u} \wedge \frac{\partial \sigma'(u,v)}{\partial v} \tag{1.3}$$

Nous développons les dérivées partielles de σ' en utilisant son expression (1.2). Avec la règle de dérivation en chaîne, on obtient :

$$\frac{\partial \sigma'}{\partial u} = \frac{\partial \sigma}{\partial u} + \frac{\partial H}{\partial u} \cdot \left(\frac{\vec{N}}{\left\| \vec{N} \right\|} \right) + H \frac{\partial}{\partial u} \left(\frac{\vec{N}}{\left\| \vec{N} \right\|} \right) \tag{1.4}$$

$$\frac{\partial \sigma'}{\partial v} = \frac{\partial \sigma}{\partial v} + \frac{\partial H}{\partial v} \cdot \left(\frac{\vec{N}}{\left\| \vec{N} \right\|} \right) + H \frac{\partial}{\partial v} \left(\frac{\vec{N}}{\left\| \vec{N} \right\|} \right) \tag{1.5}$$

Dans le contexte du placage de bosselures, H représente une mesostructure. Cette dernière peut être considérée comme négligeable par rapport à la macrostructure définie par σ. En

supposant *H* proche de zéro, le dernier terme des équations (1.4) et (1.5) peut alors être négligé. Notons que même si *H* est négligeable, ses dérivées partielles restent significatives.

Le développement du produit vectoriel de l'équation (1.3) donne l'expression suivante pour $\vec{N}'$:

$$\vec{N}' = \left(\frac{\partial \sigma}{\partial u} + \frac{\partial H}{\partial u} \cdot \left(\frac{\vec{N}}{\left\| \vec{N} \right\|} \right) \right) \wedge \left(\frac{\partial \sigma}{\partial v} + \frac{\partial H}{\partial v} \cdot \left(\frac{\vec{N}}{\left\| \vec{N} \right\|} \right) \right)$$

Ce qui vaut à :

$$\vec{N}' = \frac{\partial \sigma}{\partial u} \wedge \frac{\partial \sigma}{\partial v} + \frac{\frac{\partial H}{\partial u} \cdot \left(\vec{N} \wedge \frac{\partial \sigma}{\partial v} \right)}{\left\| \vec{N} \right\|} + \frac{\frac{\partial H}{\partial v} \cdot \left(\frac{\partial \sigma}{\partial u} \wedge \vec{N} \right)}{\left\| \vec{N} \right\|} + \frac{\frac{\partial H}{\partial u} \cdot \frac{\partial H}{\partial v} \cdot \left(\vec{N} \wedge \vec{N} \right)}{\left\| \vec{N} \right\|^2}$$

Le premier terme vaut $\vec{N}$ selon l'équation (1.1). Le dernier terme est nul puisque $\vec{N} \wedge \vec{N}$ vaut 0, $\vec{N}'$ se simplifie alors en :

$$\begin{aligned} \vec{N}' &= \vec{N} + \left(\frac{\partial H}{\partial u} \cdot \left(\vec{N} \wedge \frac{\partial \sigma}{\partial v} \right) - \frac{\partial H}{\partial v} \cdot \left(\vec{N} \wedge \frac{\partial \sigma}{\partial u} \right) \right) \Big/ \left\| \vec{N} \right\| \\ \vec{N}' &= \vec{N} + \vec{D} \end{aligned} \qquad (1.6)$$

C'est cette formule qui est utilisée dans le placage de bosselures classique. Blinn [Bli78] a donné deux interprétations concernant la normale $\vec{N}'$: La première la considère comme un décalage de $\vec{N}$ par le vecteur $\vec{D}$. Quant à la deuxième interprétation, elle considère que $\vec{N}'$ est obtenue par rotation de $\vec{N}$ autour d'un axe définie par le vecteur : $\vec{A} = \vec{N} \wedge \vec{D}$, avec un angle vérifiant : $\tan\theta = \left\| \vec{D} \right\| / \left\| \vec{N} \right\|$.

$\vec{N}'$ peut être calculée à l'aide de la formule (1.6), en utilisant directement la carte de hauteurs *M* qui encode la fonction *H*(*u*,*v*). La carte *M* peut alors être appliquée à n'importe quelle surface. Une autre solution consiste à calculer $\vec{N}'$, puis de la stocker directement dans une carte de normales. $\vec{N}'$ sera exprimée dans ce cas dans l'espace objet. Il faudra alors transformer les vecteurs *lumière* et *vue* dans cet espace, puis les interpoler pendant la rastérisation. Toutefois, cette représentation ne sera applicable qu'à la surface *σ*. Par la suite, nous allons voir comment supprimer cette dépendance, en utilisant un espace plus approprié, et en posant certaines conditions concernant la paramétrisation de surface *σ*.

Chaque pixel projeté de *σ* sera ombré en utilisant la normale $\vec{N}'$ et non $\vec{N}$. Cependant, puisque le pipeline graphique travaille sur la version maillée de *σ*, il n'est pas possible de déterminer directement les dérivées partielles de celle-ci. Il faudra alors les calculer pour chaque sommet au

moment de la facettisation. Elles seront par la suite interpolées et transmises à chaque pixel issu de la surface. Dans le cas d'un maillage arbitraire, nous pouvons nous baser sur l'interpolation linéaire[1].

Pour calculer les dérivées partielles de la fonction *H*, qui est stockée sous forme d'une image à niveau de gris, nous utilisons les différences finies suivantes :

$$\begin{cases} \dfrac{\partial H(u,v)}{\partial u} \approx \dfrac{H(u+\varepsilon,v)-H(u-\varepsilon,v)}{2\varepsilon} \\ \dfrac{\partial H(u,v)}{\partial v} \approx \dfrac{H(u,v+\varepsilon)-H(u,v-\varepsilon)}{2\varepsilon} \end{cases}$$

pour ε assez petit (un pixel dans ce cas). Nous pouvons également utiliser la convolution par un filtre gradient.

La formule (1.6) reste cependant très complexe, d'autant plus qu'il faudra l'évaluer pour chaque pixel. Il nous faudra alors trouver une solution pour éliminer le produit vectoriel. Ce qui reviendrait à trouver un espace où le calcul sera plus simplifié.

Notons d'abord les dérivées partielles de σ par :

$$\frac{\partial \sigma}{\partial u} = \vec{T} \quad et \quad \frac{\partial \sigma}{\partial v} = \vec{B}$$

$\vec{T}$ et $\vec{B}$ sont des vecteurs tangents à la surface σ, ils sont appelées respectivement *tangente* et *binormale*. Avec $\vec{N}$, ils constituent l'espace tangent ou espace TBN. Cet espace est localisé en chaque point (*u*,*v*) de σ (figure 1.15).

Réécrivons l'équation (1.6), sachant que $\vec{N} = \vec{T} \wedge \vec{B}$:

$$\vec{N}' = \frac{\dfrac{\partial H}{\partial u} \cdot \left(\left(\vec{T} \wedge \vec{B} \right) \wedge \vec{B} \right) - \dfrac{\partial H}{\partial v} \cdot \left(\left(\vec{T} \wedge \vec{B} \right) \wedge \vec{T} \right)}{\left\| \vec{N} \right\|} + \vec{N}$$

En utilisant l'identité de Lagrange suivante :

$$\left(\vec{A} \wedge \vec{B} \right) \wedge \vec{C} = \left(\vec{A}\vec{C} \right)\vec{B} - \left(\vec{B}.\vec{C} \right)\vec{A}$$

Puis en regroupant $\vec{T}$ et $\vec{B}$, nous obtenons :

[1] Nous allons évoquer cette solution dans la section 1.5.3

$$\vec{N}' = \left(\frac{\left(\vec{B}\cdot\vec{T}\right)\frac{\partial H}{\partial v} - \left(\vec{B}\cdot\vec{B}\right)\frac{\partial H}{\partial u}}{\left\|\vec{N}\right\|} \right) \vec{T} + \left(\frac{\left(\vec{T}\cdot\vec{B}\right)\frac{\partial H}{\partial v} - \left(\vec{T}\cdot\vec{T}\right)\frac{\partial H}{\partial u}}{\left\|\vec{N}\right\|} \right) \vec{B} + \vec{N} \qquad (1.7)$$

La formule (1.7) est générale, et elle est valable quelque soit la nature de la surface σ. Cependant, pour un bon nombre de surfaces, la paramétrisation est effectuée de telle sorte que les dérivées partielles $\vec{T}$ et $\vec{B}$ soient orthogonales. C'est le cas par exemple si σ est un plan, une sphère, un cylindre, un tore, une surface de révolution, ou encore une extrusion orthogonale.

En assument que $\vec{T}\cdot\vec{B} = 0$, on aura alors $\left\|\vec{N}\right\| = \left\|\vec{T}\right\| \cdot \left\|\vec{B}\right\|$, car $\vec{N} = \vec{T} \wedge \vec{B}$. L'équation (1.7) devient :

$$\vec{N}' = \left(-\frac{\left\|\vec{B}\right\|}{\left\|\vec{T}\right\|}\frac{\partial H}{\partial u} \right) \vec{T} + \left(-\frac{\left\|\vec{T}\right\|}{\left\|\vec{B}\right\|}\frac{\partial H}{\partial v} \right) \vec{B} + \vec{N}$$

Nous exprimons $\vec{N}'$ dans un repère orthonormé, sachant que :

$$\vec{T} = \left\|\vec{T}\right\| \cdot \vec{T}_n, \quad \vec{B} = \left\|\vec{B}\right\| \cdot \vec{B}_n \quad et \quad \vec{N} = \left\|\vec{N}\right\| \cdot \vec{N}_n$$

En factorisant tous les termes par $\left\|\vec{N}\right\|$, nous obtenons *:*

$$\vec{N}' = \left\|\vec{N}\right\| \cdot \left(-\frac{1}{\left\|\vec{T}\right\|}\frac{\partial H}{\partial u} \cdot \vec{T}_n - \frac{1}{\left\|\vec{B}\right\|}\frac{\partial H}{\partial v} \cdot \vec{B}_n + \vec{N}_n \right)$$

Nous pouvons ignorer le terme $\left\|\vec{N}\right\|$, car de toute façon $\vec{N}'$ n'est pas normée. L'expression de $\vec{N}'$ dans l'espace tangent orthonormé sera alors la suivante :

$$\vec{N}' = \left(-\frac{1}{\left\|\vec{T}\right\|}\frac{\partial H}{\partial u}, \quad -\frac{1}{\left\|\vec{B}\right\|}\frac{\partial H}{\partial v}, \quad 1 \right)_{Tangent}$$

Cependant, avec cette formule, l'échelle de profondeur reste fixe. Elle est égale à l'unité dans l'espace tangent. Afin de pouvoir moduler cette échelle, qui devrait logiquement être très petite par rapport à la surface σ, nous introduisons un facteur d'échelle *a* que nous pouvons contrôler à volonté. Ainsi, si nous remplaçons *H* par *aH* dans la formule précédente, nous obtenons la formule suivante pour $\vec{N}'$:

$$\vec{N}' = \left(-\frac{a}{\left\|\vec{T}\right\|}\frac{\partial H}{\partial u}, \quad -\frac{a}{\left\|\vec{B}\right\|}\frac{\partial H}{\partial v}, \quad 1 \right)_{Tangent} \qquad (1.8)$$

Nous remarquons que l'expression de $\vec{N}'$ est beaucoup plus simple. Nous pouvons encore éliminer la dépendance de $\vec{N}'$ vis-à-vis de *H*, si nous considérons que la paramétrisation de σ est régulière (patchs carrés). Nous aurons alors: $\left\|\vec{T}\right\| = \left\|\vec{B}\right\| = cte$. En intégrant cette constante dans le facteur d'échelle *a*, l'expression de $\vec{N}'$ devient finalement :

$$\vec{N}' = \left(-a\frac{\partial H}{\partial u}, \quad -a\frac{\partial H}{\partial v}, \quad 1 \right)_{Tangent} \tag{1.9}$$

La normale $\vec{N}'$ doit être normée avant d'être utilisée dans une formule d'ombrage.

Dans la section 1.6, au lieu de conserver la normale $\vec{N}'$ (dépendante de σ dans (1.8) ou indépendante de σ dans (1.9)), nous suggérons de conserver directement les dérivées partielles de $H(u,v)$ dans une carte de dérivées. Puis, de calculer $\vec{N}'$ pendant la rastérisation à l'aide de l'une des formules (1.8) et (1.9). Nous pouvons ainsi moduler l'échelle *a* en temps réel, et par la même occasion, supprimer la dépendance vis-à-vis de σ, même dans le cas où la paramétrisation n'est pas régulière.

Le calcul dans l'espace tangent nous permettra d'éviter d'évaluer une expression lourde pour $\vec{N}'$. Spécialement parce que cette évaluation doit s'effectuer pour chaque pixel traité. Evidemment, les autres vecteurs intervenant dans l'ombrage (les vecteurs *lumière* et *vue*) doivent également être exprimés dans l'espace tangent. Néanmoins, cette transformation ne sera effectuée qu'une seule fois par sommet. Par la suite, ces vecteurs seront interpolés pour chaque pixel pendant la phase de rastérisation.

Soit $\vec{V} = (x,y,z)$ un vecteur exprimé dans l'espace local. Nous souhaitons calculer les coordonnées (u,v,w) du vecteur $\vec{V}$ dans l'espace tangent. Nous avons :

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = (TBN)^{-1} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad avec \quad TBN = \begin{pmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{pmatrix}$$

Quant au passage du repère tangent au repère local, il se fait à l'aide de la formule suivante :

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = TBN \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

Notons que dans le cas où l'espace tangent est orthonormé, nous aurons:

$$(TBN)^{-1} = (TBN)^T$$

Les vecteurs *lumière* et *vue* sont généralement définis dans l'espace global de la scène, il faudra alors les exprimer dans l'espace local avant de passer à l'espace tangent.

1.5.3 Calcul de T, B et N

Les trois composantes de l'espace tangent sont calculées directement à partir de la surface paramétrique σ, lorsque celle-ci est disponible. Nous avons alors les relations suivantes :

$$\vec{T} = \frac{\partial \sigma}{\partial u}, \qquad \vec{B} = \frac{\partial \sigma}{\partial v} \qquad et \qquad \vec{N} = \vec{T} \wedge \vec{B}$$

Dans le cas d'un maillage quelconque [Gat03], il nous faudra passer par les coordonnées géométriques (x,y,z) et les coordonnées de texture (u,v) des sommets constituant le maillage. Prenons un triangle constitué de trois sommets $P_1=(x_1,y_1,z_1)$, $P_2=(x_2,y_2,z_2)$ et $P_3=(x_3,y_3,z_3)$, ayant respectivement comme coordonnées de texture : (u_1,v_1), (u_2,v_2) et (u_3,v_3). Puis, exprimons les vecteurs $\overrightarrow{P_1P_2}$ et $\overrightarrow{P_1P_3}$ en fonction de $\vec{T}$ et $\vec{B}$. Leur composante suivant $\vec{N}$ étant nulle, car ils se trouvent dans le plan tangent, nous aurons alors les expressions suivantes :

$$\overrightarrow{P_1P_2} = (P_2 - P_1) = (u_2 - u_1)\cdot\vec{T} + (v_2 - v_1)\cdot\vec{B}$$
$$\overrightarrow{P_1P_3} = (P_3 - P_1) = (u_3 - u_1)\cdot\vec{T} + (v_3 - v_1)\cdot\vec{B}$$

La résolution de ce système linéaire nous conduira à l'expression de $\vec{B}$ et $\vec{T}$. $\vec{N}$ sera ensuite déduite à partir du produit vectoriel :

$$\vec{T} = \frac{(v_3 - v_1)(P_2 - P_1) - (v_2 - v_1)(P_3 - P_1)}{(u_2 - u_1)(v_3 - v_1) - (v_2 - v_1)(u_3 - u_1)}$$

$$\vec{B} = \frac{-(u_3 - u_1)(P_2 - P_1) + (u_2 - u_1)(P_3 - P_1)}{(u_2 - u_1)(v_3 - v_1) - (v_2 - v_1)(u_3 - u_1)}$$

$$\vec{N} = \vec{T} \wedge \vec{B}$$

Les vecteurs $\vec{T}$, $\vec{B}$ et $\vec{N}$ seront calculés pour chaque sommet. Cependant, chaque sommet peut être connecté à plusieurs triangles du maillage. Dans ce cas, il faudra effectuer une moyenne de tous les vecteurs calculés à partir de chaque triangle connecté au sommet.

1.6 Synchronisation temps réel entre la modulation du microrelief et son ombrage

Les techniques de placage de déplacement par pixel utilisent le même procédé que le placage de normales décrit dans [PAC97], et qui consiste à stocker les normales calculées à partir de la carte des profondeurs. En effet, toutes ces techniques calculent la normale en fonction des dérivées partielles du microrelief et d'une constante de profondeur, puis conservent les trois composantes de la normale dans les canaux *rouge*, *vert* et *bleu* de la carte de déplacement.

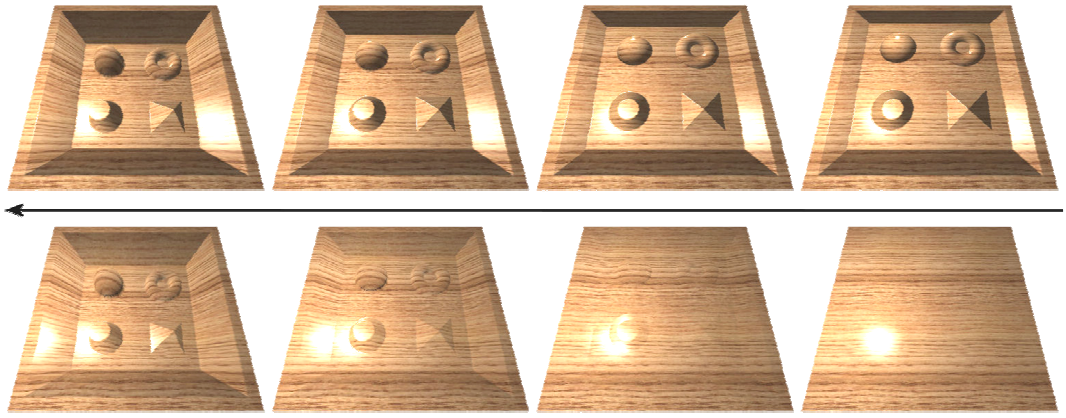


Fig. 1.19 - Synchronisation de l'ombrage avec une séquence de quatre échelles de profondeur (respectivement de gauche à droite : 20%, 10%, 5% et 0%). (En haut) Les techniques de placage de déplacement par pixel traditionnelles. (En bas) Notre méthode de synchronisation.

Cependant, cette méthode n'est adaptée que pour une échelle de profondeur fixée à l'avance. En effet, si l'on désire modifier la profondeur du microrelief en temps réel, nous allons constater que l'ombrage reste constant et ne varie pas en fonction de l'échelle de profondeur. Nous avons proposé deux solutions à cette limitation, la première consiste à recalculer la normale en fonction de la normale déjà stockée dans une carte de déplacement. La deuxième méthode consiste à stocker les dérivées partielles de la carte de profondeurs au lieu de ses normales, puis de calculer celles-ci pendant la phase de rendu en fonction de l'échelle de profondeur souhaitée (voir la figure 1.19).

1.6.1 Mise à jour de la normale en fonction de l'échelle de profondeur

Pour calculer la normale correspondante à un texel[1] donné de la carte de profondeurs, nous commençons par calculer la tangente et la binormale en fonction de la différence de profondeur entre les texels adjacents, suivant les directions *u* et *v*. La tangente et la binormale évoquées dans

[1] Le mot *texel* (*texture element*) désigne un pixel de la texture.

ce contexte ne désignent pas les composantes de l'espace tangent TBN, qui eux, sont liées à un maillage 3D. En notant par $D(i,j)$ la profondeur à la position (i,j) de la carte de profondeurs, nous pouvons exprimer la tangente $\vec{T}_a(i,j)$ et la binormale $\vec{B}_a(i,j)$ par :

$$\begin{aligned}\vec{T}_a(i,j) &= (\,1,\,0,\,aD(i+1,j)-aD(i-1,j)\,)\\ \vec{B}_a(i,j) &= (\,0,\,1,\,aD(i,j+1)-aD(i,j-1)\,)\end{aligned} \tag{1.10}$$

où la constante a désigne un facteur d'échelle permettant de moduler l'échelle de profondeur du relief. La division par 2 des dérivées partielles est incluse dans a.

La normale au point (i,j) est calculée à partir du produit vectoriel de $\vec{T}_a(i,j)$ et $\vec{B}_a(i,j)$:

$$\vec{N}_a(i,j)=\frac{\vec{T}_a(i,j)\wedge\vec{B}_a(i,j)}{\left\|\vec{T}_a(i,j)\wedge\vec{B}_a(i,j)\right\|} \tag{1.11}$$

En notant par T_z et B_z la coordonnées suivant z de la tangente et de la binormale (sans prise en compte de l'échelle c-à-d a=1), nous obtenons à partir des équations (1.10) et (1.11) la formule suivante :

$$\vec{N}_a=\frac{(-aT_z,-aB_z,1)}{\sqrt{a^2{T_z}^2+a^2{B_z}^2+1}} \tag{1.12}$$

Nous retrouvons finalement la même formule que celle en (1.9), mais en partant uniquement de la carte des profondeurs. Cela est dû au fait que, pour arriver à l'équation (1.9), nous avons exprimé la normale dans l'espace TBN, et que nous avons également assumé que les dérivées partielles de la surface paramétrique σ étaient orthogonales et de magnitudes équivalentes.

Les trois composantes de $\vec{N}_a$ sont par la suite stockées dans les canaux *rouge*, *vert* et *bleu* de la carte de déplacement. Toutefois, comme le format le plus utilisé pour les textures est le format entier positif à 8 bits (valeurs de 0 à 255), les composantes doivent être stockées comme suit :

$$(rouge,vert,bleu)=255\cdot\frac{\vec{N}_a+(1,1,1)}{2}$$

Nous utilisons la formule inverse pour récupérer les normales pendant la phase du rendu.

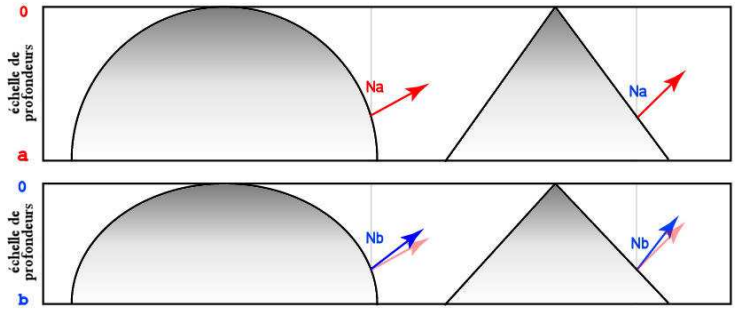


Fig. 1.20 - Le changement de l'échelle de profondeur de *a* vers *b* entraîne la modification des normales.

Pour pouvoir synchroniser l'ombrage et la mise à l'échelle de la profondeur, nous devons mettre à jour la normale en fonction de la nouvelle échelle et de la normale récupérée à partir de la carte de déplacement. Soit b la nouvelle échelle de profondeur, et $\vec{N}_b$ la normale correspondante (figure 1.20). L'expression de cette normale est donnée par :

$$\vec{N}_b = \frac{(-bT_z, -bB_z, 1)}{\sqrt{b^2 {T_z}^2 + b^2 {B_z}^2 + 1}} \tag{1.13}$$

Nous calculons T_z et B_z à partir de l'équation (1.12) :

$$T_z = -\frac{N_{ax}}{aN_{az}} \quad et \quad B_z = -\frac{N_{ay}}{aN_{az}}$$

Si nous remplaçons T_z et B_z dans l'équation (1.13), nous obtiendrons la formule suivante :

$$\vec{N}_b(i,j) = \frac{\left(\frac{b}{a}N_{ax}, \frac{b}{a}N_{ay}, N_{az}\right)}{\sqrt{\left(\frac{b}{a}\right)^2 {N_{ax}}^2 + \left(\frac{b}{a}\right)^2 {N_{ay}}^2 + {N_{az}}^2}}$$

Cette formule est équivalente à :

$$\vec{N}_b(i,j) = \frac{\left(\frac{b}{a}N_{ax}, \frac{b}{a}N_{ay}, N_{az}\right)}{\left\|\left(\frac{b}{a}N_{ax}, \frac{b}{a}N_{ay}, N_{az}\right)\right\|} \tag{1.14}$$

L'expression (1.14) démontre qu'il suffit de multiplier les composantes *x* et *y* de la normale récupérée à partir de la carte de déplacement par la constante *b/a* (ou simplement *b*, si *a* vaut 1 au départ), puis de re-normaliser le vecteur ainsi obtenu.

1.6.2 Stockage des dérivées partielles

Le stockage direct des normales dans la carte de déplacement permet d'éviter une normalisation pendant la phase de rendu. Or, cette optimisation n'est plus justifiée pour les quatre raisons suivantes :

- Certaines techniques ne disposent que de deux canaux libres dans la carte de déplacement, les deux autres étant occupés respectivement par les profondeurs et par une valeur permettant l'encodage de l'espace vide. La troisième coordonnées doit alors être retrouvée pendant le rendu à l'aide de la formule $z = \sqrt{1-(x^2+y^2)}$. Cependant, cette formule est elle-même équivalente à une normalisation.
- La carte de déplacement étant généralement sous format entier 8 bits, les normales récupérées ne sont souvent pas normées correctement, cela peut induire certaines erreurs pendant le rendu.
- Le stockage des normales à échelle constante ne permet pas la modulation de la profondeur. Or, cette fonctionnalité est parfois très utile, comme dans le cas où une même carte de déplacement doit être utilisée avec des échelles différentes, ou lorsque la profondeur doit être animée et constamment mise à jour.
- La normalisation d'un vecteur n'est plus une opération très coûteuse pour les cartes graphiques actuelles. Elle est même directement câblée sur le processeur graphique.

Partant de ces constats, nous avons proposé de stocker directement les dérivées partielles T_z et B_z, au lieu des trois composantes de la normale. Cette dernière sera calculée pendant la phase de rendu en fonction de l'échelle de profondeur souhaitée à l'aide de la formule (1.13).

Chapitre 2

Travaux Antérieurs

Plusieurs stratégies ont été développées pour calculer, en un minimum de temps, le premier point d'intersection entre le rayon de vue et le microrelief généré à partir de la carte de profondeurs. Certaines de ces stratégies sont très simples, et se contentent d'une solution très approximative, tandis que d'autres sont plus complexes, et s'appuient sur le codage de l'espace vide englobant le microrelief. Dans ce chapitre consacré à l'état de l'art, nous allons évoquer la majorité des solutions proposées pour le calcul de l'intersection. Nous les avons classées en quatre catégories : non itératives, itératives non fiables, itératives fiables, et itératives hybrides. Nous discuterons également des techniques permettant la prise en charge de l'auto-ombrage et de la silhouette. Nous présenterons ensuite les techniques de modélisation et rendu à base d'images, et qui permettent la représentation de modèles 3D entiers sans maillage.

2.1 Techniques non itératives

Comme nous l'avons schématisé sur la figure 1.16, l'objectif étant de trouver la couleur du pixel courant ayant les coordonnées de texture (*u*,*v*). Cette couleur se calcule à l'aide d'une formule d'ombrage faisant appel à la texture et à une normale. Ces deux informations sont tirées de la texture et de la carte de déplacement aux coordonnées (*s*,*t*). Les méthodes non itératives n'effectuent pas une recherche du point (*s*,*t*), mais essayent seulement de donner une solution approximative. Le cas extrême est celui de la méthode initiale, qui se contente de confondre les points (*u*,*v*) et (*s*,*t*). Cette famille de techniques est caractérisée par sa simplicité, et surtout par sa rapidité, car elle n'accède à la carte de déplacement qu'une seule fois par pixel traité.

2.1.1 Placage de bosselures

Le placage de bosselures a été la première méthode pour la simulation de microreliefs. Elle a été introduite par Blinn [Bli78]. Son principe est assez simple. Il consiste à déplacer les normales à une surface pour induire des variations d'ombrage, donnant ainsi l'illusion d'un relief sans modification de la géométrie de base. Plus concrètement, il s'agit de déplacer la normale interpolée pendant la rastérisation en fonction des dérivées partielles de la carte des hauteurs représentant le microrelief. L'avantage du placage de bosselures réside dans le fait qu'il produit un rendu très rapide et assez convaincant pour de faibles reliefs (voir la figure 2.1). Cependant, cette technique est limitée au calcul de l'ombrage, et ne gère ni l'auto-ombrage, ni la silhouette.

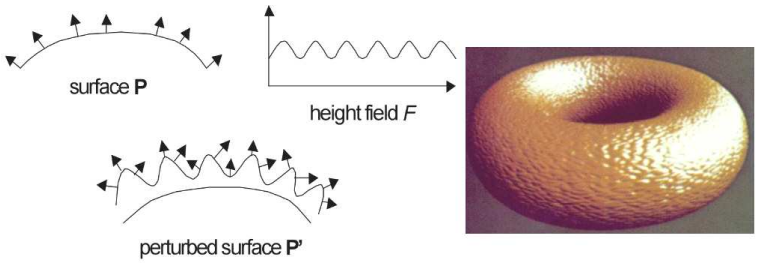


Fig. 2.1 - Technique du placage de bosselures [Bli78]

Au lieu de calculer l'ombrage dans l'espace local ou global, Peercy [PAC97] a proposé d'effectuer les calculs directement dans l'espace tangent. Il a également proposé de stocker les normales du microrelief dans une carte de normales, et ainsi éviter le calcul de ses dérivées partielles pendant la phase du rendu. D'où le terme de *placage de normales*, qui désigne parfois cette technique. Plusieurs autres implémentations, adaptées au rendu temps réel, ont été proposées dans [Kil00, ERSW98, KLP01, LPLY07].

2.1.2 Placage de parallaxe

Le placage de parallaxe [KTI+01], une extension du placage de bosselures, permet la prise en compte de l'effet parallaxe (distorsion de la texture). Cette technique effectue une recherche approximative de l'intersection entre la direction de vue et le relief contenue dans la carte du déplacement. Ce point est défini par l'intersection de la direction de vue et de la droite horizontale qui passe par la hauteur[1] du relief au point courant. L'avantage principal de cette technique est l'ajout de l'effet parallaxe. Cependant, son calcul approximatif, qui suppose que la hauteur est uniforme autour du point (u,v), limite son utilisation à un microrelief irrégulier.

En s'appuyant sur la figure 2.2, nous avons l'égalité suivante :

$$\frac{(u-s,v-t)}{H(u,v)} = \frac{(V_x,V_y)}{V_z}$$

Nous déduisons facilement la position (s,t)[2] :

$$(s,t) = (u,v) - H(u,v)\cdot\left(\frac{V_x}{V_z},\frac{V_y}{V_z}\right) \tag{2.1}$$

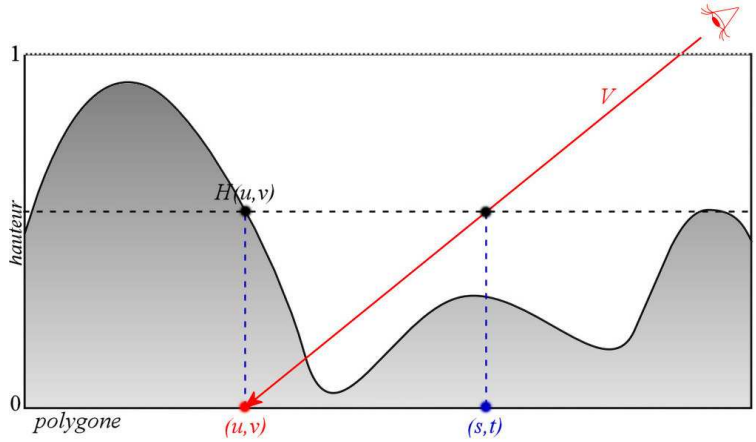


Fig. 2.2 - Principe de la technique du placage de parallaxe.

[1] Le placage de parallaxe et les techniques qui en découlent utilisent traditionnellement le concept de hauteurs plutôt que celui de profondeurs.

[2] Dans l'article original, l'expression de (s,t) est sous forme d'addition, car V est défini dans le sens opposé.

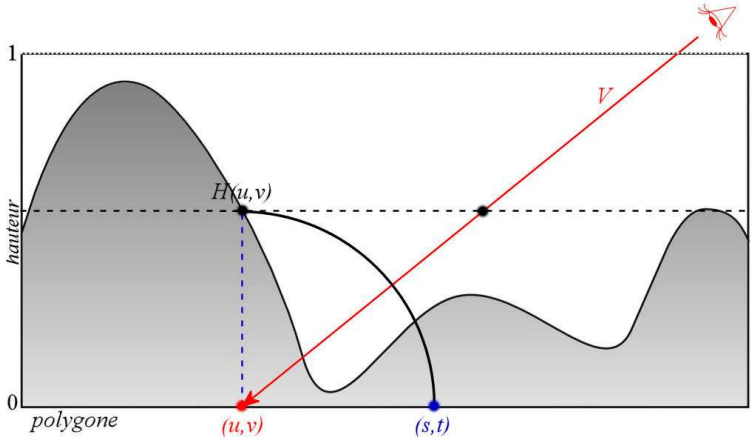


Fig. 2.3 - Technique du placage de parallaxe avec limitation du décalage.

2.1.3 Placage de parallaxe avec limitation du décalage

En plus de son calcul approximatif, le placage de parallaxe tend à être très aléatoire lorsque l'axe de vue devient quasi-parallèle à la surface du polygone. En effet, du fait de la division par V_z dans la formule (2.1), (s,t) tend vers l'infini lorsque V_z s'approche de 0. Le placage de parallaxe avec limitation de décalage [Wel04] permet de résoudre ce problème, en fixant le décalage maximum à une valeur égale à $H(u,v)$ (figure 2.3). Cela est possible en supprimant la division par V_z. $\vec{V}$ étant normé, nous pouvons vérifier dans la formule (2.2) que le décalage ne dépassera pas $H(u,v)$, car $\sqrt{V_x{}^2 + V_y{}^2} \leq 1$.

$$(s,t) \approx (u,v) - H(u,v) \cdot (V_x, V_y) \tag{2.2}$$

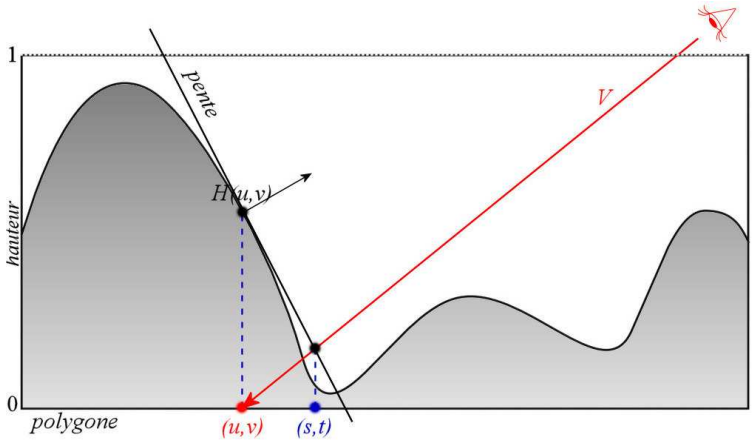


Fig. 2.4 - Technique du placage de parallaxe avec prise en compte de la pente.

2.1.4 Placage de parallaxe avec prise en compte de la pente

Le placage de parallaxe peut être encore amélioré, et sans coût supplémentaire [Pre06]. Cette technique présume que le relief est toujours planaire autour de (*u*,*v*), mais au lieu qu'il soit considéré comme parallèle à la surface du polygone, il est supposé suivre la pente au point (*u*,*v*,*H*(*u*,*v*)) (voir la figure 2.4). En substituant l'équation paramétrique de l'axe de vue : $(u,v,0)-t\vec{V}$ dans l'équation du plan de la pente : $\vec{N}\cdot((x,y,z)-(u,v,H(u,v)))=0$, on trouve :

$$\vec{N}\cdot\left((u,v,0)-t\vec{V}-(u,v,H(u,v))\right)=0$$

En solvant cette équation pour *t*, et en le remplaçant dans l'équation paramétrique de l'axe de vue, nous déduisons le point d'intersection (*s*,*t*) :

$$(s,t)=(u,v)+\frac{H(u,v)\cdot N_z}{\vec{N}\cdot\vec{V}}\left(V_x,V_y\right)$$

Pour éviter des décalages trop importants, nous éliminons le produit scalaire $\vec{N}\cdot\vec{V}$ qui peut être nul. Finalement, nous obtenons :

$$(s,t)\approx(u,v)+H(u,v)\cdot\vec{N}_z\cdot\left(V_x,V_y\right) \tag{2.3}$$

2.2 Techniques itératives non fiables

A la différence des techniques non itératives qui effectuent une recherche sommaire de l'intersection avec le microrelief, les techniques itératives accèdent à la carte de déplacement plusieurs fois, pour essayer de se rapprocher le plus possible d'une intersection. Cependant, cette intersection ne sera pas forcément la première. C'est pour cette raison qu'on parle de techniques non fiables. Ces techniques sont d'ailleurs souvent utilisées en complément d'autres méthodes qui ne risquent pas de sauter la première intersection.

2.2.1 Placage de parallaxe itératif

Cette technique, présentée dans [Pre06], découle directement du placage de parallaxe avec prise en compte de la pente. Il s'agit de répéter la formule (2.3) plus d'une fois (trois ou quatre), afin de se rapprocher encore plus d'un point d'intersection. Cette approche permet d'aboutir à un résultat plus correct, et avec un coût de calcul qui reste raisonnable.

Après l'évaluation de la première intersection (u_i,v_i,w_i), qui appartient à l'axe de vue, nous pouvons écrire :

$$\vec{N}.\left(\left((u_i,v_i,w_i)+t\vec{V}\right)-(u_i,v_i,H_i)\right)=0$$

où w_i désigne la coordonnée suivant le normale du point (u_i,v_i,w_i), $H_i = H(u_i, v_i)$ étant la hauteur du relief au point (u_i,v_i).

En résolvant cette équation pour la dernière intersection calculée, et en ignorant la division par $\vec{N} \cdot \vec{V}$, nous obtenons finalement :

$$(u_{i+1}, v_{i+1}, w_{i+1}) \approx (u_i, v_i, w_i) + (H_i - w_i) \cdot N_z \cdot \vec{V}$$

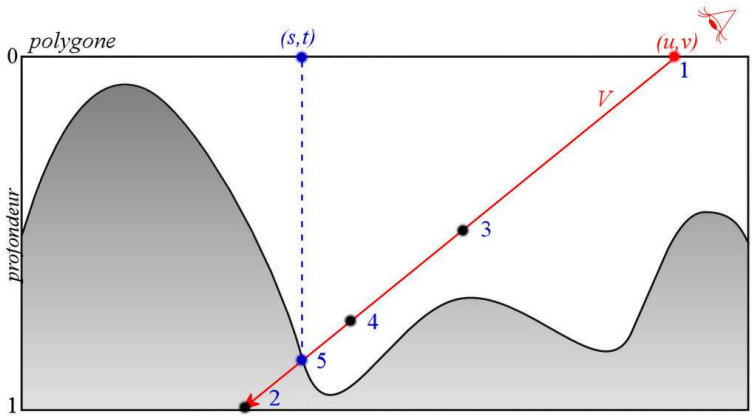


Fig. 2.5 - Recherche binaire de l'intersection.

2.2.2 Recherche binaire

La recherche binaire, ou recherche dichotomique, permet de converger très rapidement vers un point d'intersection [POC05]. L'idée est de tester si la position actuelle du rayon (u_i,v_i,D_i) se trouve sous le relief. Dans ce cas, la progression se fait d'un pas vers le haut. Sinon, la progression sera effectuée vers le bas. Le pas est divisé par deux après chaque étape. Le premier pas est calculé en divisant par deux la distance séparant le premier point (u,v,0) du point où le rayon de vue quitte la base du relief (u_f,v_f,1).

$\vec{V}$ étant normé, et d'après la figure 2.5, le premier pas vaut :

$$pas_0 = \frac{1}{2} \cdot \frac{1}{V_z}$$

La prochaine position sera calculée par la formule suivante :

$$\begin{cases} (u_{i+1}, v_{i+1}, w_{i+1}) = (u_i, v_i, w_i) + pas_i \cdot \vec{V} & Si \quad w_i < D_i \\ (u_{i+1}, v_{i+1}, w_{i+1}) = (u_i, v_i, w_i) - pas_i \cdot \vec{V} & Sinon \\ pas_{i+1} = \frac{1}{2} \cdot pas_i \end{cases}$$

où w_i est la coordonnée suivant la profondeur de la position courante, et $D_i = D(u_i, v_i)$ désigne la profondeur donnée par la carte de déplacement au point (u_i,v_i).

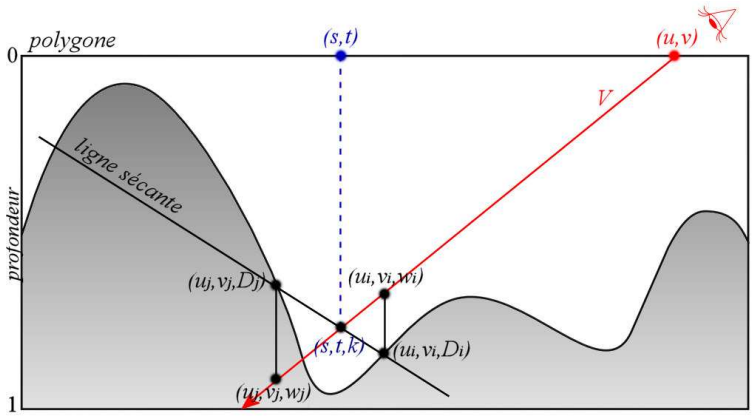


Fig. 2.6 - Recherche sécante de l'intersection.

2.2.3 Recherche sécante

La recherche binaire ne prend pas en compte le microrelief lors de la recherche. La recherche sécante utilisée dans [YJ04, BT04, RSP06] permet de converger encore plus rapidement en utilisant les profondeurs du relief pendant la recherche. L'idée est proche de la recherche des racines d'une fonction, elle consiste à tracer une droite entre deux point du relief correspondant aux deux extrémités d'un interval donné. Ensuite, calculer l'intersection de cette droite avec l'axe de vue. Le nouvel interval aura pour première extrémité ce point d'intersection. La deuxième extrémité est choisie parmi celle de l'étape précédente, en fonction de la position de l'intersection par rapport au relief.

D'après la figure 2.6, l'axe de vue a pour équation :

$$(u,v,w) = (u_i, v_i, w_i) + (u_j - u_i, v_j - v_i, w_j - w_i) \cdot \lambda$$

La droite entre les deux points extrêmes appartenant au relief (la ligne sécante) a pour équation :

$$(u,v,w) = (u_i, v_i, D_i) + (u_j - u_i, v_j - v_i, D_j - D_i) \cdot \lambda$$

Le point d'intersection entre ces deux droites vérifie l'égalité suivante:

$$(u_i, v_i, w_i) + (u_j - u_i, v_j - v_i, w_j - w_i) \cdot \lambda = (u_i, v_i, D_i) + (u_j - u_i, v_j - v_i, D_j - D_i) \cdot \lambda$$

La résolution de ce système donne la valeur du paramètre λ:

$$\lambda = \frac{D_i - w_i}{(w_j - w_i) - (D_j - D_i)}$$

Finalement, le point d'intersection est retrouvé à partir de l'équation de l'axe de vue :

$$(s,t,k) = (u_i, v_i, w_i) + \frac{D_i - w_i}{(w_j - w_i) - (D_j - D_i)} (u_j - u_i, v_j - v_i, w_j - w_i)$$

La mise à jour des extrémités s'obtient par la relation suivante:

$$\begin{cases} Si\ \ k < D(s,t)\ \ Alors\ \ (u_i, v_i, w_i) \leftarrow (s,t,k) \\ Sinon\ \ (u_j, v_j, w_j) \leftarrow (s,t,k) \end{cases}$$

2.3 Techniques itératives fiables

Contrairement aux méthodes non fiables, les techniques itératives fiables essayent de localiser la première intersection. Pour ce faire, elles se basent sur des informations supplémentaires qui sont précalculées et stockées dans la carte de déplacement (souvent dans le canal *bleu*). La recherche linéaire peut être qualifiée de méthode quasi-fiable car elle dépend fortement du nombre d'itérations et risque de rater la première intersection. Les autres techniques fiables ne souffrent pas de ce problème mais, d'un autre coté, elles ne garantissent pas d'atteindre précisément la première intersection, spécialement si le nombre d'itérations est trop faible.

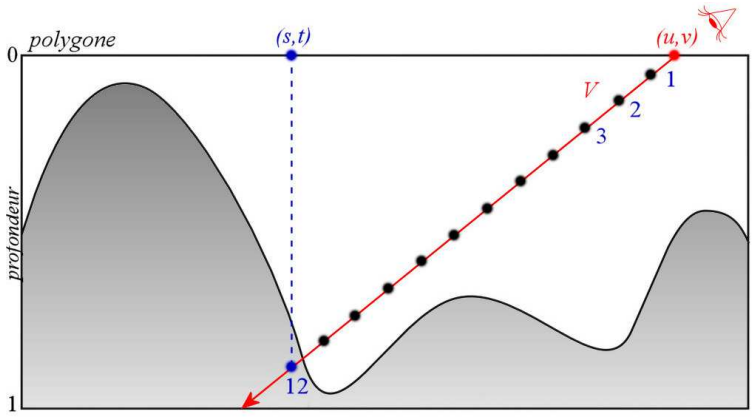


Fig. 2.7 - Recherche linéaire de l'intersection.

2.3.1 Recherche linéaire

Cette technique intuitive, introduite parallèlement dans [YJ04, BT04, MM05, POC05], consiste à commencer la recherche depuis le pixel courant (*u*,*v*), et avancer avec des pas réguliers suivant la direction de vue $\vec{V}$. Cette opération est répétée tant que la position actuelle est au-dessus du relief (figure 2.7). Le pas est calculé en fonction du nombre d'itérations *nb* à l'aide de la formule suivante :

$$pas = \frac{1}{nb} \cdot \frac{1}{V_z}$$

La position suivante est obtenue par la formule:

$$\begin{cases} (u_{i+1}, v_{i+1}, w_{i+1}) = (u_i, v_i, w_i) + pas \cdot \vec{V} & si \quad w_i < D_i \\ (s,t) = (u_i, v_i) & sinon \end{cases}$$

2.3.2 Traçage de sphères

Il s'agit d la première méthode qui se base sur le précalcul de l'espace vide afin de converger rapidement vers le premier point d'intersection entre l'axe de vue et le relief. Cette technique introduite dans [Don05] crée d'abord une carte 3D à partir de la carte des profondeurs 2D où chaque texel 3D reçoit la valeur 1 s'il appartient au relief et 0 sinon, puis calcule sa transformation distance Euclidienne qui fournit à chaque texel la distance minimale au relief. Pendant la recherche de l'intersection, un traçage de sphères permet à chaque itération de se rapprocher significativement de la première intersection avec le relief (figure 2.8), la formule utilisée et la suivante :

$$(u_{i+1}, v_{i+1}, w_{i+1}) = (u_i, v_i, w_i) + R[u_i, v_i, w_i] \cdot \vec{V}$$

où R désigne la transformation distance Euclidienne 3D de la carte des profondeurs.

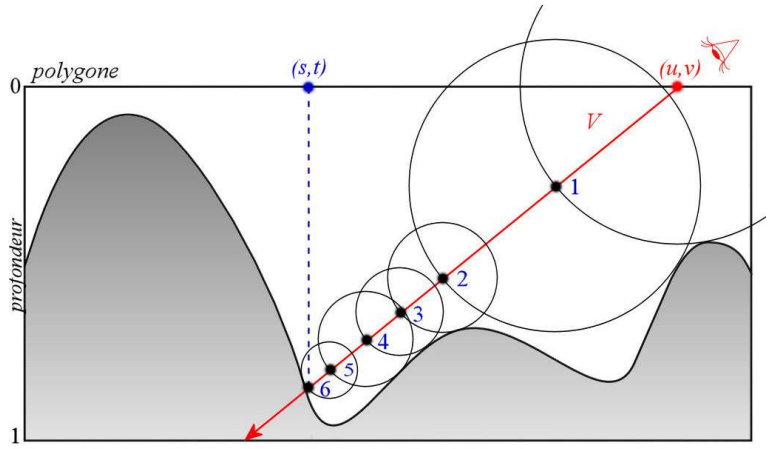


Fig. 2.8 - Principe de la technique de traçage de sphères.

2.3.3 Carte de dilatation et d'érosion

Cette technique proposée dans [KRS05] utilise deux cartes supplémentaires calculées à partir de la carte des profondeurs en s'appuyant sur des opérations de filtrage du maximum et du minimum via les formules suivantes :

$$M_{dil}^{\delta}(S) = \max\{M(S') : |u-u'| < \delta \cap |v-v'| < \delta\}$$
$$M_{ero}^{\delta}(S) = \min\{M(S') : |u-u'| < \delta \cap |v-v'| < \delta\}$$

où $S=(u,v)$ et $\delta>0$ désigne la taille choisie de la région suivant les directions de u et de v.

Les cartes de dilatation et d'érosion permettent d'avoir en chaque pixel une région sûre (espace vide). Les intersections successives de l'axe de vue avec ces régions permettent de converger au point d'intersection avec le relief (figure 2.9).

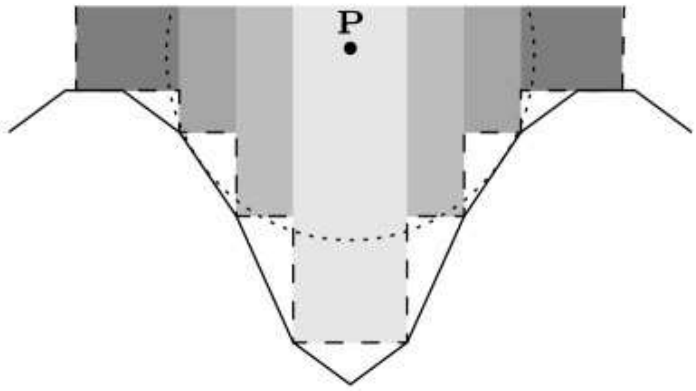


Fig. 2.9 - Principe des cartes de dilatation et d'érosion [KRS05]. Les régions sous forme de boites peuvent, en principe, coder plus d'espace vide que les sphères.

2.3.4 Traçage de cônes

Le principal défaut de la technique de traçage de sphère est l'utilisation des textures 3D. La technique de traçage de cônes introduite dans [PP94, Dum06] permet de surmonter ce problème en proposant de remplacer les sphères par des cônes ouverts vers le haut. Ces derniers sont calculés à partir de la carte de profondeurs puis stockés dans un canal libre de la carte de déplacement. Les intersections successives entre l'axe de vue et les cônes permettent d'atteindre le point d'intersection avec le relief en un minimum d'étapes. La formule utilisée est la suivante :

$$(u_{i+1}, v_{i+1}, w_{i+1}) = (u_i, v_i, w_i) + \frac{C[u_i, v_i] \cdot (D[u_i, v_i] - w_i)}{\left\|\vec{v}_{xy}\right\| + C[u_i, v_i]} \cdot \vec{v}$$

où $\vec{v} = \vec{V} / V_z$ est le vecteur normalisé par rapport à la profondeur (c-à-d v_z=1). D est le canal de profondeurs de la carte de déplacement, et C est le canal des ratios des cônes. Cette technique sera abordée en détail dans le chapitre 3.

2.3.5 Placage de déplacement pyramidal

Proposée dans [OKL06, KO07] puis dans [TIS08], cette technique s'inspire de la technique du MIP-Mapping qui est utilisée dans les algorithmes de filtrage et des niveaux de détail. L'idée est de créer une structure pyramidale des profondeurs, en calculant à chaque fois une carte qui est quatre fois plus petite que la précédente, et en prenant le maximum de la profondeur de chaque groupe de quatre pixels. Toutes ces sous-cartes sont stockées dans une seule texture

triangulaire. Le point d'intersection entre le rayon de vue et les profondeurs est obtenu par les intersections successives avec les lignes horizontales représentant la profondeur maximale de chaque niveau de la pyramide. Le premier niveau donnera finalement le point d'intersection (voir la figure 2.10). Cependant cette méthode peut donner des résultats erronés dans certains cas particuliers.

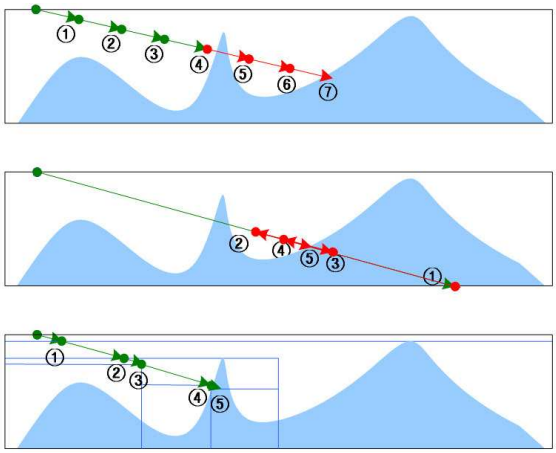


Fig. 2.10 - Principe de la recherche pyramidale selon [OKL06]. Mise en évidence de l'échec de la recherche linéaire (en haut) et la recherche binaire (au milieu). (en bas) La recherche pyramidale permet d'atteindre le point d'intersection.

2.4 Techniques itératives hybrides

Les techniques itératives hybrides essayent de combiner la fiabilité des techniques itératives fiables avec la précision et la rapidité des techniques non fiables. En effet, ces techniques agissent en deux phases : La première phase s'appuie sur une technique fiable ou quasi-fiable, afin de bien situer la zone de la première intersection. La seconde phase utilise une technique non fiable, afin d'affiner au maximum le point d'intersection.

2.4.1 Placage de relief

Cette méthode, introduite dans [POC05], est une extension d'une autre technique appelée *placage de texture en relief* [OBM00, Oli00]. Il s'agit d'une méthode très utilisée, car elle ne nécessite pas un prétraitement particulier. La technique commence par faire une recherche linéaire pour localiser l'interval où se trouve la première intersection (environ 20 étapes). Ensuite, cet interval est divisé successivement par deux, afin de mieux localiser l'intersection (environ 5 étapes pour la recherche binaire). Le placage de relief est très rapide et donne de bons résultats (voir la figure 2.11). Cependant, lorsque l'échelle de profondeur est assez grande, ou lorsque l'axe de vue rase la surface, quelques défauts deviennent nettement visibles.

Cette technique propose également la mise à jour de la profondeur des pixels dans le nuanceur de pixels. Cela permet d'avoir une interpénétration correcte entre le microrelief et les autres objets de la scène (figure 2.11, à droite)

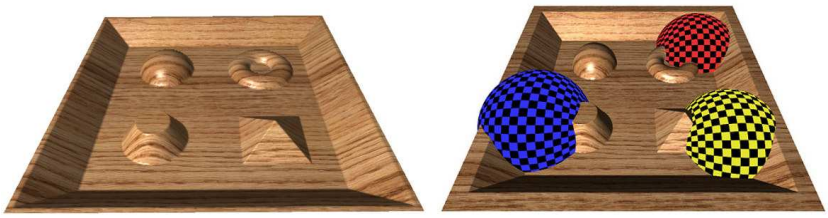


Fig. 2.11 - Technique du placage de relief [POC05]. L'image de gauche met en évidence l'interpénétration avec d'autres objets de la scène.

2.4.2 Recherche linéaire et sécante

La recherche linéaire/sécante a été parallèlement introduite dans [YJ04, BT04]. Elle consiste à effectuer une recherche linéaire pour délimiter la première intersection dans un petit interval. Ensuite, calculer l'intersection entre la ligne sécante au microrelief dans cet interval et l'axe de vue. Ce qui permet d'avoir une bonne approximation du point de l'intersection. Afin d'augmenter les performances, et diminuer les défauts de cette technique, plusieurs améliorations ont été proposées Dans [Tat06a, Tat06b]. Comme par exemple : La prise en charge des ombres adoucies, l'adaptation du nombre d'étapes de la recherche linéaire à l'angle que font la surface du polygone et l'axe de vue, ou encore l'adaptation de l'algorithme au niveau de détails requis.

Les résultats de cette technique peuvent encore être améliorés si l'intersection sécante est répétée plusieurs fois [RSP06]. Cependant, contrairement à la recherche binaire, la recherche sécante est plus coûteuse en terme de calcul, elle est alors limitée à 2 ou 3 étapes (figure 2.12).

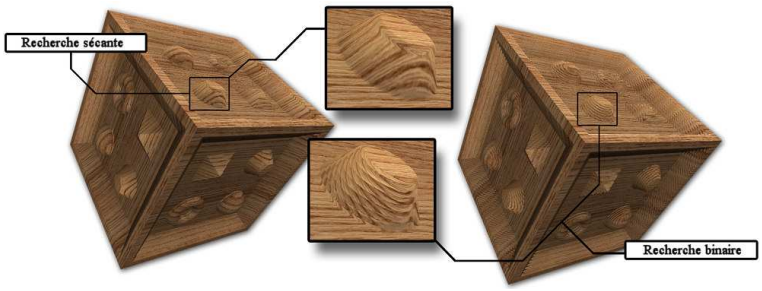


Fig. 2.12 - Comparaison entre la recherche sécante itérative et la recherche binaire [RSP06].

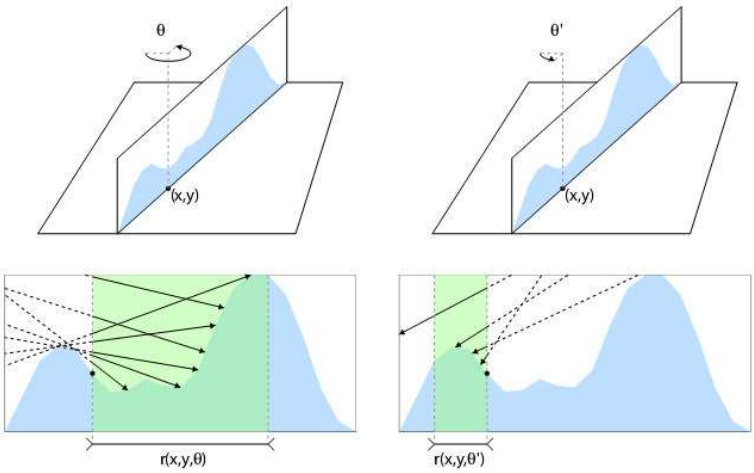


Fig. 2.13 - Technique de traçage de cylindres [BD06b].

2.4.3 Traçage de cylindres

Le traçage de cylindres [BD06b] a été la première méthode à proposer la combinaison d'une recherche par encodage de l'espace vide et une recherche binaire. L'étape de prétraitement consiste à définir pour chaque pixel de la carte de profondeurs, un rayon d'un cylindre à l'intérieur duquel, aucune direction de vue ne peut transpercer le relief plus d'une fois (voir la figure 2.13). Pendant la phase de recherche de l'intersection, ce rayon permet d'avancer sans risque de sauter la première intersection. La deuxième étape consiste à effectuer une recherche binaire entre les deux dernières positions. Cette technique permet d'éliminer les artefacts dus aux angles de vue rasant la surface. Cependant, le prétraitement de la carte de déplacement est très lourd, notamment pour des cartes de profondeurs avec une grande résolution.

2.4.4 Traçage de cônes relaxés

La technique de traçage de cônes classique n'autorise pas les cônes à percer le relief. Cette limitation provoque parfois l'arrêt de la recherche avant d'atteindre le point d'intersection, ce qui cause des distorsions très visibles dans certaines zones du relief, particulièrement avec un nombre restreint d'étapes. La technique de traçage de cônes relaxés [PO07] relaxe justement cette contrainte, et permet aux cônes de percer le relief, mais pas plus d'une seule fois. Le traçage de cônes doit être suivi par une recherche binaire ou sécante afin de situer précisément le point d'intersection. Cette technique sera traitée avec plus de détail dans le chapitre 3.

2.5 Auto-ombrage

L'auto-ombrage correspond aux ombres induites par le microrelief sur lui-même. Il s'agit d'un phénomène très important qu'il faudra prendre en compte afin de restituer avec plus de fidélité les surfaces avec mesostructures. Les techniques que nous avons présentées dans les sections précédentes ne gèrent pas l'auto-ombrage. Il faudra alors les combiner avec d'autres méthodes qui permettent l'introduction de cette fonctionnalité. Certaines de ces techniques s'appuient sur la carte de déplacement et sur un modèle d'illumination local, tandis que d'autres sont plus complexes, et se basent sur des photographies réelles ou synthétiques, couplées à des équations de simulation physique de la lumière.

2.5.1 Carte d'horizon

La technique de placage d'horizon [Max88] a été introduite spécialement pour ajouter l'effet d'auto-ombrage au placage de bosselures. L'algorithme s'appuie sur une carte d'horizon, calculée à partir de la carte de hauteurs. Cette carte sauvegarde dans chaque pixel, l'élévation pour laquelle le pixel n'est plus visible à partir de l'un des huit azimuts retenus. Dans la phase de rendu, la carte d'horizon est sollicitée pour décider si un pixel sera ombré en fonction de la position de la source lumineuse. Bien que cette technique ait été adaptée pour le rendu temps réel dans [SC00] (figure 2.14), elle reste néanmoins assez coûteuse en terme de calcul (plusieurs passes), et en terme d'occupation mémoire (carte d'horizon 3D).

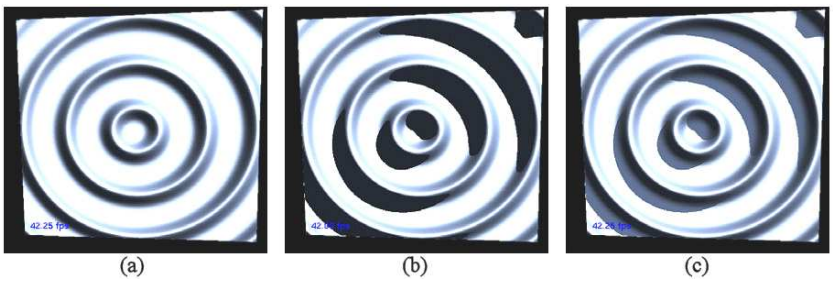


Fig. 2.14 - Placage d'horizon interactif [SC00]. (a) Placage de bosselures. (b) Placage d'horizon. (c) Placage d'horizon avec des ombres douces.

2.5.2 Carte de visibilité

Une technique plus adaptée au rendu temps réel a été proposée dans [HDKS00]. Cette technique repose sur une carte qui stocke des ellipses de visibilité. A partir de chaque point du relief, des rayons sont lancés dans toutes les directions à travers l'hémisphère centré sur ce point. Le pixel correspondant à ce point sera considéré comme illuminé, dans une direction donnée, si

le rayon correspondant quitte le relief sans intersection. Tous les rayons qui ne percent pas le microrelief sont projetés sur sa base, formant ainsi un ensemble de points. Cet ensemble est approché par une ellipse caractérisée par ces deux rayons et sa direction. Toutes ces informations sont stockées dans la carte de visibilité. Pendant la phase de rendu, il suffit de projeter le rayon lumineux sur la base du microrelief, le pixel sera illuminé si cette projection se situe à l'intérieur de l'ellipse correspondante.

2.5.3 Traçage de rayons de lumière

Le traçage de rayons de lumière est une technique d'auto-ombrage dont le principe est assez simple. Elle a été utilisée dans [WWT+03, POC05]. Elle n'est cependant valable que pour les techniques qui effectuent une recherche exacte du point de l'intersection de l'axe de vue avec le microrelief, car cette même procédure sera utilisée pour le calcul des ombres. En effet, un rayon de lumière est d'abord tracé entre le point d'intersection et la source lumineuse. L'intersection de ce rayon avec le relief donnera un autre point d'intersection. Si les deux intersections se confondent, alors le pixel sera illuminé, sinon il sera considéré dans l'ombre (voir la figure 2.15). Dans [Tat06a, Tat06b], une technique similaire est proposée pour la prise en charge des ombres douces.

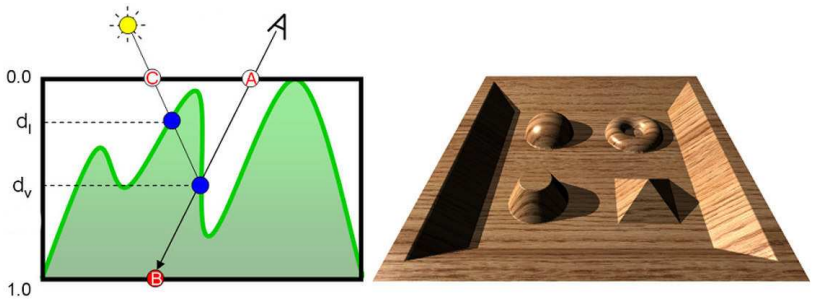


Fig. 2.15 - Auto-ombrage avec traçage de rayons de lumière [POC05]. L'image de gauche met en évidence le support de l'auto-ombrage.

2.5.4 Fonction d'illumination globale

L'auto-ombrage peut être considéré comme l'un des nombreux phénomènes lumineux complexes, caractérisant une surface donnée. L'ensemble de ces phénomènes a été formulé dans [Kaj86] par la luminance réfléchie depuis un point *x* dans la direction $\vec{\omega}_{out}$:

$$L_{out}(x,\lambda,\vec{\omega}_{out}) = L_e(x,\lambda,\vec{\omega}_{out}) + \int_{\Omega_{in}} \rho(x,\lambda,\vec{\omega}_{in},\vec{\omega}_{out}) \cdot L_{in}(x,\lambda,\vec{\omega}_{in}) \cdot \cos\theta_{in} \cdot d\omega_{in}$$

avec les paramètres suivants:

- L_e est la luminance intrinsèque.
- L_{in} est la luminance reçue depuis la direction $\vec{\omega}_{in}$.
- L_{out} est la luminance totale émise dans la direction $\vec{\omega}_{out}$.
- λ est la longueur d'onde de la lumière incidente. Pour le modèle RVB, on calcule la somme des luminances correspondantes aux longueurs d'onde des couleurs *rouge*, *verte* et *bleu*.
- Ω_{in} est l'ensemble des directions depuis lesquelles, le point *x* peut recevoir de l'énergie lumineuse.
- $\vec{\omega}_{in}$ et $\vec{\omega}_{out}$ sont les directions incidente et réfléchie, définies par leurs coordonnées cylindriques respectives $(\theta_{in}, \varphi_{in})$ et $(\theta_{out}, \varphi_{out})$ au point *x* (voir la figure 2.16).
- La fonction ρ a été formulée dans [Bli77, NRH+77]. Il s'agit de la fonction de distribution de réflectance bidirectionnelle BRDF (*Bidirectional Reflectance Distribution Function*). Elle est elle-même définie par :

$$\rho(x, \lambda, \vec{\omega}_{in}, \vec{\omega}_{out}) = \frac{L_r(x, \lambda, \vec{\omega}_{in}, \vec{\omega}_{out})}{L_i(x, \lambda, \vec{\omega}_{in}) \cdot \cos\theta_{in} \cdot d\omega_{in}}$$

  où L_i est la luminance du rayon incident. La fonction *L* est donc fortement récursive !

La BRDF telle qu'elle est définie ne permet pas de simuler l'apparence des mesostructures. En plus, c'est une fonction très complexe à échantillonner. Pour surmonter ces limitations, une nouvelle fonction a été proposée dans [DvGNK99]. Cette fonction appelée *fonction de texture bidirectionnelle* ou BTF (*Bidirectional Texture Function*), est basée sur le placage de textures 2D. Elle définit une fonction 6D, qui est échantillonnée en variant les directions de vue et de lumière. Bien que la BTF permette de reproduire les mesostructures plus fidèlement que la BRDF, son temps d'acquisition, son occupation mémoire, et son temps de calcul, restent très coûteux pour le rendu temps réel. Les textures tenseurs (*Tensor Texture*), introduites dans [VT04], permettent d'améliorer la compression de la BTF, et accélèrent son échantillonnage. De même que la fonction de texture avec coquille STF (*Shell Texture Function*), qui s'appuie sur le placage de photons, et sur une coquille qui couvre le maillage [CTW+04].

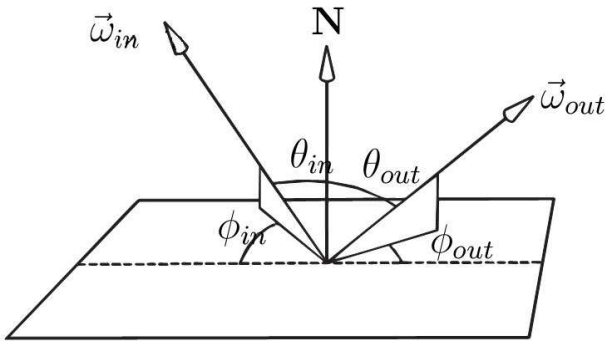


Fig. 2.16 - Paramètres de la fonction de réflectance bidirectionnelle.

Les cartes de texture polynomiales [MGW01], ou PTM (*Polynomial Texture maps*), sont certainement le modèle physique le plus abordable pour le rendu de mesostructures. En effet, cette technique simplifie la BTF, en variant seulement la direction de la lumière, et en fixant la direction de vue (figure 2.17). Le calibrage de caméra et ainsi évité, et le volume d'informations à traiter se retrouve considérablement réduit.

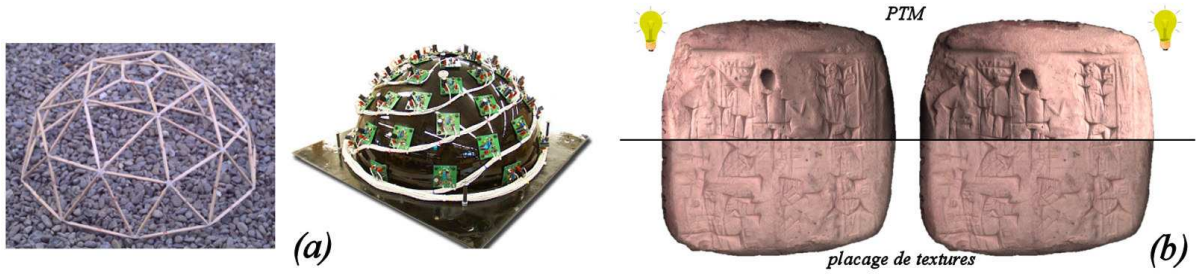


Fig. 2.17 - Cartes de texture polynomiale (PTM) [MGW01]. (a) Deux dispositifs pour échantillonner les PTMs. (b) Comparaison entre les PTMs et le placage de texture classique, en variant la position de la lumière.

2.6 Traitement de la silhouette

La silhouette d'un objet est visible sur les bords de la projection de celui-ci. Ce sont les parties où le rayon de vue rase la surface de l'objet (figure 2.18). Les techniques que nous avons décrites jusqu'à présent, supposent que tous les polygones de la surface sont coplanaires. L'approximation planaire est plutôt acceptable sur toutes les parties de l'objet, sauf sur les bords où la silhouette n'est pas convenablement restituée. Trois méthodes ont été proposées pour la prise en charge de la silhouette : La première est basée sur l'approximation quadratique, la seconde effectue une extrusion du maillage, quant à la dernière approche, elle se base sur une phase de prétraitement spécifique.

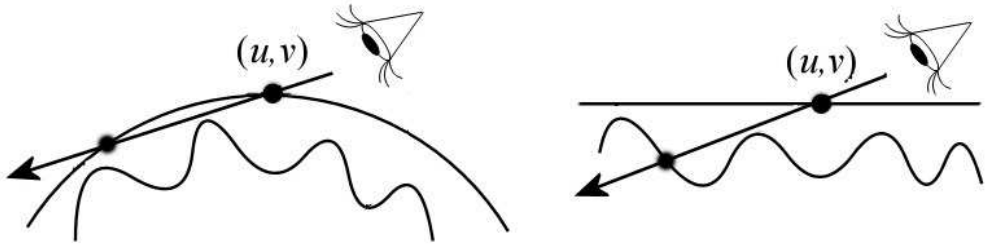


Fig. 2.18 - Mise en évidence de la silhouette. (A droite) une surface plane permet toujours d'avoir une intersection avec le relief. (A gauche) avec une surface courbe, le rayon de vue peut quitter la surface sans percer le relief. Le pixel (*u*,*v*) dans ce cas appartient à la silhouette de la surface.

2.6.1 Approximation quadratique

Afin de pouvoir représenter correctement la silhouette, cette technique s'appuie sur l'approximation locale la courbure en chaque point de la surface [OP05]. L'approximation retenue se base sur les quadriques [Pet02]. Cette approche donne de bons résultats, particulièrement lorsque la courbure de la surface est assez régulière (voir figure 2.19).

Soit T un ensemble de triangles partageant un sommet (x_k, y_k, z_k), et soit $P = \{p_1, p_2, \ldots, p_n\}$ l'ensemble des sommets de T. Les coordonnées de tous les sommets dans P sont exprimées dans l'espace tangent de p_k. Étant donné $P' = \{p'_1, p'_2, \ldots, p'_n\}$, avec $p'_i = (x'_i, y'_i, z'_i) = (x_i - x_k, y_i - y_k, z_i - z_k)$, nous calculons les coefficients quadratiques de p_k en utilisant les coordonnées 3D de tous les sommets P'. La surface quadrique, qui sera associée à chaque sommet, est de la forme :

$$ax^2 + by^2 = z$$

Les coefficients a et b sont calculés en résolvant le système linéaire $\boldsymbol{Ax=b}$ suivant :

$$\begin{pmatrix} x'^2_1 & y'^2_1 \\ x'^2_2 & y'^2_2 \\ : & : \\ x'^2_n & y'^2_n \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} z'^2_1 \\ z'^2_2 \\ : \\ z'^2_n \end{pmatrix}$$

Les coefficients a et b seront calculés pour chaque sommets, puis seront interpolés durant la phase de rendu pour chaque pixel. Pendant la recherche de l'intersection, la distance d, entre l'approximation planaire et la surface quadrique, sera évaluée par :

$$d = w - \left(au^2 + bv^2\right)$$

Lorsque d aura une valeur négative, le pixel courant sera ignoré, et fera donc parti de la silhouette.

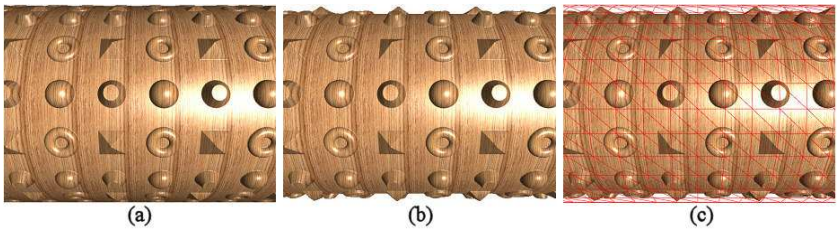


Fig. 2.19 - Prise en charge de la silhouette avec l'approximation quadratique [OP05]. (a) Le placage de relief traditionnel ne permet pas d'avoir une silhouette. (b) L'approximation quadratique permet un support correct de la silhouette. (c) Mise en évidence du maillage de base.

2.6.2 Placage à l'aide de coquille

Une des meilleures solutions pour la prise en charge de la silhouette, consiste à extruder le maillage de base, afin de créer une zone qui abritera le microrelief à plaquer sur la surface [HEGD04]. Cette coquille est obtenue en extrudant chaque triangle du maillage suivant les normales de ses trois sommets (voir figure 2.20). L'extrusion donne un prisme constitué de huit triangles qu'il faudra inclure dans le processus du rendu. Cependant, afin d'éviter quelques défauts de discontinuité liés à l'interpolation bilinéaire, le prisme est subdivisé en trois tétraèdres à l'aide d'un algorithmes décrit dans [ST90]. Des coordonnées de texture 3D sont alors attribuées aux sommets créés, en fonction des coordonnées des textures 2D, et de la position de ces sommets (à la base ou en surface).

L'utilisation de coordonnées barycentriques, introduites dans [PBFJ05], permet de définir une relation entre chaque point 3D contenu dans le prisme, et un texel unique de la carte de déplacement. Cela permet de réduire les distorsions du relief. Dans [DLP05], l'utilisation de textures 3D semi-transparentes permet le support de quelques fonctionnalités plus avancées. La division du prisme en tétraèdres n'est pas la seule solution pour réduire les distorsions. En effet, dans [JMW07], une fonction de lissage, couplée à des patchs de Coons, permet d'éliminer fortement les distorsions, et produit ainsi des résultas très satisfaisants.

Le principal défaut de l'utilisation de la coquille est la densification du maillage, qui est au moins multiplié par huit. Or, cette densification pénalise fortement l'utilisation des solutions à base de coquille pour le rendu temps réel. Elles restent cependant une alternative très prometteuse dans l'avenir, notamment avec l'évolution rapide que connaît le matériel graphique.

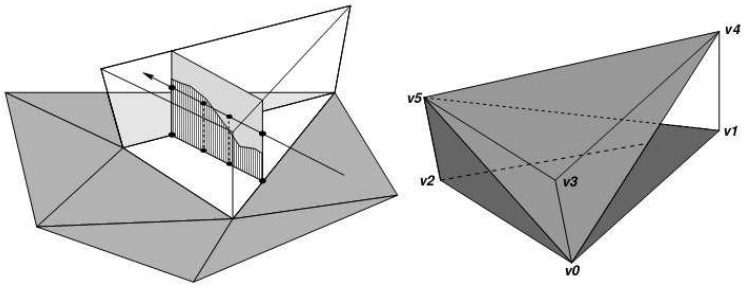


Fig. 2.20 - Placage de déplacement à l'aide d'une coquille [HEGD04]. (A gauche) Extrusion de chaque polygone suivant les normales aux sommets. (A droite) Le prisme est divisé en trois tétraèdres.

2.6.3 Placage de déplacement dépendant du point de vue

Introduite dans [WWT+03], cette technique adopte une approche de précalcul. L'idée principale consiste à calculer, pour chaque direction de vue, la distance entre chaque point d'un polygone et la surface de déplacement. Afin de pouvoir gérer également la silhouette, la courbure de la surface de base doit également être prise en considération. Une fonction dite VDM (*view-dependent displacement mapping*) à cinq dimensions est ainsi définie: $d_{vdm}(u,v,\theta,\varphi,c)$, où (u,v) désignent les coordonnées de texture, (θ,φ) sont les coordonnées sphériques de la direction de vue dans l'espace tangent, et c représente l'indice de courbure de la surface suivant la direction de vue.

Pendant la phase de rendu, lorsqu'un pixel (u,v) est traité, la courbure c_v de la surface suivant le direction de vue $\vec{V}$ est d'abord déterminée par :

$$c_v = \frac{c_{\max}\left(\vec{V}\cdot\vec{D}_{\max}\right)^2 + c_{\min}\left(\vec{V}\cdot\vec{D}_{\min}\right)^2}{1-\left(\vec{V}\cdot\vec{N}\right)^2}$$

où $c_{\max}$ et $c_{\min}$ sont les valeurs de courbure principale suivant les directions de courbure principale $D_{\max}$ et $D_{\min}$. (θ,φ) peuvent être déduites de $\vec{V}$. En ayant l'ensemble de ces valeurs, la mise à jour du pixel se fait à l'aide de la formule suivante :

$$(u',v') = (u,v) + d_{vdm}(u,v,\theta,\varphi,c_v)\cdot\vec{V}_{xy}$$

La fonction d_{vdm} représente une grande quantité de données. Pour cette raison, elle est compressée et stockée sous forme d'une texture 3D. Cette technique a été généralisée dans [WTL+04], afin de pouvoir gérer les cartes de profondeurs 3D, et pour limiter certaines distorsions, notamment en utilisant une coquille comme l'illustre la figure 2.21.

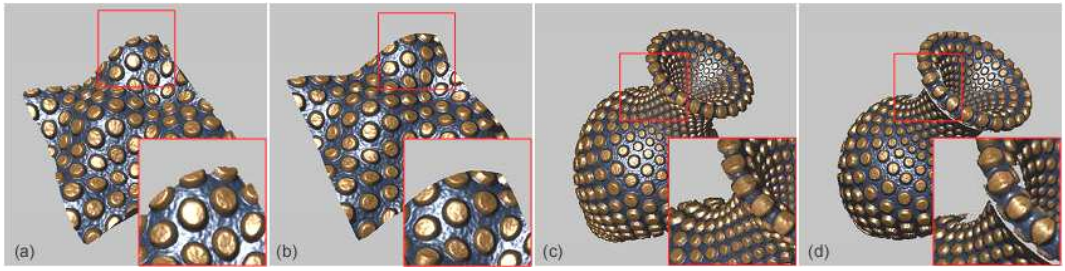


Fig. 2.21 - Comparaison entre la version simple et la version généralisée du placage de déplacement dépendant du point de vue [WTL+04]. (a,b) Une surface ouverte rendu avec la version généralisée puis la version simple. (c,d) Surface fermée fortement courbée avec la version généralisée puis la version simple.

2.7 Modélisation et rendu à base d'images

En se basant sur le placage de déplacement par pixel, certaines techniques ont été proposées pour pouvoir représenter des objets 3D en entier, et sans maillage polygonal, sauf sous une forme très basique comme des plans ou des boites. Pour réaliser cet objectif, la forme géométrique de l'objet 3D est codée sous forme d'images 2D ou 3D, et qui seront par la suite plaquées sur un maillage simplifié. Ces techniques, dites de *modélisation et rendu à base d'images* (ou IBMR pour *Image Based Modeling and Rendering*), sont très prometteuses, particulièrement pour des scènes complexes comportant un très grand nombre d'objets en second plan. Ces techniques ne sont cependant pas encore très au point, et nécessitent d'avantage d'optimisation.

La technique IBMR la plus utilisée consiste à créer six cartes de déplacement d'un objet 3D, et de les plaquer ensuite sur les six faces d'une boite englobante. La boite est par la suite rendue avec l'une des techniques de placage de déplacement par pixel.

Dans [OBM00], avant d'être plaquées sur la boite, les textures de déplacement sont déformées suivant l'angle de vue, en se basant sur les équations de McMillan [McM97]. Cette méthode permet aussi bien de représenter des mesostructures, que de rendre des objets 3D en entier. Yerex [YJ04] utilise le placage de déplacement inverse avec une recherche linéaire puis sécante. Dans [HEGD04], la coquille est utilisée comme support pour le placage, tandis que dans [BD06b], la technique s'appuie sur le traçage de cylindres, appliqué à six vues en perspective de l'objet 3D. Dans [TL08], des primitives paramétriques (cylindres, cônes et tores) sont rendues dans le GPU par traçage de rayons. Cette technique permet notamment d'optimiser la visualisation des plateformes industrielles.

L'utilisation de plusieurs faces n'est pas l'unique solution pour créer des modèles 3D. En effet, dans [PO06], deux ou quatre cartes de déplacement suffisent à rendre des objets complexes, en combinant la technique des imposteurs avec le placage de relief. Enfin, dans

[Rit06], des textures 3D sont utilisées pour coder un objet donné, ensuite, une carte de transformation distance Euclidienne 3D est générée afin de d'orienter un algorithme de traçage de sphères. La figure 2.2 présente quelques techniques IBMR.

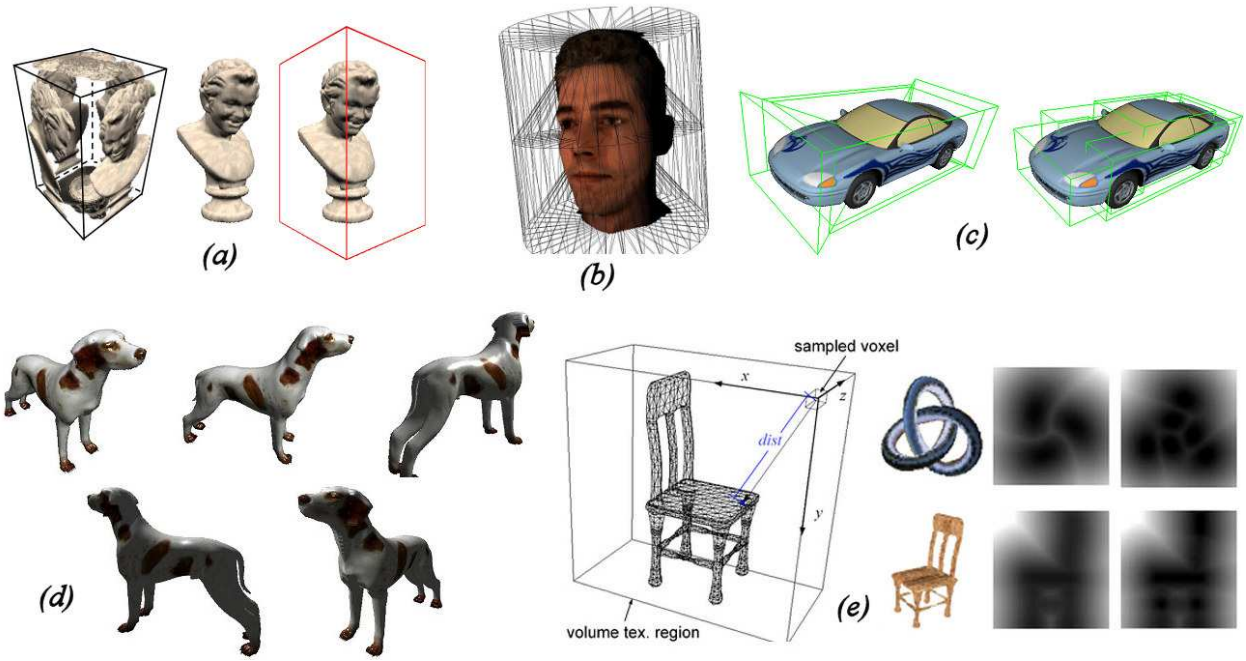


Fig. 2.22 - Exemple de techniques de modélisation et rendu à base d'images. (a) Textures de relief [OBM00]. (b) Placage à l'aide de coquille [HEGD04]. (c) Placage de plans en perspective avec traçage de cylindres [BD06b]. (d) Imposteurs avec placage de relief [PO06]. (e) Textures 3D avec traçage de sphères [Rit06]

2.8 Récapitulatif

Le placage de déplacement par pixel est une technique qui permet, à moindre coût, d'augmenter le réalisme des scènes 3D, en simulant les petits détails que peuvent avoir certaines surfaces. Cette technique est basée sur le placage de textures, mais les textures dans ce cas sont utilisées pour stocker un microrelief au lieu des couleurs traditionnelles. La technique repose également sur le traçage de rayons entre la caméra et les pixels projetés d'un objet 3D. L'objectif étant de calculer l'intersection entre ces rayons et le relief plaqué comme une texture sur l'objet à traiter.

Plusieurs stratégies ont été développées pour calculer le point d'intersection en un minimum de temps. Certaines de ces stratégies sont très simples, et se contentent de chercher une solution très approximative, tandis que d'autres sont plus complexes, et s'appuient sur le codage de l'espace vide autour du relief. Le tableau 2.1 représente un récapitulatif des différentes techniques suivant la nature des deux phases de recherche de l'intersection. La figure 2.23 donne une comparaison entre les techniques les plus significatives, en affichant le résultat du rendu d'un relief à petite échelle. La figure 2.24 fait de même, mais avec une échelle de relief plus conséquente.

		Phase I		
		Recherche linéaire	**Encodage de l'espace vide**	**Aucune**
Phase II	**Recherche binaire**	Placage de relief [POC05]	Traçage de cônes relaxés [PO07] Traçage de cylindres [BD06]	
	Recherche sécante	DMRC [YJ04] Occlusion parallaxe [BT04] Placage avec interval [RSP06]	Traçage de cônes relaxés [PO07]	
	Pente			Placage de parallaxe itératif [Pre06]
	Aucune	placage de parallaxe robuste [MM05]	Traçage de sphères [Don05] Erosion-Dilatation [KRS05] Traçage de cônes [Dum06] Recherche pyramidale [OK06]	Placage de bosselures [Bli78] Placage de parallaxe [KKI+01, Wel04]

Tab. 2.1 - Classification des techniques de placage de déplacement par pixel, suivant les méthodes de recherche retenues pendant les deux phases de recherche.

La notion de placage de déplacement a été étendue afin de pouvoir créer et afficher des objets 3D entiers, sans passer par le maillage traditionnel. Les résultats sont satisfaisants, néanmoins, la plupart des techniques proposées ne sont toujours pas très adaptées au rendu temps réel. Le placage de déplacement a été également utilisé pour simuler des effets autres que du simple microrelief, tels que la fourrure et les poils dans [MM05, DLP05], la surface de l'eau en mouvement dans [BD06a], ou des surfaces translucides à plusieurs couches dans [FPBB08].

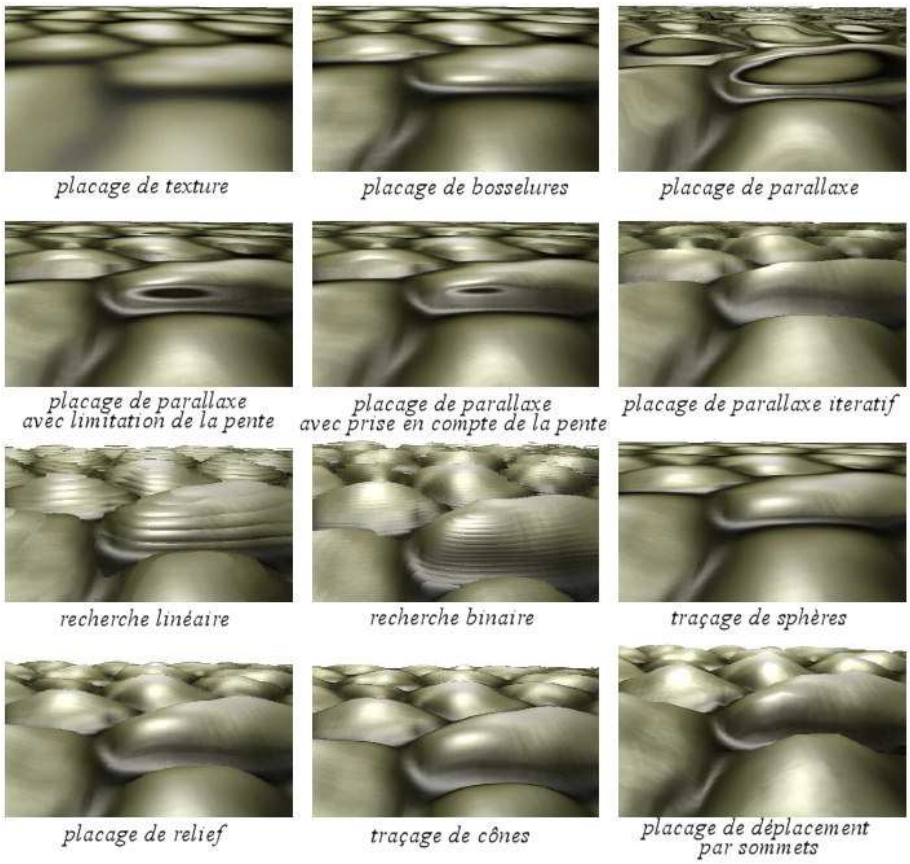


Fig. 2.23 - Comparaison entre les techniques du placage de déplacement pour un microrelief avec une petite échelle de profondeur [SKU08].

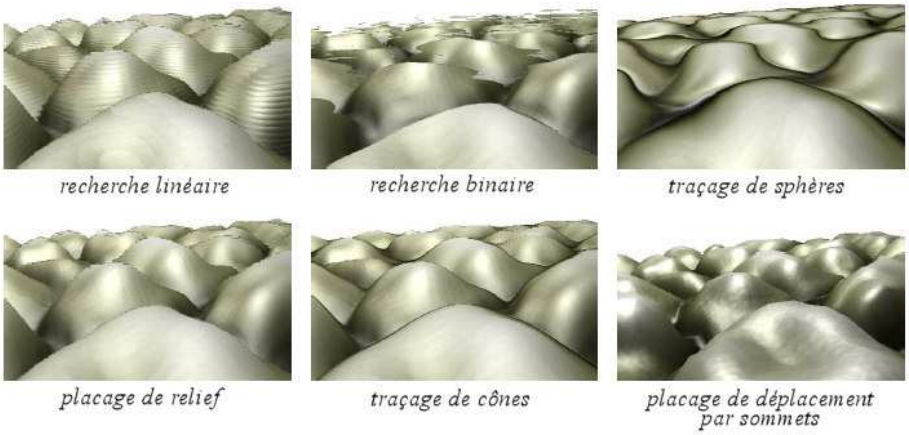


Fig. 2.24 - Comparaison entre les techniques du placage de déplacement pour un microrelief avec une échelle de profondeur plus conséquente [SKU08].

Chapitre 3

Placage de Déplacement Par Pixel avec Traçage de Cônes

La technique de traçage de cônes, utilisée pour le rendu de surfaces très détaillées en temps réel, est certainement l'une des meilleures techniques à l'heure actuelle. Malheureusement, dans ses deux versions: conservative et relaxée, cette technique souffre d'un lourd prétraitement à complexité quadratique. En plus, le traitement des textures non-carrées reste problématique. Dans ce chapitre, nous proposons un nouvel algorithme à complexité linéaire pour la génération des cartes de cônes relaxés. Nous proposons également d'utiliser l'algorithme HDDT (*Height Distributional Distance Transform*) *pour la génération des cartes de cônes conservatifs. Nous introduisons également une nouvelle méthode pour la prise en charge des textures rectangulaires en utilisant une rectification elliptique. Finalement, nous utilisons le rayon des cônes au lieu de leur ratio. Cela permet d'éliminer la contrainte limitant l'angle de la pente des cônes à $\pi/4$, ce qui améliore significativement les performances de la technique de traçage de cône.*

3.1 Introduction

Le traçage de cônes est l'une des meilleures techniques pour le rendu de microreliefs en temps réel. Afin de trouver l'intersection entre la direction de vue et le microrelief, cette technique utilise une carte de cônes pour encoder l'espace vide, et converger ainsi très rapidement vers le point d'intersection. Le traçage de cônes existe en deux versions : La première utilise des cônes conservatifs [Dum06], tandis que la deuxième utilise des cônes relaxés couplés à une recherche binaire [PO07]. Les deux variantes s'appuient sur des algorithmes à complexité $O(n^2)$ pour générer la carte de cônes (n étant le nombre total de pixels). Les deux processus de prétraitement calculent et stockent le ratio des cônes dans un canal libre de la carte de déplacement. Enfin, il faut noter que les deux versions de cette technique n'ont été initialement définies que pour des textures carrées.

En partant de ces remarques, nous avons proposé les améliorations suivantes :

- Un algorithme de prétraitement avec une complexité $O(n)$ au lieu de $O(n^2)$, utilisé pour la génération des cartes de cônes relaxés.
- Utilisation de l'algorithme *Height Distributional Distance Transform* HDDT [PP94] pour la génération des cartes de cônes conservatifs. L'HDDT est un algorithme à complexité $O(n)$, utilisé pour la génération des cartes de cônes pour le rendu de terrain.
- Extension aux textures non-carrées en suggérant une nouvelle approche qui effectue une rectification elliptique au lieu d'une pré-normalisation, sachant que cette dernière réduit considérablement la convergence du traçage de rayons.
- Calcul et stockage du rayon des cônes au lieu de leur ratio. Ainsi, nous pouvons utiliser des cônes avec un angle de pente allant jusqu'à $\pi/2$ au lieu de $\pi/4$. Cela améliore sensiblement le résultat pendant le rendu.

Afin de calculer la première intersection entre la direction de vue et les profondeurs formant le relief virtuel, certaines techniques s'appuient sur des approches dites *non fiables* (c-à-d linéaires, binaires ou sécantes). Ces techniques ont l'avantage de ne pas nécessiter un prétraitement important, et n'utilisent que la carte de profondeurs et ses dérivées partielles. Toutefois, ces méthodes risquent de sauter la première intersection, particulièrement pour des directions de vue rasant la surface, ou lorsque le relief est très fin. Ces situations entraînent des défauts très visibles lors du rendu. Afin de surmonter ce problème, des méthodes dites *conservatives* ont été

développées, comme le traçage de sphère, de cylindres, de cônes ou encore le traçage pyramidal. Ces techniques calculent et stockent des données supplémentaires au cours d'une phase de prétraitement. Par la suite, ces données sont utilisées pour sauter l'espace vide, afin de converger rapidement vers le premier point d'intersection sans l'éviter. Mais néanmoins, sans garantie de l'atteindre, spécialement lorsque le nombre d'étapes utilisées est insuffisant.

3.2 Traçage de cônes

Dans cette section, nous discuterons de deux techniques conservatives, le traçage de cônes conservatifs et la récente technique de traçage de cônes relaxés. Cette dernière combine le saut de l'espace vide avec une technique non fiable. Nous proposons ensuite d'utiliser le rayon des cônes au lieu de leur ratio. Finalement nous présenterons une méthode pour étendre ces techniques aux textures non-carrées. L'algorithme 3.1 synthétise les deux techniques, ainsi que les améliorations proposées.

3.2.1 Traçage de cônes conservatifs

Dans la technique du traçage de cônes conservatifs, Dummer [Dum06] assigne à chaque texel un cône ouvert orienté vers le haut, représentant l'espace vide autour de ce cône (tangentiel au relief). Le ratio de chaque cône est calculé durant une phase de prétraitement, puis stocké dans un canal libre de la carte de déplacement, appelée dés lors *carte de cônes*.

Soit $P_i=(x_i,y_i,z_i)$ la position actuelle sur l'axe de vue. Pendant la phase du traçage de cônes, la position suivante s'obtient par l'intersection entre l'axe de vue et le cône stocké dans le texel courant (x_i,y_i). Nous commençons d'abord par normaliser la profondeur du rayon de vue (c-à-d $\vec{v} = \vec{V} / V_z$, où $\vec{V}$ désigne le rayon de vue normé).

D'après la figure 3.1, nous avons la relation de similitude suivante :

$$\frac{c}{a} = \frac{\left\|\vec{v}_{xy}\right\|}{v_z} = \left\|\vec{v}_{xy}\right\| \quad \text{d'où} \quad c = a \cdot \left\|\vec{v}_{xy}\right\| \tag{3.1}$$

Nous avons également:

$$\frac{c}{b} = \frac{c}{D[P_i] - a - z_i} = cone\_ratio \tag{3.2}$$

En résolvant (3.1) et (3.2) par rapport à *a*, nous obtenons :

$$a = \frac{cone\_ratio \cdot \left(D[P_i] - z_i\right)}{\left\|\vec{v}_{xy}\right\| + cone\_ratio} \tag{3.3}$$

Sur l'axe de vue, nous avons la relation suivante :

$$\frac{P_{i+1} - P_i}{a} = \frac{\vec{v}}{v_z}$$

Puisque v_z=1, et d'après la relation (3.3), la prochaine position sur l'axe de vue sera calculée à partir de la relation suivante :

$$P_{i+1} = P_i + \frac{cone\_ratio \cdot (D[P_i] - z_i)}{\left\|\vec{v}_{xy}\right\| + cone\_ratio} \cdot \vec{v} \qquad (3.4)$$

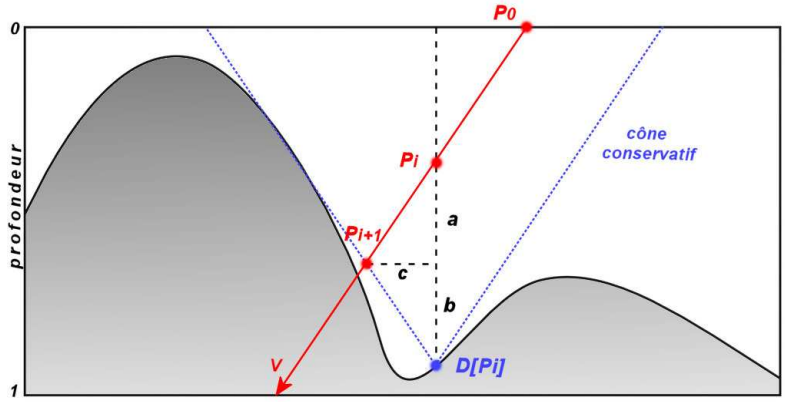


Fig. 3.1 - Traçage de rayons sur la carte de profondeurs (coupe). A chaque étape, la position suivante P_{i+1} du rayon est calculée en fonction de la position courante P_i du rayon de vue v et de la distance a

3.2.2 Traçage de cônes relaxés

Les cônes conservatifs sont définis d'une manière telle que le rayon de vue ne perse jamais le relief. Cette contrainte peut causer des distorsions dans certains cas, comme une recherche avec un nombre d'étapes insuffisant, ou dans le cas où l'axe de vue est rasant, ou encore avec des reliefs très fins. Ces distorsions, parfois très visibles, sont dues à l'arrêt prématuré du traçage de rayon. Le traçage de cônes relaxés [PO07] allège la contrainte conservative, et force le rayon à percer le relief dès que c'est possible. Cette approche a été d'abord utilisée avec des cylindres dans [BD06b].

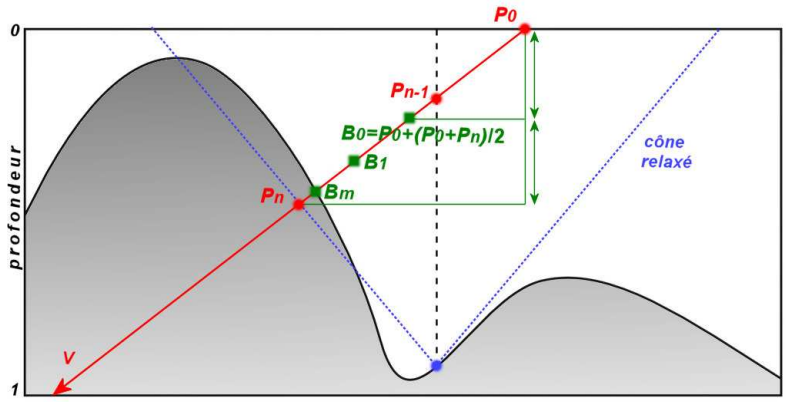


Fig. 3.2 - La phase de la recherche binaire associée au traçage de cônes relaxés. La recherche binaire est effectuée entre la dernière position du rayon P_n et la position de départ P_0.

Dans la technique de traçage de cônes relaxés, le lancer de rayon sera le même que dans le cas conservatif en utilisant l'équation (3.4). A la fin de cette étape, nous assumons que la position du rayon se trouve au dessous de la surface formant le relief. Dès lors, une recherche binaire est effectuée pour affiner le point d'intersection. Comme le montre la figure 3.2, la recherche binaire est effectuée entre la position de départ du lancer de rayon P_0, et la position actuelle P_n. Notons qu'il est possible d'utiliser le dernier point qui se situe au-dessus du relief P_{n-1}. Cependant, la sauvegarde de P_{n-1} nécessite un branchement conditionnel dans le code, ce qui ralentirait considérablement les performances du programme. La méthode sécante décrite dans [RSP06], peut également être utilisée au lieu de la recherche linéaire. Néanmoins, l'utilisation des cônes relaxés ne corrigera pas toutes les distorsions dues à l'approche conservative, car dans de nombreux cas, le rayon ne percera jamais le relief.

3.2.3 Utilisation du rayon des cônes au lieu de leur ratio

Les deux techniques de traçage de cônes, conservatifs et relaxés, calculent et utilisent le ratio des cônes. Ces ratios sont stockés pendant la phase du prétraitement dans le canal *bleu* de la carte de déplacement. Comme les textures utilisées dans le rendu temps réel sont souvent sous un format entier de 8 bits, le nombre de valeurs possibles pour le ratio n'est que de 256 valeurs. Pour cette raison, les deux techniques ont opté pour la restriction du ratio à la valeur 1 (c-à-d l'angle de la pente du cône à $\pi/4$). La technique conservative utilise la racine carrée du ratio afin d'avoir une meilleure distribution.

Pour éliminer cette contrainte, nous proposons de stocker le rayon des cônes calculé à la surface (c-à-d pour la profondeur 0). Comme le ratio du cône est égal au rapport entre le rayon et la profondeur, la combinaison du canal des rayons et celui des profondeurs permettra d'avoir

jusqu'à 256x256 valeurs possibles pour le ratio. Dès lors, il n'est plus nécessaire de restreindre ce ratio à la valeur 1, et qui peut théoriquement être infinie. Par conséquent, l'angle de la pente peut atteindre la valeur de $\pi/2$. L'utilisation du rayon des cônes va entraîner la modification de la relation (3.4). Comme nous avons :

$$cone\_ratio = \frac{cone\_rayon}{D[P_i]} \tag{3.5}$$

L'équation (3.4) devient alors :

$$P_{i+1} = P_i + \frac{cone\_rayon \cdot (D[P_i] - z_i)}{D[P_i] \cdot \|\vec{v}_{xy}\| + cone\_rayon} \cdot \vec{v}$$

3.2.4 Extension aux cartes de déplacement non-carrées

Les techniques faisant appel à la distance, comme le traçage de sphères ou de cônes, se confrontent au problème de l'espace non-orthonormé des textures non carrées. En effet, toutes les coordonnées de texture sont par la suite normalisées automatiquement par le pipeline graphique.

Les développeurs des précédentes techniques n'ont pas évoqué clairement cette problématique dans leurs articles respectifs, mais l'ont tout de même prise en considération dans leur code. Ils ont choisi de la traiter dans la partie prétraitement, en effectuant les calculs après la normalisation des coordonnées des texels. Cela évite le problème de déformation lié à la normalisation automatique lors du traçage de rayons. Cependant, ce choix réduit considérablement la largeur des cônes, et par conséquent, ralenti la convergence de l'algorithme de traçage de cônes.

La figure 3.3 donne une comparaison entre les deux approches pour le prétraitement des textures non-carrées. Dans la carte de profondeur rectangulaire (a), les texels *r* et *q* sont candidats pour donner le cône basé sur *p* (*q* et *r* ont la même profondeur). Ici, le point *q* est le plus proche de *p.* Il donne alors le cône bleu. En (b), une normalisation est effectuée avant le calcul des cônes. Dans ce cas, le texel *r* devient plus proche de *p* et donne le cône vert. Pendant la phase de rendu (c), nous pouvons clairement remarquer que le cône elliptique, déduit du cône bleu, va permettre d'avancer avec des pas plus importants suivant la majorité des directions $\vec{V}$, excepté suivant les directions $\vec{V}'$. Mais dans ce dernier cas, les différences entre les pas sont minimes.

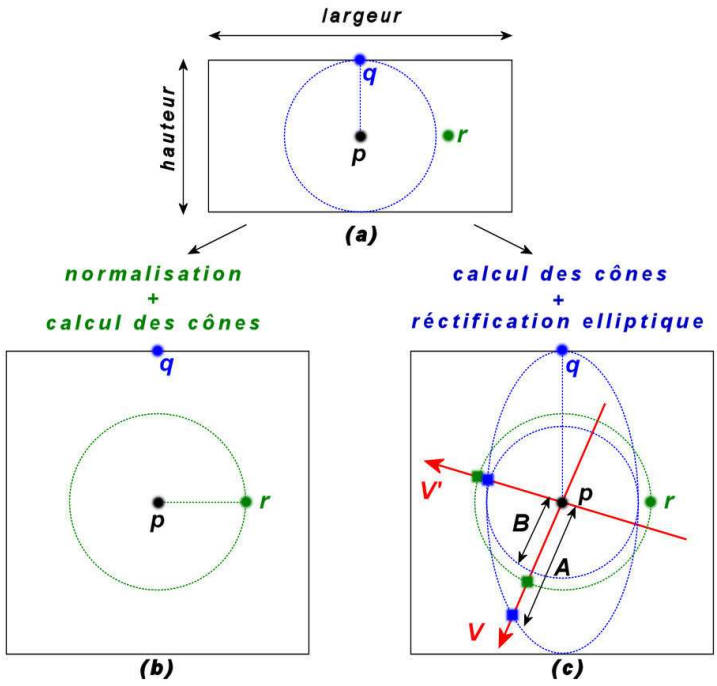


Fig. 3.3 - Comparaison entre les deux approches pour le prétraitement des textures non-carrées. (a) Une carte de profondeur rectangulaire. (b) Normalisation avant le calcul des cônes. (c) La phase de rendu.

Afin d'éviter cette perte de convergence, nous proposons d'ignorer la normalisation pendant le prétraitement. Nous rapportons alors les distances uniquement à l'une des deux dimensions, la largeur par exemple (c-à-d *largeur*→1). Bien évidemment, cela va induire des erreurs lors du traçage de cônes tel que nous l'avons défini précédemment. Car, étant dans un espace orthonormé, les cônes sont supposés arrondis. Ce qui n'est plus le cas après la normalisation automatique d'une texture non-carrée, où les cônes deviennent elliptiques. Nous pouvons noter alors que pour un rayon de vue donné, le rapport entre la distance parcourue dans le cas des cônes elliptiques, et celle parcourue dans le cas des cônes arrondis, reste constant et dépend uniquement du rayon de vue et du rapport entre les dimensions de la texture (rapport *A*/*B* sur la figure 3.3). Ce rapport peut être calculé facilement à partir de l'équation de l'ellipse, il vaut :

$$r = \frac{\left\|\vec{V}_{xy}\right\|}{\sqrt{V_x^2 + \rho^2 V_y^2}} \quad \text{avec} \quad \rho = hauteur/largeur$$

où nous pouvons vérifier que si *hauteur*=*largeur*, alors r=1 (texture carrée). Et quand V_y=0, alors r=1, parce que nous avons choisi de rapporter les distances à la largeur.

r est calculé une seule fois pour un rayon de vue donné, l'équation (3.4) devient alors:

$$P_{i+1} = P_i + a \cdot \vec{v}_\rho \quad \text{avec :} \quad \vec{v}_\rho = r \cdot \vec{v}$$

Algorithme 3.1 - Traçage de cônes

```
Entrée   : pixel courant (s,t), rayon de vue V normé, rayon de lumière L normé
           carte de couleur M, carte de cônes C et son Rapport R =largeur/hauteur
           composantes Ka, Kd, Ks, n (ambiante, diffuse, spéculaire et lustre)
           nombre de répétition T sur x et y, échelle de profondeur a
Sortie   : couleur du pixel courant (r,v,b)

p0  = (T.x*s, T.y*t, 0), p=p0          // point de départ
v  = V/Vz                              // normalisation de la profondeur
v.z = -v.z                             // la profondeur varie dans le sens contraire des z
D = norme(vxy)
vρ = v * D/(vx*vx+ρ*ρ*vy*vy)  // rectification elliptique

// recherche de l'Intersection
pour i allant de 1 à NB_ETAPES faire
       cone = C[p.x,p.y].bleu   // le paramètre du cône est récupéré à partir du canal bleu
       f = C[p.x,p.y].alpha     // la profondeur est récupérée à partir du canal alpha
       p = p + vρ *(cone*max(f-p.z,0.0)/(cone + D)        // stockage des ratios
       // p = p + vρ *(cone*max(f-p.z,0.0)/(cone + f*D)   // si stockage des rayons
fin pour

// recherche binaire (uniquement dans le cas des cônes relaxés)
vρ = (vρ*p.z)/2                // pas initial
p  = p0 + vρ                   // point de départ
pour i allant de 1 à NB_ETAPES_BIN faire
       f = C[p.x,p.y].alpha  // la profondeur est récupéré à partir du canal alpha
       vρ = vρ/2
       si( p.z < f ) alors   // si la position p est à l'extérieur du relief
              p = p + vρ     // avancer
       sinon
              p = p - vρ     // reculer
       fin si
fin pour

// fenêtrage
si p.x<0 ou p.x>T.x ou p.y<0 ou p.y>T.y alors
       annuler               // pas d'intersection
fin si

// calculer la normale au point (p.x,p.y)
N = (a*(2*F[p.x,p.y].rouge-1), a*(2*F[p.x,p.y].vert-1), 1)
normer(N)

// calcul de l'ombrage (Blinn-Phong)
(r,v,b) = M[p.x,p.y]           // récupérer la couleur de la texture
H = (L+V)/norme(L+V)
(r,v,b) = Ka*(r,v,b) + Kd*(r,v,b)*max(0,N.L) + ks*max(0, (H.N)^n )
```

3.3 Prétraitement

Étant donné une carte de profondeurs à niveau de gris, le processus de prétraitement crée la carte de déplacement (ou carte de cônes) comme une texture RVBA qui stocke les données nécessaires à la technique de traçage de cônes. Généralement, les profondeurs sont assignées au canal *alpha,* tandis que leurs dérivées partielles suivant *x* et *y* sont modulées dans l'intervalle [0,1], puis stockées respectivement dans les canaux *rouge* et *vert*. Durant la phase de rendu, la normale sera retrouvée à l'aide de la formule suivante :

$$N_{scale} = \frac{(-a \cdot dx, -a \cdot dy, 1)}{\|(-a \cdot dx, -a \cdot dy, 1)\|}$$

où *a* désigne l'échelle de profondeur, et qui peut être modulée en temps réel.

Enfin, le canal *bleu* est utilisé pour encoder les cônes conservatifs ou relaxés, en utilisant leur ratio ou leur rayon suivant la méthode retenue.

Dans cette section, nous allons commencer par présenter les algorithmes quadratiques utilisés dans les deux techniques de traçage de cônes. Ensuite, nous présenterons les algorithmes linéaires. Puis, nous parlerons de l'option de la répétition des cartes de déplacement.

3.3.1 Algorithmes quadratiques

3.3.1.1 Carte de cônes conservatifs

Le processus de prétraitement assigne un cône à chaque texel de la carte de profondeurs. Ce cône tant le plus large possible qui ne perce pas le relief formé par les profondeurs. Dummer [Dum06], au lieu d'utiliser l'algorithme HDDT [PP94], a proposé un algorithme $O(n^2)$ optimisé, dont la description est la suivante: pour chaque texel, calculer tout autour (c-à-d en traçant des carrés de l'intérieur vers l'extérieur) le ratio minimal des cônes basés sur ce texel. Cette recherche est arrêtée lorsque aucun autre cône ne peut être plus étroit que le cône minimal actuel. Le pseudocode correspondant est listé par l'algorithme 3.2.

Algorithme 3.2 - Cônes Conservatifs (Quadratique)

```
Entrée    : Carte de profondeurs D
Sortie    : Carte de cônes C

pour chaque texel t faire
  C[t] = VALEUR_MAX
  pour chaque texel p dans chaque carrée extérieur S faire
    si C[t] > S_Rayon/(D[t]-D[p]) alors
      arrêt     /* traiter le texel suivant t */
    sinon si D[t] > D[p] alors
      C[t] = min( dist(t,p)/(D[t]-D[p]), C[t] )
    fin si
  fin pour p
fin pour t
```

3.3.1.2 Carte de cônes relaxés

À la différence des cônes conservatifs, les cônes relaxés sont définis sous la contrainte suivante : Tant que le rayon de vue traverse le cône, il ne peut percer le relief plus d'une fois. Les cônes obtenus ainsi sont nettement plus larges que les cônes conservatifs.

Pour calculer la carte de cônes relaxés. Policarpo et Oliveira [PO07] ont utilisé un algorithme à complexité $O(n^2)$, décrit par le pseudocode de l'algorithme 3.3. L'idée est la suivante : Pour chaque texel source *t*, tracer un rayon vers chaque texel destination *p*. Ce rayon commence au point $(t_x,t_y,0)$ et pointe vers le point $(p_x,p_y,D[p])$. Pour chaque rayon ainsi obtenu, calculer l'intersection suivante (la deuxième) avec le relief, en utilisant une recherche linéaire fiable. Ensuite, utiliser cette intersection pour calculer le ratio du cône $C_p[t]$. Le ratio final $C[t]$ est donné par le plus petit de tous les cônes calculés pour *t*.

L'algorithme des cônes relaxés n'implémente pas une optimisation comme dans le cas conservatif. Par conséquent, il est beaucoup trop lent. En plus, l'algorithme tel qu'il est décrit dans [PO07] n'offre qu'une solution approximative. Cela induit certaines erreurs comme le montre la figure 3.4. Afin d'éviter ces défauts, le rayon utilisé pour trouver la deuxième intersection doit commencer à la position $(t_x,t_y,D[p])$ et non $(t_x,t_y,0)$. Ce rayon est horizontal, et devient alors inadapté à la recherche linéaire, qui a besoin de l'intersection avec la base du relief. Un algorithme à la Bresenham devrait alors être utilisé pour trouver l'intersection entre le rayon de vue et le relief. Toutefois, une telle solution rendra l'algorithme encore plus lent. Nous n'allons donc pas explorer cette voie, puisque l'algorithme linéaire que nous proposons ne se confronte pas à ce problème.

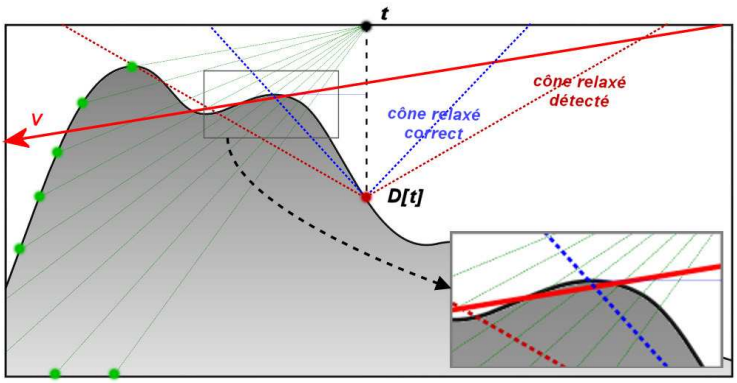


Fig. 3.4 - L'algorithme quadratique risque de sauter une bosse. Par conséquent, certains rayons de vue rasants peuvent percer et quitter le relief à l'intérieur du cône relaxé.

Algorithme 3.3 - Cônes Relaxés (Quadratique)

```
Entrée    : carte de profondeurs D
Sortie    : Carte de cônes C

pour chaque texel t faire
  C[t] = VALEUR_MAX
  pour chaque texel p faire
    ray  = (p.x-t.x, p.y-t.y, D[p]) /* rayon de vue */
    ray  = ray/ray.z                /* mise à l'échelle du rayon de vue */
    ray  = ray × (1-D[p])           /* deuxième mise à l'échelle */
    étape = ray/NB_ETAPES           /* longueur d'une étape */
    pos  = (p.x, p.y, D[p]) + étape
    pour i=0 à NB_ETAPES faire
      si pos.z ≤ D(pos.xy) alors
        pos = pos + étape
      sinon
        arrêt
      fin si
    fin pour i
    si D[t] > pos.z alors
      C[t]=min(dist(t,pos.xy)/(D[t]-pos.z), C(t))
    fin si
  fin pour p
fin pour t
```

3.3.2 Algorithmes linéaires

Le calcul des cônes se base sur les distances d_i entre un texel donné t_i et les autres texels p_j. En effet, le ratio du cône basé sur t_i est le rapport minimal entre les distances d_i et les différences de profondeur entre t_i et p_j. La distance Euclidienne joue alors un rôle primordial. Nous proposons donc d'utiliser l'EDT (*Euclidean Distance Transform*) d'une image binaire, que nous

allons définir par la suite, sachant qu'ils existent des algorithmes linéaires pour le calcul de l'EDT.

L'EDT est définie ainsi: à partir d'une image binaire ***B*** formée par un objet ***O*** et son arrière-plan ***O'***, une transformation de distance Euclidienne crée l'image (carte de distance) ***T***, dans laquelle la valeur de chaque pixel est la distance Euclidienne entre ce pixel et l'objet ***O***, c-à-d la distance au plus proche pixel de ***O***.

$$T(p) = \min\{dist_e(p,q), q \in O\}$$

où $dist_e$ est la distance Euclidienne.

Pour le calcul de l'EDT, nous avons opté pour l'algorithme 4SED (*4-point Sequential Euclidean Distance*) présenté dans [Dan80]. L'algorithme 4SED effectue une propagation dans les quatre directions de la distance relatives à l'objet ***O*** (Algorithme 3.4). Cet algorithme a l'avantage d'être très rapide, tout en donnant une très bonne approximation de la distance. La différence avec une approche exacte réside dans quelques pixels seulement. Néanmoins, d'autres algorithmes linéaires, donnant une distance exacte, peuvent être utilisés.

3.3.2.1 Carte de cônes conservatifs

Afin de présenter l'idée principale de cet algorithme, qui est une adaptation du Height Distributional Distance Transform (HDDT) présenté dans [PP94], imaginons que la carte de profondeurs ne contient que deux niveaux *a* et *b*. En se basant sur la figure 3.5 (image de gauche), nous pouvons facilement remarquer que le cône le plus large basé sur le texel *t*, et qui ne perce pas le relief, est celui dont le ratio est donné par:

$$C[t] = \frac{dist_e(t,p)}{b-a} \tag{3.6}$$

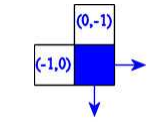
où *t* est un texel de profondeur *b*, et *p* est le texel le plus proche de *t* ayant une profondeur *a* (avec $b>a$).

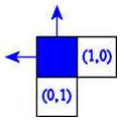
Dès lors, le problème se rapporte au calcul de la transformation distance Euclidienne de l'image binaire constituée des deux profondeurs *a* et *b*. Ainsi, le ratio des cônes à profondeur *b* sera calculé en se basant sur cette EDT à l'aide de la formule (3.6). Pour les texels à profondeur *a*, leur ratio ne sera pas calculé à partir de l'EDT (puisque leur valeur sera nulle), et vont donc conserver leur ratio initial, qui est le ratio maximal dans ce cas.

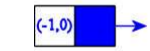
Algorithme 3.4 - Transformation Distance Euclidienne

```
Entrée        : Carte Binaire B
Sortie        : Carte de Distance T
Intermédiaire : Cartes de Distance suivant x et y DX, DY
/* initialisation */
pour x,y allant de 0,0 à largeur-1,hauteur-1 faire
              si B[x,y]==1 alors DX[x,y]=DY[x,y]=+∞   fin si
fin pour x,y
/* scanner l'image du haut vers le bas */
pour y allant de 0 à hauteur-1 faire
       /* scanner l'image de la gauche vers la droite */
       pour x allant de 0 à largeur-1 faire
              Combiner(x,y,-1, 0), Combiner(x,y, 0,-1)
       fin pour x
       /* scanner l'image de la droite vers la gauche */
       pour x allant de largeur-1 à 0 faire
              Combiner(x,y,1,0)
       fin pour x
fin pour y
/* scanner l'image du bas vers le haut */
pour y allant de hauteur-1 à 0 faire
       /* scanner l'image de la droite vers la gauche */
       pour x allant de largeur-1 à 0 faire
              Combiner(x,y,1,0), Combiner(x,y,0,1)
       fin pour x
       /* scanner l'image de la gauche vers la droite */
       pour x allant de 0 à largeur-1 faire
              Combiner(x,y,-1,0)
       fin pour x
fin pour y
/* calcul de la distance Euclidienne */
pour x,y allant de 0,0 à largeur-1,hauteur-1 faire
              T[x,y]= racine_carrée( (DX[x,y])^2+(DY[x,y])^2 )
fin pour x,y

procédure Combiner( entier x, entier y, entier decx, entier decy )
       entiers: x', y', dx, dy, dx', dy'
       x'=x+decx , y'=y+decy
       si x'≥0 et x'<largeur et y'≥0 et y'<hauteur alors
              dx=DX(x,y), dy=DY(x,y)
              dx'=DX(x',y')+abs(decx), dy'=DY(x',y')+abs(decy)
              si (dx')^2+(dy')^2<(dx)^2+(dy)^2 alors
                     DX[x,y]=dx', DY[x,y]=dy'
              fin si
       fin si
fin procédure
```







La carte de profondeurs étant une image à niveaux de gris, le relief est constitué de plusieurs niveaux de profondeur (voir la figure 3.5). Il suffit alors de calculer le ratio du cône de chaque niveau *l*, en se basant sur l'EDT de l'image binaire constituée de 1 pour les texels à profondeur supérieure à *l*, et 0 pour les autres texels. Le ratio des cônes sera modifié si sa valeur est inférieure à la valeur calculée précédemment. Le pseudocode de cet algorithme est détaillé dans l'algorithme 3.5. Nous utilisons exactement le même algorithme, que ce soit pour stocker le ratio des cônes, ou pour stocker leur rayon, en s'appuyant sur la formule (3.5).

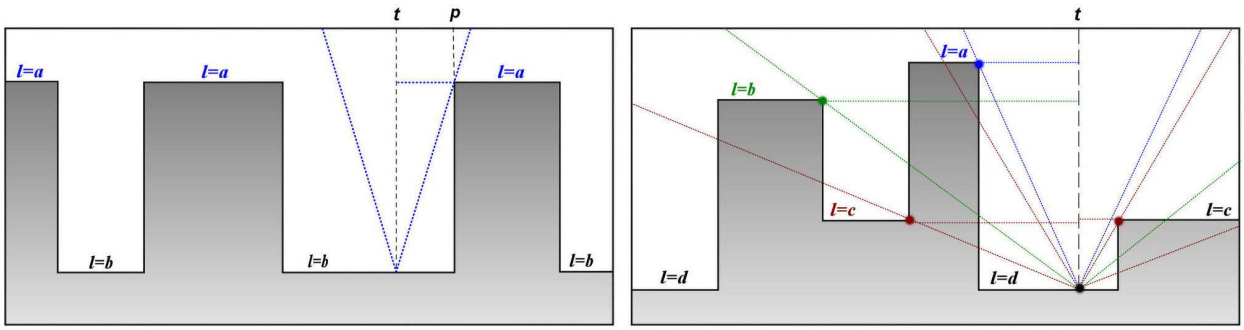


Fig. 3.5 - Calcul des cartes de cônes conservatifs à l'aide de l'algorithme linéaire. (A droite) Nous supposons que la profondeur a seulement deux niveaux *a* et *b* (A gauche) Extension du principe à plusieurs niveaux de profondeurs.

Algorithme 3.5 - Cônes Conservatifs (Linéaire)

```
Entrée       : Carte de profondeurs D
Sortie       : Carte de cônes C
Intermédiaire : carte binaire B, Carte de distances T

pour chaque niveau l de D faire
  pour chaque texel t faire
   B[t] = {0 si D[t] ≤ l, 1 sinon}
  fin pour t
  T = EDT( B )    /* distance des 1 aux 0 */
  Pour chaque texel t faire
    si D[t] > l alors
      C[t] = min( T[t]/(D[t]-l), C[t] )
    sinon
      C[t] = VALEUR_MAX
    fin si
  fin pour t
end for l
```

Le calcul de l'EDT d'une image binaire est un algorithme à complexité $O(n)$, où $n=largeur x hauteur$. Dans notre cas, le calcul de l'EDT est répété un nombre constant de fois, qui est égale au nombre de niveaux contenus dans la carte de profondeurs. Par conséquent, l'algorithme HDDT pour le calcul de la carte des cônes conservatifs est également un algorithme

linéaire. Bien évidemment, la complexité dépend du nombre de niveaux de profondeur, mais 256 niveaux (c-à-d 8 bits, que ce soit avec un format entier ou réel), est largement suffisant pour le rendu de microreliefs. Notons enfin que cet algorithme n'est pas limité à des profondeurs entières.

3.3.2.2 Carte de cônes relaxés

Quoique les algorithmes quadratiques pour le calcul des cônes conservatifs et des cônes relaxés soient assez différents, nous allons les rapprocher dans leur forme linéaire afin de pouvoir utiliser l'EDT. Nous pouvons remarquer que les cônes relaxés, tels qu'ils ont été définis dans [PO07], sont les cônes qui passent par les maxima locaux de la carte de profondeurs. Ainsi, le cône relaxé basé sur un texel *t* est le plus petit des cônes basés sur *t* passant par un des maxima locaux du relief. (Voir figure 3.6)

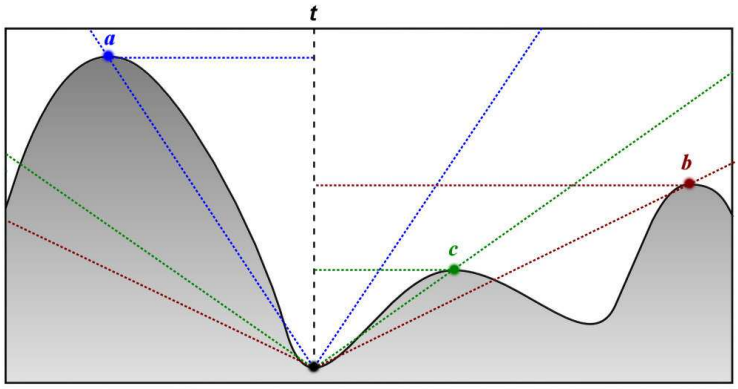


Fig. 3.6 - Principe de l'algorithme linéaire pour le calcul des cartes de cônes relaxés. Le cône relaxé relatif au texel *t* et le cône le plus étroit passant par un maximum local du relief (cône bleu).

L'algorithme est alors le suivant : Nous commençons d'abord par calculer les maxima locaux dans une autre image binaire, où les maxima locaux auront la valeur 1. La seconde partie de l'algorithme est assez proche de celui des cônes conservatifs. Pour chaque niveau de profondeur des maxima, une image binaire est construite où seuls les maxima locaux inférieurs ou égaux au niveau de profondeur courant auront la valeur 0. Tous les autres texels auront la valeur 1. Ensuite nous calculons l'EDT de cette image que nous utiliserons pour calculer le ratio (ou le rayon). Ce dernier ne sera modifié que si la nouvelle valeur est plus petite que celle déjà sauvegardée (Voir l'Algorithme 3.6).

Il faut noter que lorsque la carte de profondeurs contient des régions avec une profondeur uniforme, l'algorithme quadratique peut donner des cônes plus larges. Comme nous pouvons le

constater sur la figure 3.7, l'algorithme linéaire a tendance à prendre le premier maximum *q*, alors que l'algorithme quadratique prend le dernier *r*. Mais cela n'est pas toujours le cas, car le plus souvent, ce dernier prendra le point tangentiel le plus proche de la région uniforme. Par contre, l'algorithme linéaire retiendra toujours le point le plus proche de cette région.

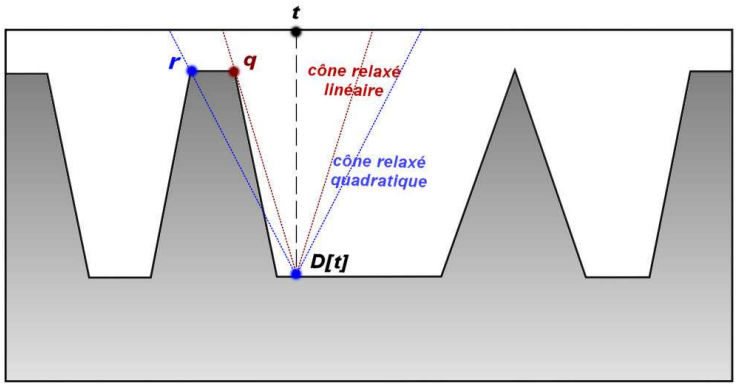


Fig. 3.7 - Comparaison entre l'algorithme linéaire et l'algorithme quadratique pour le calcul des cartes de cônes relaxés.

Algorithme 3.6 - Cônes Relaxés (Linéaire)

```
Entrée       : Carte de profondeurs D
Sortie       : Carte de cônes C
Intermédiaire : Carte binaire M   /* maxima locaux */
                Carte binaire B, carte de distances T

M = MAXIMA_LOCAUX( D ) /* si maximum 1 sinon 0 */
pour chaque niveau l of D ∩ M faire  /* niveau des maxima */
  pour chaque texel t faire
    B[t] = {0 si D[t] ≤ l et M[t]==1, 1 sinon}
  fin pour t
  T = EDT( B )    /* distance des 1 aux 0 */
  pour chaque texel t faire
    si D[t] > l alors
      C[t] = min( T[t]/(D[t]-l), C[t] )
    sinon
      C[t] = VALEUR_MAX
    fin si
  fin pour t
fin pour l
```

3.3.3 Répétition des cartes de cônes

Si la carte de déplacement devra à être plaquée par répétition, nous devons prendre en considération le type de cette répétition pendant le prétraitement. Dans les algorithmes quadratiques, nous pouvons accéder aux texels qui sont hors de la texture (coordonnées négatives par exemple) avec des fonctions dites *miroir* ou *décalage*. Ce qui permet alors de générer des cartes de déplacement adaptées au type de répétition souhaité. Mais dans le cas des algorithmes linéaires, qui utilisent la transformation distance, le traitement direct ne permet d'avoir qu'une version à répétition miroir. Afin de générer la version par décalage, nous utilisons, dans un processus intermédiaire, une carte de déplacement faisant le double de l'originale, et qui sera créée en plaçant la carte originale au centre, et en remplissant les zones restantes par décalage. Ensuite, nous calculons la carte de cônes correspondante, et finalement nous extrayons le centre de cette dernière. Ce processus est schématisé sur la figure 3.8.

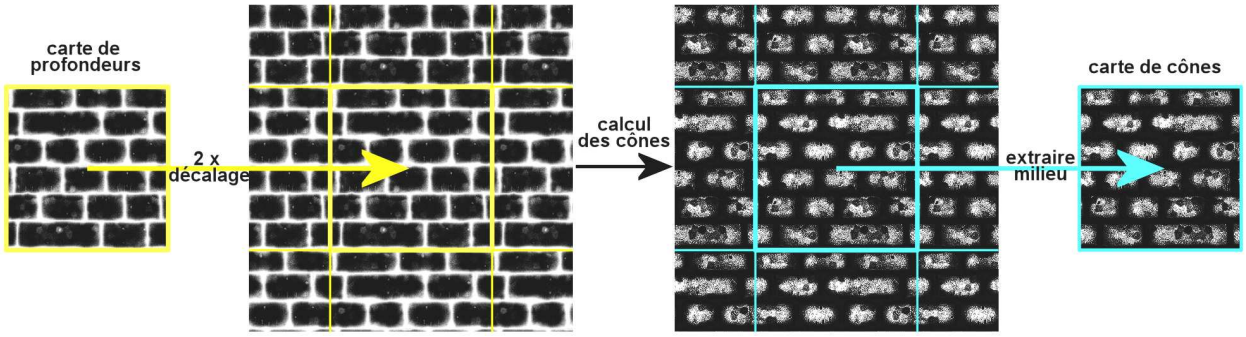


Fig. 3.8 - Calcul de la carte de cônes permettant une répétition avec décalage.

3.4 Résultats

Nous avons implémenté les différents algorithmes présentés dans ce chapitre en C++. Pour les programmes de rendu, nous avons utilisé OpenGL et GLSL. Les mesures et les figures ont été réalisées à l'aide d'un PENTIUM IV, avec 512Mb de RAM, et une carte graphique GeForce 7600GS AGP avec 256Mb de mémoire vidéo.

3.4.1 Prétraitement

Les temps d'exécution des quatre algorithmes décrits dans ce chapitre sont présentés dans le tableau 3.1. La figure 3.9 représente le graphique d'une partie des données du tableau. Comme prévu, les résultats révèlent bien le caractère quadratique ou linéaire des algorithmes. Nous

pouvons d'ailleurs vérifier que dans le cas linéaire, le rapport entre le temps d'exécution d'une texture avec une dimension donnée, et celle avec la dimension inférieure, est égal à 4, car le nombre de texels est à chaque fois multiplié par 4.

		cônes conservatifs		**cônes relaxés**	
		Linéaire	**quadratique**	**linéaire**	**quadratique**
256x256	***a***	2 (9)	1 (2)	1 (4)	30640
	b	2 (6)	3 (4)	2 (7)	30528
512x512	***a***	10 (35)	14 (21)	8 (26)	490240 *
	b	7 (26)	38 (52)	8 (28)	488440 *
1024x1024	***a***	39 (140)	194 (294)	42 (149)	
	b	28 (101)	568 (792)	31 (110)	
2048x2048	***a***	154 (554)	4597 (5978)	171 (600)	
	b	110 (400)	9088 (12672)	123 (442)	

Tab. 3.1 - Temps d'exécution du prétraitement des cartes de cônes en stockant les ratios (en secondes). Nous avons utilisé deux cartes de profondeurs *a* et *b* avec plusieurs résolutions (voir la figure 3.9). Les valeurs entre parenthèses sont les temps d'exécution des versions pour la répétition avec décalage. L'étoile désigne que la valeur a été seulement estimée.

Concernant les algorithmes conservatifs, le tableau 3.1 montre que dans le cas de la texture *a*, la vitesse de l'algorithme quadratique reste raisonnable jusqu'à la dimension 1024x1024. Elle est même supérieure à celle de la version linéaire avec décalage, mais uniquement pour les dimensions inférieures à 512x512. Mais pour la texture *b*, qui contient un relief très espacé, l'algorithme quadratique est nettement plus lent dés la dimension 256x256.

Pour les algorithmes des cônes relaxés, l'algorithme linéaire est d'une vitesse comparable à celle des conservatifs. Par contre l'algorithme quadratique est extrêmement lent. Nous nous somme contenter alors de calculer le temps d'exécution des cartes de taille 256x256, cela a pris tout de même 8h ! Pour cette raison, Policarpo et Oliveira [PO07] ont utilisé un prétraitement basé sur le GPU afin de réduire le temps de calcul.

Il est à noter que, concernant le calcul et le stockage du rayon des cônes au lieu de leur ratio, les temps d'exécution sont quasiment les mêmes, excepté dans le cas quadratique de l'algorithme des cônes conservatifs, dont la rapidité est nettement inférieure. Cela est du au fait que l'angle de la pente des cônes n'est plus limitée à $\pi/4$, ce qui affecte directement l'optimisation utilisée dans cet algorithme.

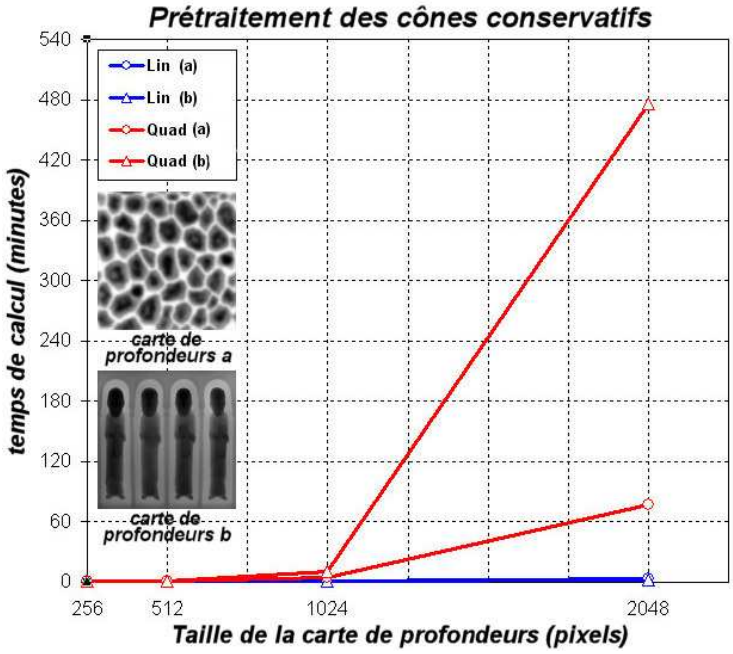


Fig. 3.9 - Graphique des temps d'exécution des algorithmes des cônes conservatifs.

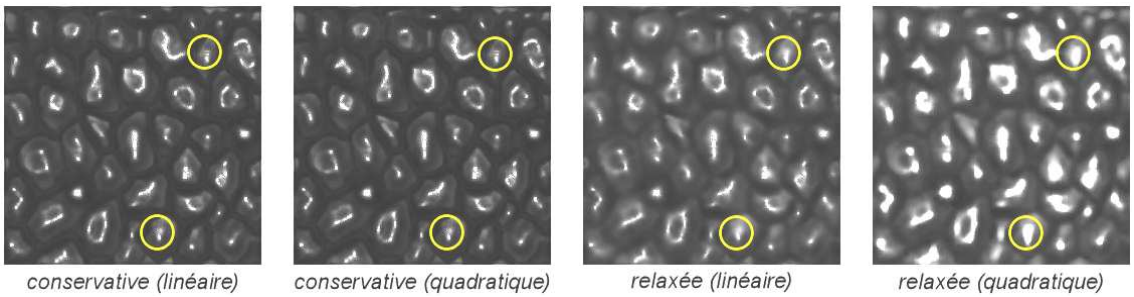


Fig. 3.10 - Les cartes de cônes résultantes des quatre algorithmes décrits dans ce chapitre.

La figure 3.10 montre des cartes de cônes stockant des ratios. Nous pouvons y remarquer que dans le cas linéaire, les algorithmes conservatifs produisent exactement la même carte. Dans le cas des algorithmes relaxés, les deux produisent des cartes avec des cônes plus larges (c-à-d avec des zones plus claires). Mais nous pouvons clairement constater que l'algorithme quadratique produit les cônes les plus larges. Cela est du à l'approximation utilisée par cet algorithme, et qui entraîne des défauts très visibles (voir la figure 3.11), même si la recherche binaire tend à rectifier une grande partie de ces défauts.

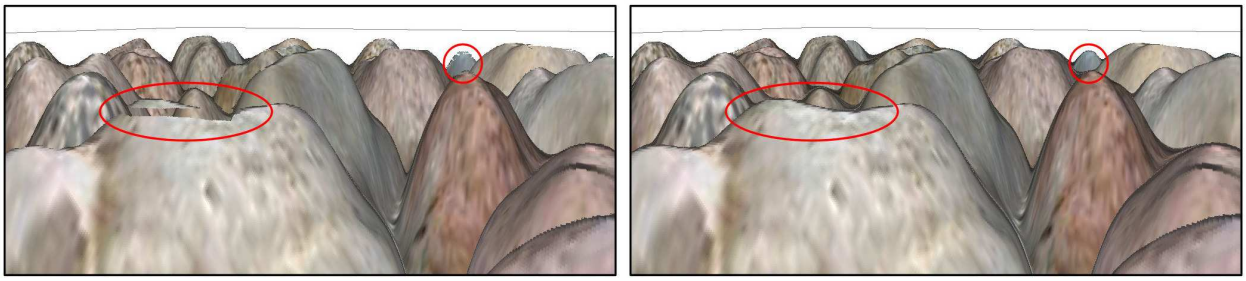


Fig. 3.11 - Comparaison entre l'algorithme quadratique (à gauche) et l'algorithme linéaire (à droite) pour le calcul des cartes de cônes relaxés.

3.4.2 Rendu

Afin de comparer les performances du rendu des différentes techniques, nous avons utilisé 20 étapes pour le traçage de cônes, et 10 étapes pour la recherche binaire utilisée dans le cas des cônes relaxés. La figure 3.12 montre des captures d'écran des quatre techniques. Nous pouvons observer que l'utilisation des rayons au lieu des ratios entraîne une meilleure convergence, sans pour autant ralentir les performances (115 IPS). Nous observons également que l'utilisation des cônes relaxés donne des résultats nettement meilleurs, mais avec le coût de la recherche binaire (94 IPS). Par conséquent, l'exécution de l'algorithme des cônes conservatifs avec un nombre d'étapes plus important, produira quasiment les mêmes résultats que ceux de l'approche relaxée, et avec la même vitesse approximativement.

La figure 3.13 met en évidence l'utilisation des textures non-carrées. Comme nous l'avons schématisé sur la figure 3.3, nous pouvons observer que la normalisation de la carte de profondeurs avant le prétraitement des cônes, ne donne des résultats légèrement meilleurs que lorsque le rayon de vue est presque parallèle à la direction de la plus grande dimension. Par contre, dans toutes les autres directions, la méthode de rectification elliptique est très nettement meilleure.

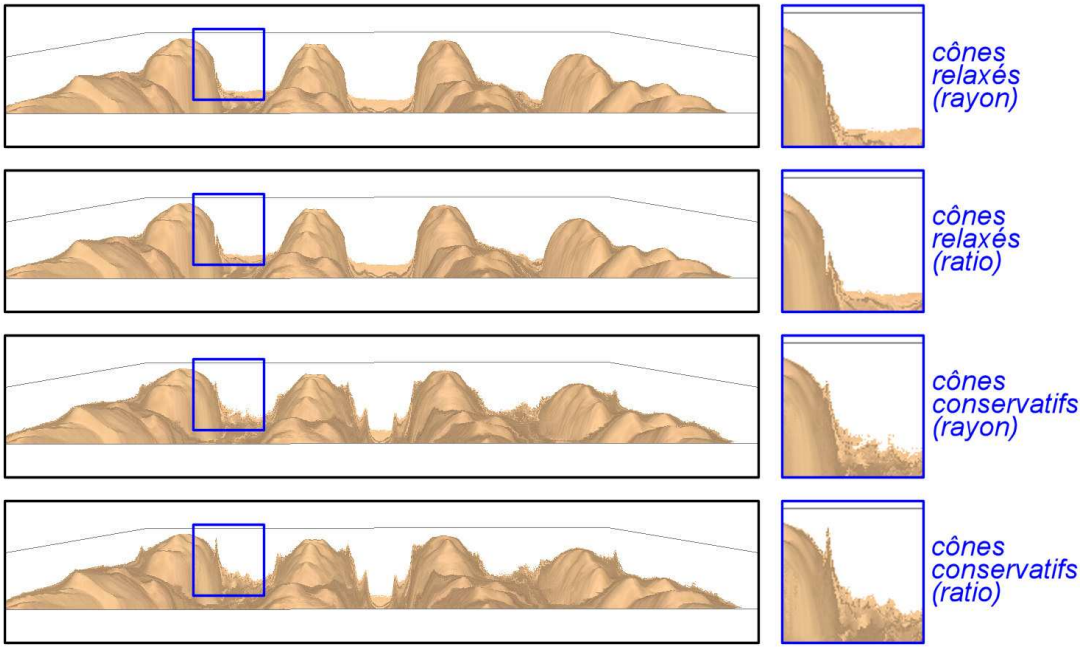


Fig. 3.12 - Comparaison des quatre techniques de traçage de cônes. Nous pouvons clairement constater que la qualité du rendu augmente du bas vers le haut.

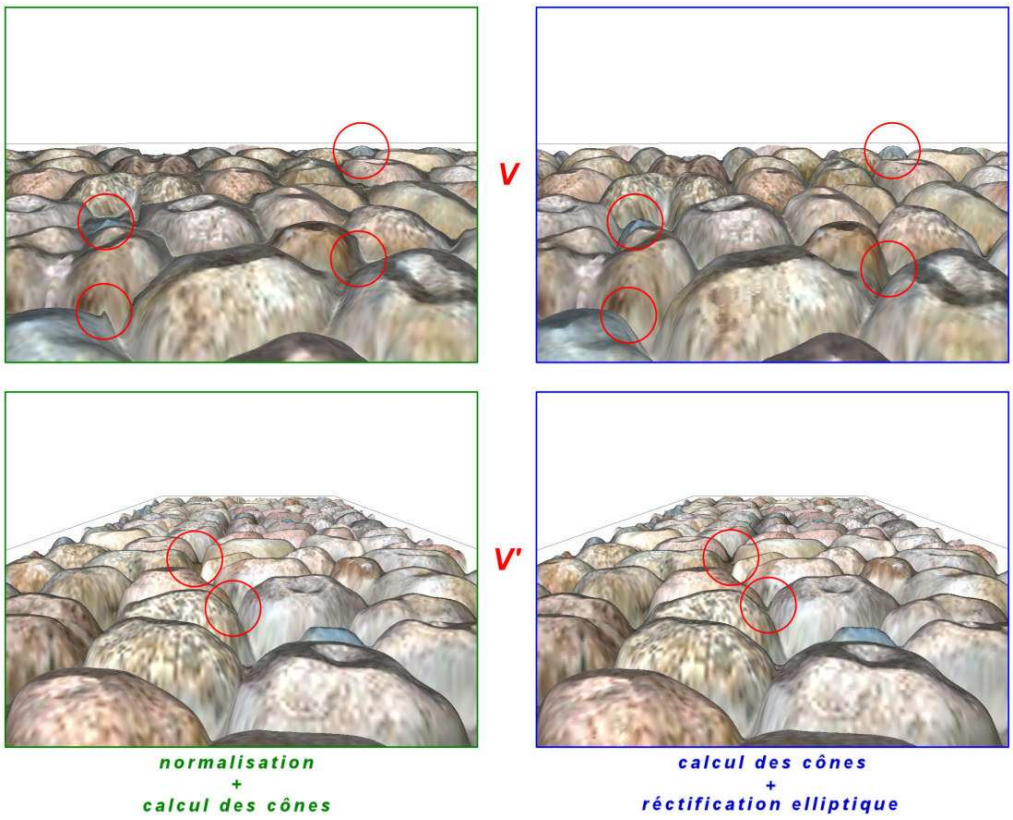


Fig. 3.13 - Comparaison entre les deux approches pour le prétraitement des cartes de profondeurs rectangulaires.

3.5 Conclusion

Dans ce chapitre, nous avons introduit un nouvel algorithme à complexité $O(n)$ pour le calcul des cartes de cônes relaxés. Nous avons également suggéré d'utiliser l'algorithme linéaire HDDT pour générer les cartes de cônes conservatifs. Les cartes de cônes sont utilisées par les techniques de traçage de cônes pendant la phase de rendu. Nous avons également proposé une méthode pour l'extension de ces techniques aux textures rectangulaires. Enfin, nous avons proposé d'utiliser le rayon des cônes au lieu de leur ratio, ce choix permet de stocker des cônes avec un angle de pente allant jusqu'à $\pi/2$, améliorant ainsi la convergence et les résultats du traçage de cônes.

Nous somme convaincus que le passage vers un prétraitement avec complexité linéaire favorisera encore plus la technique de traçage de cône, qui reste l'une des meilleures approches pour la représentation des mesostructures en temps réel. Il faut noter que même si la vitesse du prétraitement n'affecte pas les performances pendant la phase de rendu, elle n'en demeure pas moins importante, notamment dans le cas d'applications interactives, ou dans le cas où la carte de profondeurs doit être mise à jour régulièrement.

Chapitre 4

Placage d'Extrusion et de Révolution

Dans ce chapitre, nous introduisons une nouvelle technique de placage de déplacement 3D par pixel permettant, entre autres, l'affichage de motifs extrudés ou biseautés. Nous proposons également une nouvelle méthode de modélisation et rendu à base d'images utilisée pour créer des modèles 3D sans maillage, obtenus à l'aide de l'extrusion ou de la révolution d'une forme 2D conservée sous forme d'une image binaire. Dans la première partie de ce chapitre, nous présenterons l'étape du prétraitement de la carte de forme qui sera utilisée pendant la phase du rendu. Dans la partie suivante, nous détaillerons la technique du placage d'extrusion et des techniques qui lui sont associées (*extension, biseau, chanfrein*)*. La section suivante sera consacrée à la technique du placage de révolution. Nous évoquerons par la suite la création des boites de forme pour la représentation de modèles 3D complets. Finalement, nous présenterons les résultats obtenus concernant les deux phases: prétraitement et rendu.*

4.1 Introduction

Le rendu des scènes complexes, comportant des surfaces très détaillées, reste l'un des problèmes majeurs pour le rendu temps réel. La technique traditionnelle consiste à utiliser des maillages à haute définition comportant un très grand nombre de sommets et de polygones. Cependant, une scène complexe peut contenir des millions de primitives graphiques, rendant du coup son affichage impossible pour un rendu interactif, même pour un matériel graphique récent. Pour pallier à ce problème, les infographistes créent des environnements moins détaillés et comportant beaucoup moins d'objets. Or, même en utilisant des techniques comme le placage de textures ou le placage de bosselures, afin d'augmenter le réalisme de ces scènes, la plupart des surfaces paraissent plates, et la polygonisation reste clairement visible sur la silhouette des objets.

L'architecture des nouvelles cartes graphiques, disposant d'un pipeline programmable, a permis d'introduire une alternative au rendu à base de polygones. En effet, les techniques de modélisation et rendu à base d'images ou IBMR (*Image Based Modeling and Rendering*) s'appuient sur des textures qui servent à stocker des données relatives à la géométrie. Cette géométrie est restituée par la suite, à l'aide d'un algorithme simplifié de type *traçage de rayons* qui s'exécute dans les unités programmables du processeur graphique. Les techniques IBMR évitent ainsi le traitement d'un grand nombre de sommets et de polygones. De plus, elles opèrent dans l'espace image, et de ce fait, ne traitent que les pixels qui seront visibles à l'écran, contrairement au rendu à base de polygones qui utilise des algorithmes de détermination de visibilité, et qui sont eux-mêmes très coûteux en terme de ressources système.

Ce chapitre introduit une nouvelle technique pour le rendu de surfaces comportant des détails extrudés, biseautés ou chanfreinés. Il introduit également une technique pour la modélisation et le rendu d'objets entiers formés par une extrusion ou une révolution, à l'aide d'une simple boite et d'une seule texture. Les techniques proposées se basent sur le traçage de rayons, et utilisent la transformation distance Euclidienne pour accélérer la recherche d'intersection.

Le placage d'extrusion est la technique la plus rapide pour le rendu de motifs extrudés. Elle donne également les meilleurs résultats sans présenter les défauts liés à une grande échelle de profondeur, ou aux angles de vue rasant la surface. Le placage de révolution génère des surfaces de révolution très convaincantes sans effet de polygonisation, et qui sont affichées d'une manière interactive avec les cartes graphiques actuelles. Les techniques que nous proposons peuvent

limiter considérablement le nombre de primitives graphiques constituant les scènes complexes, et qui sont généralement composées d'environnements architecturaux et d'objets manufacturés créés à l'aide d'extrusion et de révolution.

Contrairement aux techniques classiques pour le placage de déplacement par pixel, qui se basent sur une image à niveaux de gris représentant un relief quelconque, notre approche est plus ciblée car elle vise la représentation de motifs et d'objets créés à partir d'extrusion, de biseau, de chanfrein ou de révolution. Par conséquent, les techniques proposées utilisent une image binaire, où seuls les pixels à valeur zéro constituent la forme de base qui va générer la géométrie par extrusion ou par révolution. L'algorithme est de type traçage de rayons, il s'appuie sur la recherche d'intersection entre l'axe de vue et la géométrie virtuelle. Une carte de distance 2D est utilisée pour coder l'espace vide, et accélérer ainsi la recherche de l'intersection.

L'algorithme est destiné principalement au rendu temps réel, il sera donc implémenté via les unités programmables de sommets et de pixels, afin de bénéficier au maximum de la puissance du GPU. Cependant, l'approche peut également être adaptée au rendu différé.

Les contributions apportées dans ce chapitre sont les suivantes:

- Une nouvelle technique de placage de déplacement par pixel, utilisé pour le rendu temps réel des surfaces présentant des motifs extrudés, biseauté ou avec chanfrein, en s'appuyant sur une forme 2D binaire.
- Une nouvelle technique IBMR pour la création et le rendu d'objets complets, générés à partir d'extrusion ou de révolution d'une forme de base.

4.2 Prétraitement : la carte de forme

La carte de forme est une texture RVBA qui contient les données nécessaires aux algorithmes de placage d'extrusion et de révolution (voir la figure 4.1). Le canal *alpha* sert à stocker la forme de base représentée par une image binaire où seuls les pixels à valeur zéro sont considérés comme faisant partie de la forme. Le canal *bleu* est utilisé pour stocker la carte de distance calculée à partir du canal *alpha*. Enfin, les canaux *rouge* et *vert* abritent respectivement les composantes *Gx* et *Gy* du gradient unitaire de la carte de distance. Le gradient définit, en chaque pixel, la normale à la forme de base, ou plus généralement, la normale aux lignes de niveaux de la carte de distance. Pour le calcul du gradient, nous avons utilisé le filtre de Prewitt avec différentes tailles.

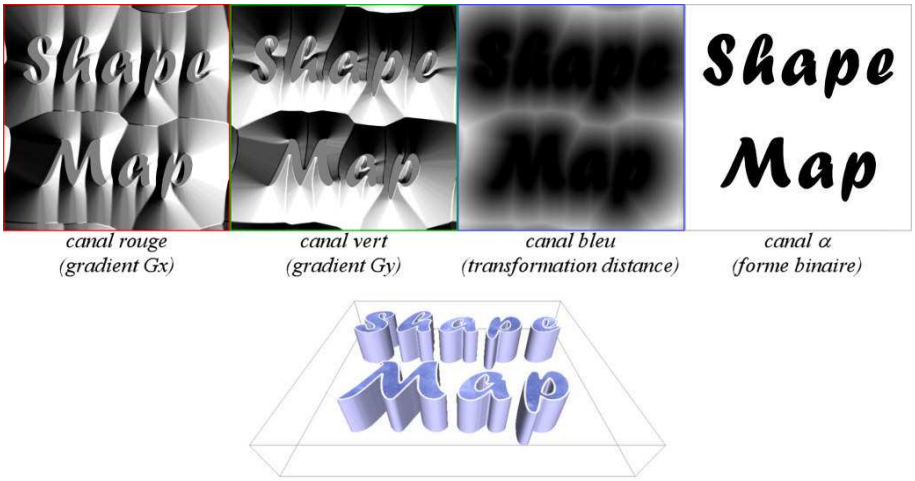


Fig. 4.1 - Présentation de la carte de forme. La forme binaire est stockée dans le canal *alpha*. Sa transformation distance est conservée dans le canal *bleu*. Tandis que les canaux *rouge* et *vert* sont utilisés pour stocker les composantes *Gx* et *Gy* du gradient unitaire de la transformation distance. (En bas) Rendu d'une boite avec placage d'extrusion en utilisant la carte de forme du haut

4.2.1 Transformation Distance Euclidienne

Pour effectuer la recherche du point d'intersection de l'axe de vue avec la géométrie virtuelle, nous utilisons la transformation distance Euclidienne EDT (*Euclidien Distance Transform*) décrite dans [Dan80], appliquée à l'image binaire représentant la forme de base (voir l'algorithme 3.4 du chapitre 3). Cet algorithme a l'avantage d'être très rapide, tout en donnant une très bonne approximation de la distance. L'utilisation de la distance, au lieu d'une recherche linéaire à intervalles réguliers, a l'avantage de converger très rapidement, et permet notamment d'éviter d'omettre des intersections à angles de vue rasants.

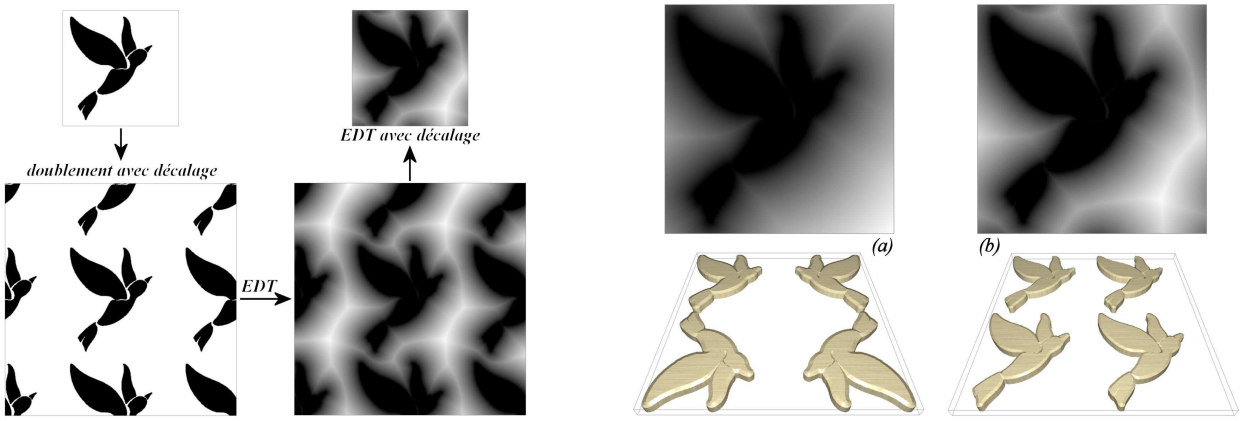


Fig. 4.2 - Répétition de la carte de forme. (A gauche) Principe du calcul de l'EDT pour la répétition en mode décalage. (A droite) Comparaison entre les deux versions pour le calcul de l'EDT. (a) Le calcul direct et rapide qui permet d'avoir une carte de distance sans répétition ou avec une répétition en miroir. (b) Le calcul par l'intermédiaire d'une image doublée permet une répétition avec décalage.

Le placage d'extrusion et de révolution n'utilise qu'une seule texture 2D, et la distance n'occupe qu'un seul canal. Contrairement aux techniques de rendu d'un relief arbitraire, décrites dans [Don04, Rit06], et qui s'appuient sur des cartes de distance 3D nécessitant un temps de prétraitement conséquent et un espace mémoire très important.

Puisque nous allons stocker l'EDT dans un canal au format entier 8 bits, nous devons étaler les distances dans l'intervalle [0,1] afin d'avoir une distribution optimale. Nous devons alors, pour chaque pixel *p*, moduler les distances en fonction de la valeur maximale de l'EDT :

$$S_b(p) = \frac{EDT(p)}{\max(EDT)}$$

où S_b représente le canal bleu de la carte de forme. Pendant la phase de rendu, nous développons la distance à l'aide de la formule suivante :

$$EDT(p) = \tau \cdot S_b(p) \quad avec: \frac{n}{\max(EDT)}$$

avec *n* la largeur de la carte de forme. Dans notre implémentation, nous utilisons le nom de la carte de forme pour sauvegarder le facteur d'échelle τ.

Dans le cas où la forme plaquée doit être répétée plusieurs fois, l'EDT de l'image binaire d'entrée ne permet d'avoir qu'une répétition avec symétrie (miroir). Pour la prise en compte de la répétition avec décalage, il faudra calculer l'EDT d'une autre image, faisant le double de la taille de l'image d'origine. Cette dernière sera placée au centre, tandis que les autres parties seront obtenues par décalage. Une fois l'EDT calculée, la partie centrale est extraite pour donner la carte de distance finale. La figure 4.2 montre la différence entre cette approche et le calcul directe.

4.2.2 Gradient unitaire de la carte de distance

L'étape du calcul d'ombrage, qui succède à la recherche de l'intersection, nécessite la normale à la géométrie au point de l'intersection. En général, cette normale est calculée à partir des composantes *Gx* et *Gy* du gradient unitaire de la forme de base. Cependant, dans certains cas, le point d'intersection ne sera pas atteint exactement. Par conséquent, il ne se trouvera pas à proximité du contour de la forme où le gradient est non nul. Ce problème se posera également dans le cas du placage d'extrusion étendue, du placage de biseau et du placage de chanfrein, où le contour de la forme de base ne sera jamais atteint. Afin de surmonter ce problème, nous calculons le gradient unitaire de la carte de distance au lieu de la forme de base, car les lignes de

niveau de la distance représentent des prolongements du contour de la forme sur toutes les parties de l'image (voir la figure 4.3). Le gradient peut être calculée parallèlement à l'EDT, car le calcul de cette dernière s'appuie implicitement sur le gradient. Cependant, nous préférons utiliser une convolution par un filtre gradient, dont nous pouvons varier la taille, afin de limiter certains défauts de pixellisation.

Fig. 4.3 - Gradient unitaire de la carte de forme. (A gauche) Mise en évidence des lignes de niveau de la distance Euclidienne (canal *bleu*). (Au milieu) Gradient unitaire de la forme binaire. (A droite) Le gradient unitaire de la carte de distance que nous avons retenu.

4.3 Placage d'extrusion

Le placage d'extrusion consiste à créer une géométrie virtuelle au-dessous d'un model polygonal. Cette géométrie représente l'extrusion de la forme plaquée sur le polygone en utilisant les traditionnelles coordonnées de texture. Pour chaque pixel projeté, nous devons calculer l'intersection de l'axe de vue avec la géométrie de l'extrusion. Le calcul d'intersection est effectué dans l'espace tangent associé au pixel courant.

Les vecteurs de vue et de lumière sont transformés de l'espace global vers l'espace tangent pour chaque sommet, puis sont interpolés pour chaque pixel pendant la rastérisation. La profondeur de l'extrusion est fixée par l'utilisateur, et peut varier en temps réel.

4.3.1 Placage d'extrusion basique

Soit (u,v) les coordonnées du pixel courant et p_0 le point de départ de la recherche avec les coordonnées $(x_0, y_0, z_0)=(u,v,0)$. Et soit $\vec{W}$ la direction de vue, calculée en normalisant le vecteur allant du point de vue vers la position 3D de p_0, et en prenant l'opposé[1] de sa composante z. Afin de converger rapidement vers le premier point d'intersection p_j, nous utilisons l'EDT pour

[1] Nous utilisons l'opposé de z car bien que la normale de l'espace TBN soit dirigée vers le haut, la profondeur croit vers le bas entre 0 à 1

avancer d'une manière fiable et rapide le long de l'axe de vue. L'algorithme 4.1 détaille cette technique.

4.3.1.1 Calcul du point d'intersection

Tout d'abord, nous contrôlons l'échelle de profondeur en divisant la coordonnée *z* de $\vec{W}$ par la profondeur maximale κ, puis nous re-normalisons le vecteur ainsi obtenu :

$$\vec{V} = \left(W_x, W_y, \frac{W_z}{\kappa}\right) \Big/ \left\| \left(W_x, W_y, \frac{W_z}{\kappa}\right) \right\|$$

Comme il est schématisé sur la figure 4.4, à chaque étape, la distance d_i, entre le point p_i et la forme extrudée, est extraite du canal *bleu* de la carte de forme aux coordonnées (x_i, y_i). Ensuite, à partir la relation de similitude suivante :

$$\frac{p_{i+1} - p_i}{d_i} = \frac{\vec{V}}{\left\|\vec{V}_{xy}\right\|}$$

nous déduisons la position suivante p_{i+1} sur l'axe de vue :

$$p_{i+1} = p_i + \frac{d_i}{\left\|\vec{V}_{xy}\right\|} \cdot \vec{V}$$

Si la carte de forme n'est pas carrée, nous devons utiliser la rectification elliptique (voir la section 3.2.4 du chapitre 3), en multipliant la distance d_i par le facteur correcteur suivant:

$$r = \frac{\left\|\vec{V}_{xy}\right\|}{\sqrt{V_x^2 + \rho^2 V_y^2}}$$

Avec ρ=*hauteur/largeur*, l'expression de p_{i+1} devient alors :

$$p_{i+1} = p_i + \frac{d_i}{\sqrt{V_x^2 + \rho^2 V_y^2}} \cdot \vec{V}$$

Puisque la rectification ne dépend que du rayon de vue $\vec{V}$, nous pouvons utiliser l'optimisation suivante :

$$p_{i+1} = p_i + d_i \cdot \vec{V}_\rho \qquad \text{avec :} \quad \vec{V}_\rho = \frac{1}{\sqrt{V_x^2 + \rho^2 V_y^2}} \cdot \vec{V} \tag{4.1}$$

La répétition de la carte de forme est effectuée en multipliant les coordonnées du pixel courant (*u*,*v*) par le nombre de répliques suivant *x* et *y :*

$$p_0 = (T_x \cdot u,\ T_y \cdot v,\ 0)$$

Après la dernière étape p_j, nous effectuons un fenêtrage pour éliminer les points qui ne font pas partie de la géométrie d'extrusion. Le point p_j sera éliminé si l'une des relations suivantes est vraie: $x_j<0$, $x_j>T_x$, $y_j<0$, $y_j>T_y$ et $z_j>1$.

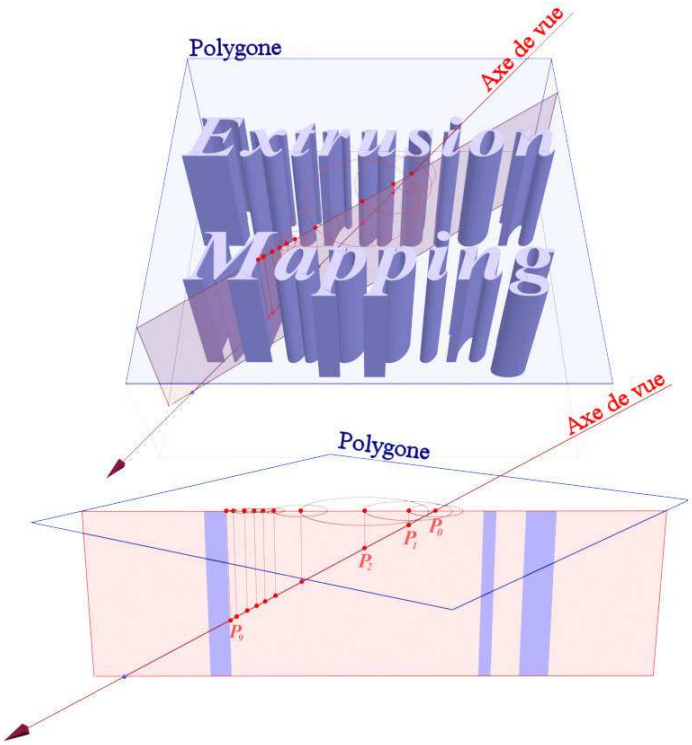


Fig. 4.4 - La recherche d'intersection entre le rayon de vue et l'extrusion générée à partir de la carte de forme. (En haut) Vue 3D de la procédure. (En bas) une coupe incluant l'axe de vue permet d'observer nettement la convergence vers le point d'intersection.

4.3.1.2 Calcul de la normale

Afin de pouvoir procéder au calcul d'ombrage du pixel traité, nous devons calculer la normale à la géométrie d'extrusion au point de l'intersection $p_j = (x_j, y_j, z_j)$. Nous récupérons d'abord les composantes du gradient unitaire (Gx_j, Gy_j) qui sont stockées respectivement dans les canaux *rouge* et *vert* de la carte de forme à la position (x_j, y_j). Ces deux composantes doivent par la suite être étalées sur l'intervalle [-1,1]. Comme la normale est perpendiculaire à la géométrie de l'extrusion, sa coordonnée *z* sera nulle. La normale est donc définie par :

$$\vec{N}_j = (Gx_j, Gy_j, 0)$$

excepté sur la surface du polygone (c-à-d $p_j=p_0$) où la normale sera égale à celle du polygone.

Dans le cas où la carte de forme est plaquée avec répétition en miroir, l'orientation de la normale changera en fonction de l'imparité de la partie entière des coordonnées suivant *x* et *y* du point de l'intersection p_j. La normale sera dans ce cas définie par :

$$\vec{N}_j = (S_x \cdot Gx_j, S_y \cdot Gy_j, 0)$$

avec:

$$S_x = \begin{cases} +1 & si\ E(x_j)\ est\ pair \\ -1 & sinon \end{cases} \quad et \quad S_y = \begin{cases} +1 & si\ E(y_j)\ est\ pair \\ -1 & sinon \end{cases}$$

Algorithme 4.1 - Placage d'Extrusion (PE)

```
Entrée :
W, L, Nf       : Rayon de vue normé et mis en échelle, rayon de lumière normé,
                 Normale du polygone. Tous exprimés dans l'espace TBN.
(u,v),(Tx,Ty),κ: Pixel courant, nombre de répétition sur x et y, échelle de profondeur
C, F, ρ, τ     : carte de couleur, carte de forme, son rapport, et l'échelle de distance
Ka, Kd, Ks, n  : Composantes ambiante, diffuse, spéculaire, et lustre
Sortie :
(r,v,b)        : couleur finale du pixel courant

// Initialisation
p = (Tx*u, Ty*v, 0)                  // point de départ
V = (W.x,W.y,-W.z/κ), V=V/norme(V)    // mise à l'échelle et inversion du sens suivant z
Vρ= τ*V/(Vx*Vx+ ρ*ρ*Vy*Vy)           // rectification elliptique et modulation de la distance
// recherche de l'Intersection
pour i allant de 1 à NB_ETAPES faire
       d = F[p.x,p.y].bleu   // la distance est récupérée à partir du canal bleu
       p = p + d * Vρ
fin pour
// fenêtrage
si p.z>1 ou p.x<0 ou p.x>Tx ou p.y<0 ou p.y>Ty alors
       annuler               // pas d'intersection
fin si
// calculer la normale au point (p.x,p.y)
si p.z == 0 alors
       N = Nf
sinon
       N = (2*F[p.x,p.y].rouge-1, 2*F[p.x,p.y].vert-1, 0)
       si répétition_miroir alors
              N.x = -N.x si E(p.x) est impair     // E() est la partie entière
              N.y = -N.y si E(p.y) est impair
       fin si
fin si
// calcul de l'ombrage (Blinn-Phong)
(ξ,η)   = TexCoord(p)        // fonction qui calcule les coordonnées de textures
(r,v,b) = C[ξ,η]             // récupérer la couleur de la texture
H = (L+V)/norme(L+V)
(r,v,b) = Ka*(r,v,b) + Kd*(r,v,b)*max(0,N.L) + ks*max(0, (H.N)^n )
```

4.3.2 Placage d'extrusion étendue

Comme les lignes de niveau de la carte de distance constituent des formes étendues de la forme de base, nous pouvons dès lors les utiliser pour créer l'extrusion. Il suffit pour cela de remplacer la distance dans l'algorithme de l'extrusion basique par la distance à la forme étendue (Algorithme 4.2). La formule (4.1) devient :

$$p_{i+1} = p_i + \max(0, d_i - e) \cdot \vec{V}_\rho \qquad (4.2)$$

où *e* est un paramètre modifiable en temps réel, permettant de moduler l'effet de l'extension.

L'extrusion étendue est légèrement moins rapide que l'extrusion de la forme d'origine. Cependant, elle peut être très utile pour lisser les formes comme le montre la figure 4.5.

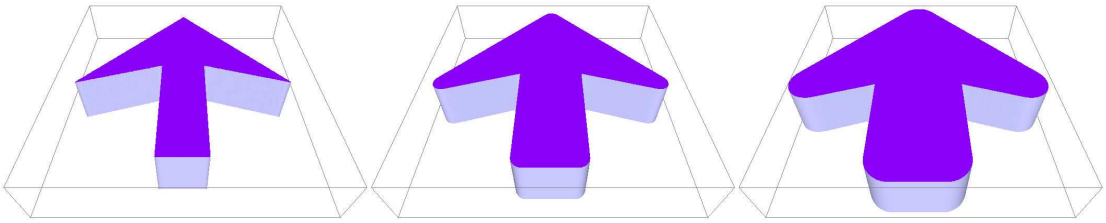


Fig. 4.5 - Rendu d'une forme de flèche avec le placage d'extrusion étendue. Augmenter la valeur de l'extension même en temps réel (de gauche à droite) permet d'avoir des extrusions plus adoucies.

Algorithme 4.2 - Placage d'Extrusion Etendue (PEE)

```
Entrée :
...             : Données identiques à celles du Placage d'Extrusion (Algorithme 4.1)
e               : Composante de l'extension
Sortie :
(r,v,b)         : couleur finale du pixel courant

// Initialisation
...     // partie identique à l'algorithme 4.1
pour i allant de 1 à NB_ETAPES faire
        d = F[p.x,p.y].bleu   // la distance est récupérée à partir du canal bleu
        p = p + max(0, d-e) * Vρ
fin pour
...     // partie identique à l'algorithme 4.1
```

4.3.3 Placage d'extrusion biseautée

Le biseau consiste à créer une extrusion étendue vers l'extérieur qui varie en fonction de la profondeur (voir la figure 4.6). Nous commençons par faire une recherche de l'intersection en se basant sur la distance à la forme d'origine avec la formule (4.2) (car nous combinons le biseau avec l'extrusion étendue). Pendant cette recherche, il faut tester si la position actuelle est à l'intérieur de la géométrie en utilisons la différence suivante :

$$\begin{aligned} \Delta d_i &= \max(0, d_i - e) - b \cdot z_i \\ \Delta d_i &= \max(0, d_i - e) - \delta_i \end{aligned} \tag{4.3}$$

où b est un paramètre permettant de contrôler l'effet du biseau. δ_i désigne la largeur du biseau au niveau de z_i (voir la figure 4.7).

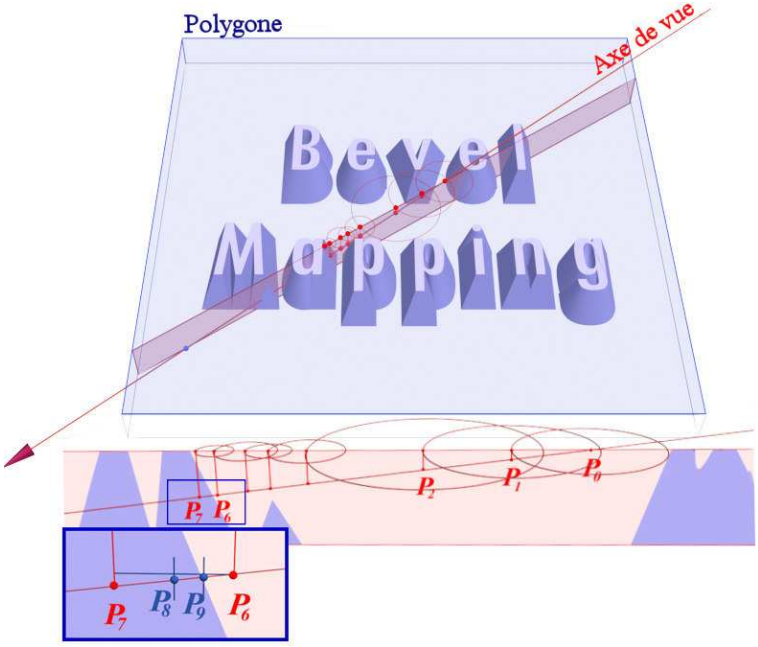


Fig. 4.6 - Intersection du rayon de vue avec la surface biseautée.

Si Δd_i est positive, nous calculons la position suivante p_{i+1} à l'aide de l'équation (4.2), car le point p_i de l'étape courante est toujours à l'extérieur de la géométrie. Si Δd_i est négative, nous obtenons le point p_j qui se trouve à l'intérieur de l'extrusion. Nous passons alors à la phase de l'affinage binaire, en divisant successivement par 2 la dernière distance $\max(0, d_{j-1} - e)$ qui a amené le point à l'intérieur de la géométrie. Nous gardons la même formule (4.3) pour le test au-dessus/au-dessous (figure 4.7).

La coordonnée *z* de la normale n'est plus nulle dans le cas du biseau. Cependant, elle reste uniforme et s'obtient par rotation du gradient. Elle vaut la valeur du biseau *b* comme le montre la figure 4.7. Elle doit être ensuite normée, avant de procéder au calcul d'ombrage. Sa formule est alors la suivante :

$$\vec{N}_j = \frac{(Gx_j, Gy_j, b)}{\left\|(Gx_j, Gy_j, b)\right\|}$$

où (Gx_j, Gy_j) représente les coordonnées du gradient unitaire au point (x_j, y_j) de la carte de forme (voir l'algorithme 4.3 pour plus de détails).

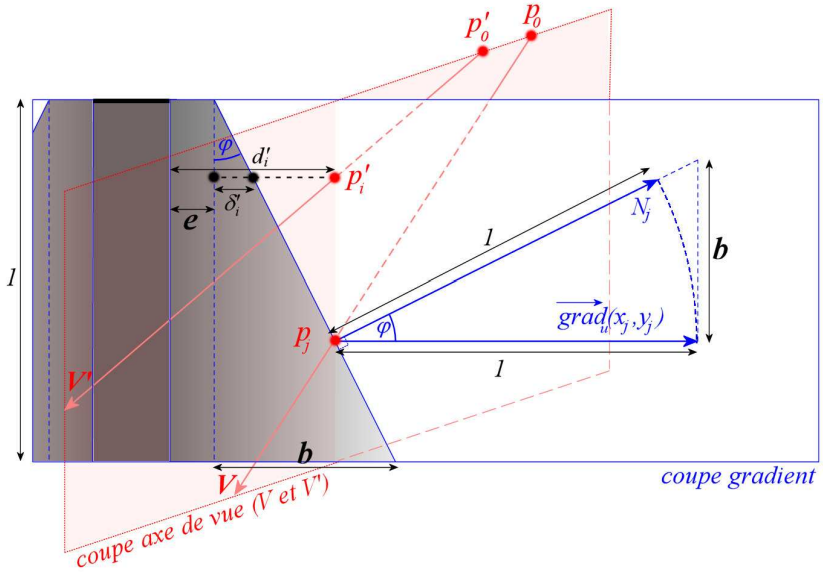


Fig. 4.7 - Calcul de la normale dans le cas de l'extrusion biseautée.

Algorithme 4.3 - Placage d'Extrusion Biseautée (PEB)

```
Entrée :
...            : Données identiques à celles du Placage d'Extrusion (Algorithme 4.1)
e, b           : Composantes de l'extension et du biseau
Sortie :
(r,v,b)        : couleur finale du pixel courant

// Initialisation
...    // partie identique à l'algorithme 4.1
// recherche de l'Intersection
pour i allant de 1 à NB_ETAPES faire
       d = F[p.x,p.y].bleu   // la distance est récupérée à partir du canal bleu
       d = max(0, d-e)       // distance à la forme étendue
       si( d - b*p.z > 0 ) alors
              p  = p + d * Vρ
              d' = d         // sauvegarder la dernière distance
       fin si
fin pour
//affinage binaire de l'intersection
pour i allant de 1 à NB_ETAPES_BIN faire
       d' = d'/2
       d = F[p.x,p.y].bleu
       d = max(0, d-e)
       si( d - b*p.z > 0 ) alors
              p  = p + d' * Vρ
       sinon
              p  = p - d' * Vρ
       fin si
fin pour
// fenêtrage
...
// calculer la normale au point (p.x,p.y)
si p.z == 0 alors
       N = Nf ;
sinon
       N = (2*F[p.x,p.y].rouge-1, 2*F[p.x,p.y].vert-1, b)
       N = normer(N)
       si répétition_miroir alors
              N.x = -N.x si E(p.x) est impair     // E() est la partie entière
              N.y = -N.y si E(p.y) est impair
       fin si
fin si
// calcul de l'ombrage (Blinn-Phong)
...
```

4.3.4 Placage d'extrusion avec chanfrein

Afin d'avoir des bords avec un chanfrein, nous devons limiter l'effet du biseau à une certaine profondeur (entre 0 et 1). Pour ce faire, il suffit de remplacer la formule (4.3), qui définit le test au-dessus/au-dessous du placage d'extrusion biseautée, par la formule suivante :

$$\begin{aligned} \Delta d_i &= \max(0, d_i - e) - b \cdot \min(z_i, c) \\ \Delta d_i &= \max(0, d_i - e) - \delta_i \end{aligned} \tag{4.4}$$

où c est un paramètre qui permet de moduler le chanfrein, en limitant l'effet biseau entre les valeurs 0 et 1. La modification de la coordonnée z de la normale doit également se limiter à la profondeur c.

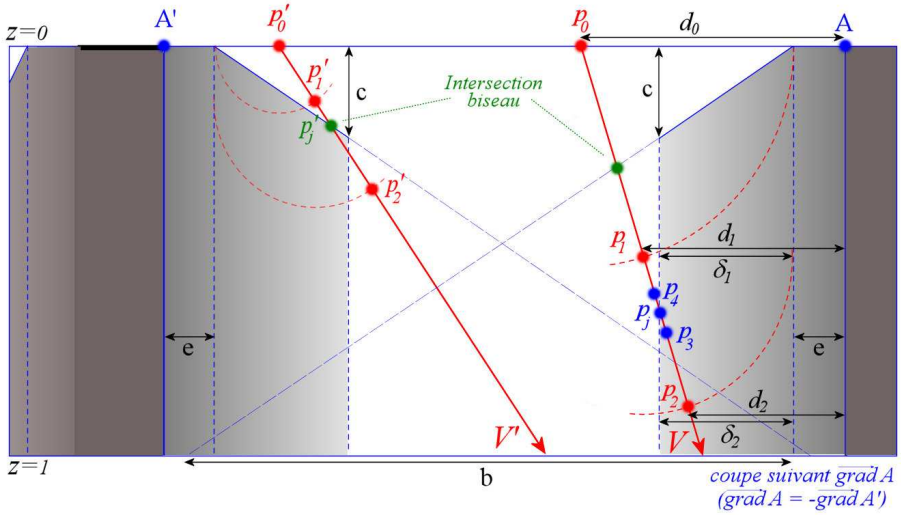


Fig. 4.8 - Une coupe 2D verticale de l'extrusion avec chanfrein.

Cependant, l'utilisation d'un algorithme similaire à celui de l'extrusion biseautée n'est valable que pour des petites valeurs du biseau b. Pour de grandes valeurs, nous risquons de sauter une intersection dans la partie biseautée comme le montre la figure 4.8 (pour le rayon de vue $\vec{V}'$). Pour contourner ce problème, nous commençons d'abord par effectuer une recherche de l'intersection du rayon de vue avec l'extrusion biseautée. Ce point sera l'intersection retenue si sa profondeur est inférieure à la valeur de c. Sinon, nous effectuons une deuxième recherche en utilisant formule (4.5), qui est équivalente à celle de l'extrusion étendue (cas du rayon $\vec{V}$ dans la figure 4.8). Cette formule est la suivante :

$$p_{i+1} = p_i + \max\left(0, d_i - (e + b \cdot c)\right) \cdot \vec{V}_p \tag{4.5}$$

Algorithme 4.4 - Placage d'Extrusion avec Chanfrein (PEC)

```
Entrée :
...            : Données identiques à celles du Placage d'Extrusion (Algorithme 4.1)
e, b, c        : Composantes de l'extension, du biseau et du chanfrein
Sortie :
(r,v,b)        : couleur finale du pixel courant

// Initialisation
...
// récupérer l'intersection avec l'extrusion biseautée
p' = intersection_biseau     // algorithme 4.3
si p'.z < c alors
       p = p'
sinon
       // Recherche de l'Intersection
       pour i allant de 1 à NB_ETAPES faire
              e' = e + b*c
              d  = F[p.x,p.y].bleu
              p  = p + max(0, d-e') * Vρ
       fin pour
fin si
// fenêtrage
...
// calculer la normale au point (p.x,p.y)
si p.z == 0 alors
       N = Nf ;
sinon
       si p.z < c alors
              N = (2*F[p.x,p.y].rouge-1, 2*F[p.x,p.y].vert-1, b)
              N = normer(N)
       sinon
              N = (2*F[p.x,p.y].rouge-1, 2*F[p.x,p.y].vert-1, 0)
       fin si
       si répétition_miroir alors
              N.x = -N.x si E(p.x) est impair     // E() est la partie entière
              N.y = -N.y si E(p.y) est impair
       fin si
fin si
// calcul de l'ombrage (Blinn-Phong)
...
```

4.4 Placage de révolution

Le placage de révolution consiste à placer la carte de forme verticalement par rapport la surface du polygone, puis de lui appliquer une révolution autour d'un axe perpendiculaire passant par un point donné ***O***, généralement (0,5, 0,5). Nous obtenons ainsi une surface de révolution virtuelle au dessous du polygone.

La carte de forme représente une coupe verticale de la surface de révolution passant par le centre de révolution et le point de départ $p_0=(x_0,y_0,0)$. Or, la distance minimale d'un point de la coupe à la surface de révolution, est la distance minimale entre ce point et la forme 2D révolue. Nous pouvons alors utiliser la transformation distance Euclidienne pour sauter l'espace vide, ce qui va permettre de converger rapidement vers le point d'intersection entre le rayon de vue et la surface de révolution (voir la figure 4.9). Les différentes étapes de cette technique sont présentées dans l'algorithme 4.5.

Notre approche pour le traçage de rayons pour les surfaces de révolution est fondamentalement différente des algorithmes utilisés dans le rendu par lancer de rayons. En effet, tous ces algorithmes sont basés sur une décomposition géométrique de la surface de révolution (maillage, collection de cônes tronqués, surface quadrique). Quant à l'accélération du traçage de rayons, elle s'effectue à l'aide d'un arbre de bandes (*strip tree*), en englobant la courbe génératrice [Kaj83, Bid90, BJ01], ou avec un arbre de partitionnement binaire de l'espace (BSP tree), en englobant la surface de révolution [JTJ04].

4.4.1 Calcul du point d'intersection

Soit $p_i = (x_i, y_i, z_i)$ la position actuelle sur l'axe de vue, et soit $\vec{V}$ le vecteur de vue normé et mis en échelle. Nous cherchons l'intersection de l'axe de vue avec la surface de révolution. Pour ce faire, nous allons nous baser sur la carte de distance pour calculer, à chaque étape, le pas vers la position suivante. Nous devons tout d'abord calculer les coordonnées (s_i,t_i), afin d'accéder à la carte de forme, et récupérer ainsi la distance minimale entre la position actuelle et la forme de base (voir la figure 4.9). s_i est donnée par la distance qui sépare le point p_i de l'axe de révolution. Quant à t_i, elle est donnée par la profondeur du point p_i :

$$(s_i\,,\,t_i) = \left(\left\|\vec{R}_i\right\|,\, z_i\right) \qquad \text{avec}:\ \vec{R}_i = \left(x_i - O_x,\, y_i - O_y\right) \tag{4.6}$$

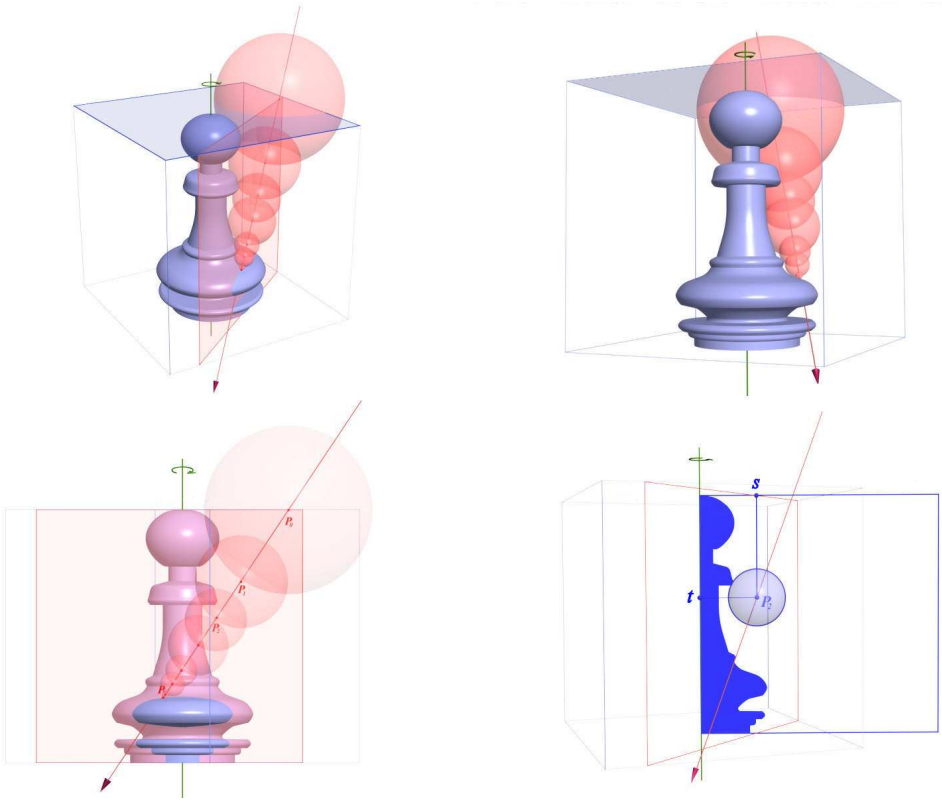


Fig. 4.9 - Procédure d'intersection entre le rayon de vue et la surface révolue.

La valeur d_i, récupérée depuis la carte de distance aux coordonnées (s_i, t_i), fournit une distance de sécurité autour du point p_i. Cette distance forme un cercle dans la coupe verticale passant par p_i et incluant la direction de vue $\vec{V}$. La révolution de ce cercle va générer un tore. p_{i+1} sera logiquement l'intersection du rayon de vue avec ce tore. Cependant, le calcul d'une telle intersection est très coûteux pour le rendu temps réel. Nous devons alors nous tourner vers une approximation acceptable. En effet, puisque la sphère de rayon d_i est contenue entièrement dans le tore, nous pouvons alors choisir d_i comme un pas d'avancement. Ainsi, la prochaine position sur l'axe de vue sera donnée par la formule suivante:

$$p_{i+1} = p_i + d_i \cdot \vec{V}$$

Le facteur de rectification pour les cartes de forme non carrées n'est pas équivalent à celui utilisé dans le placage d'extrusion. Puisque la carte de forme est placée verticalement, la rectification de la distance mènera à la formule suivante pour p_{i+1} :

$$p_{i+1} = p_i + d_i \cdot \frac{\left\|\vec{V}\right\|}{\sqrt{V_x^2 + V_y^2 + \rho^2 V_z^2}} \cdot \vec{V}$$

Et puisque $\left\|\vec{V}\right\| = 1$, nous pouvons utiliser l'optimisation suivante :

$$P_{i+1} = P_i + d_i \cdot \vec{V}_\rho \quad \text{avec:} \quad \vec{V}_\rho = \frac{1}{\sqrt{V_x^2 + V_y^2 + \rho^2 V_z^2}} \cdot \vec{V}$$

Comme dans le cas de l'extrusion, nous pouvons là aussi lisser la forme, en modulant un paramètre d'extension *e* :

$$P_{i+1} = P_i + \max(0, d_i - e) \cdot \vec{V}_\rho$$

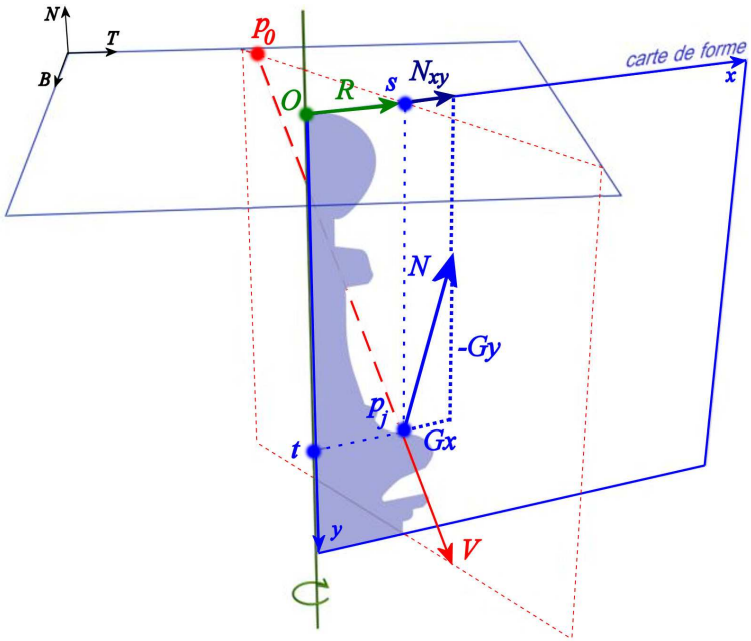


Fig. 4.10 - Calcul de la normale au point d'intersection avec la surface de révolution.

4.4.2 Calcul de la normale au point d'intersection

Après avoir calculer le point d'intersection p_j, nous récupérons les coordonnées du gradient unitaire (Gx_j, Gy_j) au point (s_j, t_j) de la carte de forme, respectivement dans les canaux *rouge* et *vert*. La carte de forme étant placée verticalement, la composante *z* de la normale $\vec{N}$ correspond à l'opposé[1] de la composante Gy_j. La composante Gx_j fournit la norme du sous-vecteur $\vec{N}_{xy}$. La direction de ce dernier est donnée par le sous-vecteur unitaire: $\vec{R}_j / \left\| \vec{R}_j \right\|$ (voir la figure 4.10). Nous déduisons finalement les trois composantes de la normale :

[1] Nous utilisons l'opposé de Gy car la direction de l'axe y de la carte de forme est opposée à la direction de l'axe z de l'espace TBN.

$$\vec{N}_j = \left(Gx_j \frac{R_{j_x}}{\left\| \vec{R}_j \right\|}, Gx_j \frac{R_{j_y}}{\left\| \vec{R}_j \right\|}, -S_z \cdot Gy_j \right)$$

avec :

$$S_z = \begin{cases} +1 & si\ E(z_j)\ est\ pair \\ -1 & sinon \end{cases}$$

4.4.3 Répétition de la carte de forme

Si la carte de forme est plaquée avec répétition, il y aura autant d'axes de révolution que de répliques (voir la figure 4.11). Tous les calculs doivent alors être effectués en fonction de l'axe de révolution le plus proche du point p_i. La position de cet axe se calcule en fonction des parties entières des coordonnées de p_i :

$$O_i = \left(O_x + \mathrm{E}(x_i), O_y + \mathrm{E}(y_i) \right)$$

Le point O doit être remplacé par O_i dans la formule (4.6) qui devient alors :

$$(s_i, t_i) = \left(\left\| \vec{R}_i \right\|, z_i \right) \qquad avec :\ \vec{R}_i = \left(x_i - E(x_i) - O_x,\ y_i - E(y_i) - O_y \right)$$

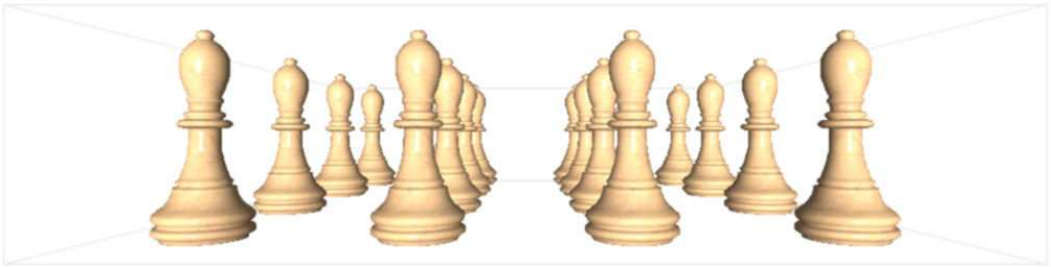


Fig. 4.11 - Rendu du placage de révolution avec répétition d'une pièce d'échecs

Algorithme 4.5 - Placage de Révolution (PRév)

```
Entrée :
W, L, Nf          : Rayon de vue normé, rayon de lumière normé, et normale du polygone.
                    Tous exprimés dans l'espace TBN.
(u,v),(Tx,Ty,Tz),κ: Pixel courant, nombre de répétition sur x, y et z, échelle de profondeur
C, F, ρ, τ        : carte de couleur, carte de forme, son rapport, et l'échelle de distance
Ka, Kd, Ks, n     : Composantes ambiante, diffuse, spéculaire et lustre
O                 : Centre de révolution
Sortie :
(r,v,b)           : couleur finale du pixel courant

// Initialisation
p = (Tx*u, Ty*v, 0)                  // point de départ
V = (W.x,W.y,-W.z/κ), V=V/norme(V)    // mise à l'échelle et inversion du sens suivant z
Vρ= τ*V/(Vx*Vx+Vy*Vy+ρ*ρ*Vz*Vz)       // rectification elliptique et modulation de la distance
// Recherche de l'Intersection
pour i allant de 1 à NB_ETAPES faire
       Oi.x = O.x + E(p.x), Oi.y = O.y + E(p.y)    // E est la partie entière
       R  = (p.x-Oi.x, p.y-Oi.y)
       s = norme(R), t = p.z
       d  = F[s,t].bleu
       p  = p + d * Vρ
fin pour
// fenêtrage
si p.z>Tz ou p.x<0 ou p.x>Tx ou p.y<0 ou p.y>Ty alors
       annuler       // pas d'intersection
fin si
// récupérer la normale au point (s,t)
si p.z == 0 alors
       N = Nf ;
sinon
       Gx = 2*F[s,t].rouge-1, Gy = 2*F[s,t].vert-1
       normer(R)
       N = (Gx*R.x, Gx*R.y, -Gy)    // N est déjà normée
       si répétition_miroir et E(p.z) est impair alors
              N.z = -N.z
       fin si
fin si
// calcul de l'ombrage (Blinn-Phong)
(ξ,η)   = TexCoord(p)        // fonction qui calcule les coordonnées de textures
(r,v,b) = C[ξ,η]             // récupérer la couleur de la texture
H = (L+V)/norme(L+V)
(r,v,b) = Ka*(r,v,b) + Kd*(r,v,b)*max(0,N.L) + ks*max(0, (H.N)^n )
```

4.5 Correction du tampon de profondeurs

Afin de prendre en compte l'interaction entre la géométrie virtuelle, créée par placage d'extrusion ou de révolution, et les autres objets de la scène, nous devons mettre à jour la profondeur *z* des pixels pour lesquels une intersection a été trouvée (c-à-d, les pixels faisant partie de la forme extrudée ou révolue). Cette étape est nécessaire pour permettre l'interpénétration entre les différents objets de la scène (voir la figure 4.12). Elle est également indispensable pour la prise en charge de la technique du placage des ombres. La valeur rectifiée de la profondeur *z* se calcule à l'aide de la formule suivante :

$$z = \frac{\Pi_r \cdot (z_{vue} + \Pi_v)}{z_{vue} \cdot (\Pi_r - \Pi_v)}$$

où z_{vue} est la coordonnée *z* du point d'intersection, exprimé dans l'espace de vue. Π_v et Π_r sont les distances associées aux plans *avant* et *arrière* de la zone de fenêtrage 3D.

Toutefois, la mise à jour du tampon de profondeurs va entraîner un ralentissement des performances, suite à la désactivation automatique par le pipeline graphique du test initial d'occlusion. Ce test permet d'éviter le traitement d'un grand nombre de pixels, qui seront de toute façon masqués par d'autres pixels qui ont une profondeur inférieure. Par conséquent, cette option ne doit être activée que si elle est jugée nécessaire par le développeur. Toutefois, une adaptation de la technique du test initial d'occlusion explicite, décrite dans [STM04, MS04], pourrait optimiser la vitesse du rendu.

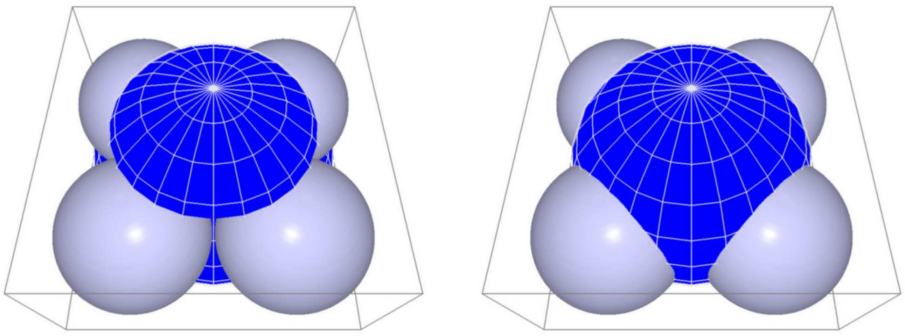


Fig. 4.12 - Mise en évidence de la mise à jour du tampon de profondeurs. (A gauche) Rendu sans correction de profondeurs. (A droite) La mise à jour de la profondeur des pixels des petites sphères permet une interpénétration correcte entre les deux types d'objets.

4.6 Modélisation et rendu d'objets extrudés ou révolus

Le placage de la carte de forme sur de simples maillages polygonaux, ne permettra pas d'afficher des objets formés par une extrusion ou une révolution dans leur intégralité. En effet, la géométrie ne sera visible que depuis la partie texturée des polygones. La figure 4.13 illustre bien cette problématique. Afin de pouvoir modéliser et afficher des objets 3D entiers, quelque soit l'angle de vue, nous devons construire une sorte de coquille (*shell space*). Cette coquille s'obtient en entourant le volume virtuel de l'extrusion ou de la révolution, par des polygones supplémentaires.

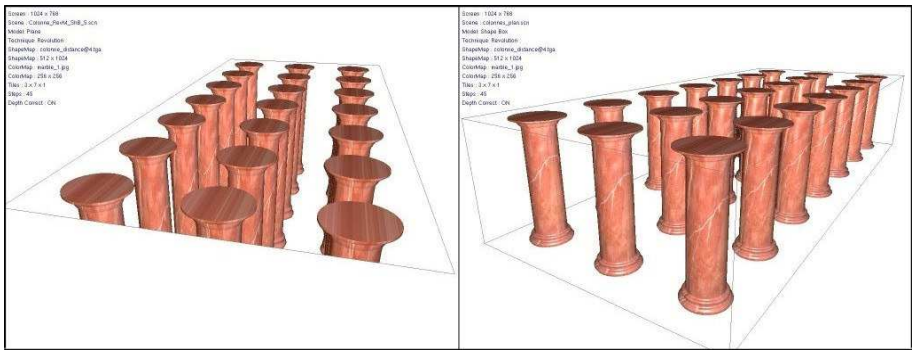


Fig. 4.13 - Utilité de la boite de forme. (A gauche) Le placage de révolution sur un plan simple ne permet pas de représenter les objets en entier. (A droite) l'utilisation de la boite de forme permet l'affichage des objets dans leur intégralité quelque soit l'angle de vue.

4.6.1 Création de la boite de forme

La coquille pourra être créée pour tout le maillage, en extrudant tous ses triangles, suivant la direction des normales aux sommets. L'utilisation de la coquille permettra d'éviter les distorsions qui apparaissent sur les surfaces fortement courbées, et qui sont dues à l'approximation planaire. Cependant, pour pouvoir afficher des objets créés à partir d'extrusions ou de révolution, il suffit d'utiliser une coquille formée par une seule boite. Cette dernière sera créée à partir de la carte de forme, comme l'illustre la figure 4.14. *W* et *H* sont les dimensions de la carte de forme. *S* est un vecteur modifiable en temps réel, représentant l'échelle de la boite dans le repère de la scène. Les valeurs en bleu sont les coordonnées de texture.

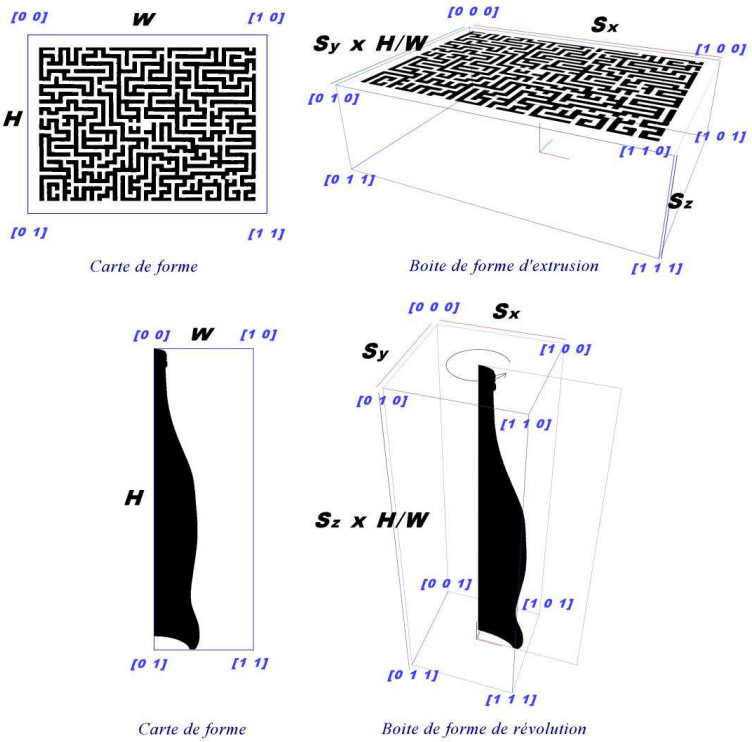


Fig. 4.14 - Création de la boite de forme en fonction des dimensions de la carte de forme. (En haut) une boite de forme extrudée, ici la forme est placée sur la surface de la boite. (En bas) une boite de forme révolue où la forme est placée verticalement à la surface.

L'utilisation de la boite de forme entraînera quelques changements dans les algorithmes du placage d'extrusion (de 4.1 à 4.4) :

```
p  = (Tx*u, Ty*v, w)  // le point de départ n'est plus forcément sur la surface
...
// fenêtrage
si p.z<0 ou p.z>1 ou p.x<0 ou p.x >Tx ou p.y<0 ou p.y>Ty alors
       annuler              // pas d'intersection
fin si
...
```

Dans le cas du placage de révolution (algorithme 5.5), où la répétition se fait également sur l'axe des z, Nous aurons les changements suivants :

```
p  = (Tx*u, Ty*v, Tz*w)      // le point de départ n'est plus forcément sur la surface
...
// fenêtrage
si p.z<0 ou p.z>Tz ou p.x<0 ou p.x >Tx ou p.y<0 ou p.y>Ty alors
       annuler              // pas d'intersection
fin si
...
```

4.6.2 Orientation avant/arrière des polygones

Les modèles architecturaux, créés à partir d'extrusions et de révolutions, peuvent avoir de très grandes dimensions. Il est alors nécessaire que l'observateur puisse s'y rapprocher, et même d'y naviguer. Or, avec l'orientation par défaut vers l'avant[1], l'objet disparaît dés que la position de l'observateur se trouve à l'intérieur de la boite de forme. Pour contourner ce problème, il faudra passer en mode orientation vers l'arrière. Dans ce cas, le point de départ pour la recherche d'intersection devient la position de l'observateur, et non plus le pixel courant. Cette position doit être exprimée dans l'espace tangent. Le résultat des deux modes d'orientation est présenté dans la figure 4.15.

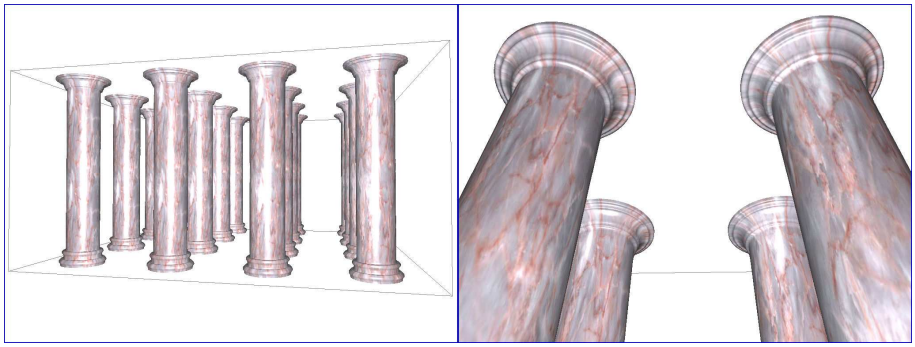


Fig. 4.15 - Changement de l'orientation des polygones suivant la position de l'observateur. (A gauche) L'orientation par défaut (vers l'avant). (A droite) Basculement vers l'orientation vers l'arrière.

4.6.3 Texturation de la boite de forme

Après le calcul du point d'intersection p_j, ainsi que de sa normale N_j, il faudra déterminer ses coordonnées de texture (ξ_j,η_j) afin de pouvoir envelopper une texture couleur sur la surface de révolution ou d'extrusion. L'utilisation directe des coordonnées (x_j,y_j,z_j) ne permettra d'avoir qu'un plaquage avec projection planaire, qui conviendra bien à la boite d'extrusion, mais pas à celle de la révolution. En effet, Celle-ci nécessitera une projection cylindrique ou sphérique.

Comme le plaquage de texture doit être effectué par rapport à chaque réplique. Nous devons utiliser seulement la partie fractionnaire des coordonnées de l'intersection :

$$(x'_j, y'_j, z'_j) = (x_j, y_j, z_j) - \left(\mathrm{E}(x_j), \mathrm{E}(y_j), \mathrm{E}(z_j)\right)$$

[1] Rendu d'un polygone seulement si sa normale pointe vers l'observateur

4.6.3.1 Projection planaire

Le plaquage par projection planaire se déduit directement des coordonnées du point d'intersection p_j, et du plan de projection choisi. Nous avons par exemple:

$$\begin{cases} (\xi_j, \eta_j) = (x'_j, y'_j) & pour\ la\ droite : (z = 0) \\ (\xi_j, \eta_j) = (x'_j, z'_j) & pour : (y = 0) \\ (\xi_j, \eta_j) = \left(\frac{1}{2}(x'_j + y'_j),\ z'_j\right) & pour : (x = y) \end{cases}$$

4.6.3.2 Projection cylindrique

Pour avoir un plaquage par projection cylindrique, nous utilisons les coordonnées cylindriques du point d'intersection p_j, exprimées par rapport à un axe centré sur la boite de forme :

$$\begin{cases} x'_j - 0{,}5 = r\cos\varphi \\ y'_j - 0{,}5 = r\sin\varphi \end{cases} \quad avec : \ r = \sqrt{(x'_j - 0{,}5)^2 + (y'_j - 0{,}5)^2}$$

Nous en déduisons la valeur de φ:

$$\varphi = \text{atan2}(y'_j - 0{,}5,\ x'_j - 0{,}5)$$

La coordonnée de texture η_j est égale à z'_j. Quant à ξ_j, elle passe de 0 à 1 lorsque φ passe de π à $-\pi$. Nous avons alors:

$$(\xi_j, \eta_j) = \left(\frac{\pi - \varphi}{2\pi},\ z'_j\right)$$

Pour la placage de révolution, il faut utiliser (O_x,O_y) au lieu de (0,5,0,5) si l'axe de révolution n'est pas centré sur la boite de forme.

4.6.3.3 Projection sphérique

Le plaquage par projection sphérique s'obtient en fonction des coordonnées sphériques du point d'intersection p_j, exprimées par rapport à un repère placé au centre de la boite de forme.

$$\begin{cases} x'_j - 0{,}5 = r\sin\theta\cos\varphi \\ y'_j - 0{,}5 = r\sin\theta\sin\varphi \\ y'_j - 0{,}5 = r\cos\theta \end{cases} \quad avec : \ r = \sqrt{(x'_j - 0{,}5)^2 + (y'_j - 0{,}5)^2 + (z'_j - 0{,}5)^2}$$

Nous en déduisons les valeurs de φ et θ :

$$\begin{cases} \varphi = \text{atan2}(y'_j - 0{,}5,\ x'_j - 0{,}5) \\ \theta = \arccos\dfrac{z'_j - 0{,}5}{r} \end{cases}$$

ξ_j varie dans [0,1] lorsque φ passe de π à -π, et η_j varie dans [0,1] quand θ passe de π à 0. Nous avons alors :

$$(\xi_j, \eta_j) = \left(\frac{\pi - \varphi}{2\pi}, \frac{\pi - \theta}{\pi} \right)$$

4.7 Résultats

Nous avons implémenté les différents algorithmes décrits dans ce chapitre en C++ avec la bibliothèque graphique OpenGL, associée à son langage de nuanceurs GLSL. Les mesures et les figures ont été réalisées à l'aide d'un PENTIUM IV avec 512Mb de RAM, et d'une carte graphique GeForce 7600GS AGP avec 256Mb de mémoire vidéo.

4.7.1 Prétraitement de la carte de forme

Le tableau 4.1 présente le temps du prétraitement de quelques cartes de forme sous différentes résolutions. Le temps de calcul des cartes varie significativement (de 0.1s à 2s), en fonction de la taille de la carte de forme, du mode de répétition, et de la taille du filtre du gradient. Mais globalement, le calcul de la carte de forme s'effectue très rapidement, et reste du même ordre que le calcul d'une carte de normales traditionnelle.

		EDT		**Gradient**		**Total miroir**		**Total décalage**	
		miroir	**décalage**	**5x5**	**9x9**	**5x5**	**9x9**	**5x5**	**9x9**
Formes	**512x512**	0,031	0,281	0,078	0,250	0,109	0,281	0,359	0,531
Formes	**1024x1024**	0,173	1,016	0,343	1,015	0,140	0,328	0,359	0,547
Tubes	**512x512**	0,047	0,235	0,110	0,297	0,516	1,188	1,359	2,031
Pion	**512x512**	0,047	0,266	0,093	0,281	0,624	1,343	0,344	2,063
Pion	**1024x1024**	0,218	0,938	0,406	1,125	0,250	0,516	0,531	0,797
Bouteille	**376x1150**	0,094	0,375	0,156	0,422	0,109	0,281	0,359	0,531

Tab. 4.1 - Temps de calcul (Gradient et EDT) de différentes cartes de forme. L'EDT avec décalage prend plus de temps car le nombre de pixels est multiplié par deux. La taille du filtre gradient influe également sur la durée du prétraitement. Cependant, le calcul se fait très rapidement et reste proche de celui de la génération d'une carte de normales classique.

La figure 4.16 met en évidence l'importance de la taille du filtre gradient. En effet, le filtre 3x3 est très sensible à la pixellisation de la forme de base, et entraîne un effet de crénelage assez

marqué (effet escalier). Pour cette raison, il est préférable d'utiliser une taille de filtre plus conséquente. Le filtre 9x9 par exemple, donne de bons résultats, et permet de lisser correctement les contours de la forme.

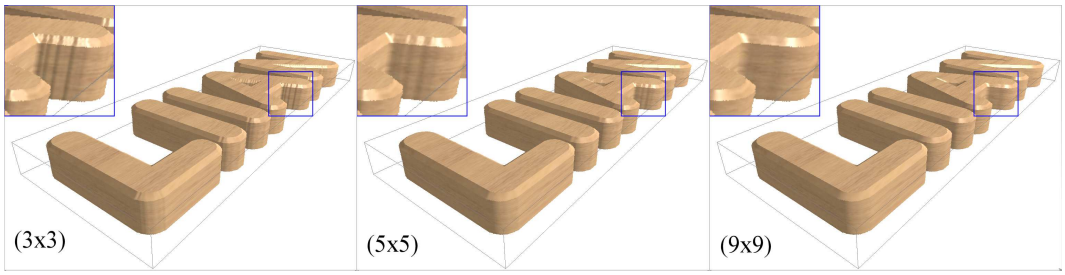


Fig. 4.16 - Trois rendus d'une même forme avec différentes tailles pour le filtre du gradient. Le filtre 3x3 est très sensible à la pixellisation de la forme. Le filtre 9x9 permet de corriger l'effet escalier et lisse correctement la forme.

4.7.2 Rendu avec placage d'extrusion et de révolution

Le tableau 4.2 affiche les vitesses du rendu des différentes techniques proposées, ainsi que certaines techniques de référence. Le nombre total d'itérations pour la recherche d'intersection est 30, sauf pour le placage d'extrusion biseautée (35 dont 10 pour l'affinage binaire), et le placage d'extrusion avec chanfrein (60 dont 35 pour la phase *biseau*). Comme prévu, la technique de placage d'extrusion, basique et étendue, est nettement plus rapide. Les autres techniques sont néanmoins rendues avec un taux de rafraîchissement interactif (supérieur à 16). Les images de la figure 4.17 sont des captures d'écran prises pendant le test. Nous pouvons y remarquer que la boite de forme occupe la majeure partie de l'écran.

Il faut noter que la différence de vitesse entre les différentes techniques n'est pas homogène. Elle peut dépendre de plusieurs paramètres, comme la complexité de la forme de base, la taille de la carte de forme, l'échelle de profondeur, ou encore le nombre de répliques.

La figure 4.18 effectue une comparaison entre le placage d'extrusion, et quelques techniques de référence concernant le placage de déplacement par pixel. Nous remarquons d'une manière très nette, que ces algorithmes ne sont guère adaptés à l'extrusion de formes. Spécialement pour de grandes échelles de profondeurs, ou pour des motifs très fins. En plus, ces techniques sont nettement moins rapides que le placage d'extrusion.

		Ecran: 800 x 600		Ecran: 1024 x 768	
		512 x 512	1024 x 1024	512 x 512	1024 x 1024
Placage d'Extrusion Basique	**(PE)**	126	88	87	46
Placage d'Extrusion Etendue	**(PEE)**	110	74	75	41
Placage d'Extrusion Biseautée	**(PEB)**	71	33	47	30
Placage d'Extrusion avec Chanfrein	**(PEC)**	44	24	29	17
Placage de Relief	**(PR)**	76	42	50	40
Placage de Parallaxe avec Occlusion	**(PPO)**	54	41	34	32
Traçage de Cônes	**(TC)**	63	30	42	24
Placage de Révolution	**(PRév)**	35	22	25	16

Tab. 4.2 - Vitesse du rendu du placage d'extrusion et de révolution ainsi que quelques techniques de référence (calculé en nombre d'images par seconde). Nous pouvons constater que la technique de placage d'extrusion (simple et étendue) est nettement plus rapide. Les autres techniques sont néanmoins rendues avec une vitesse interactive.

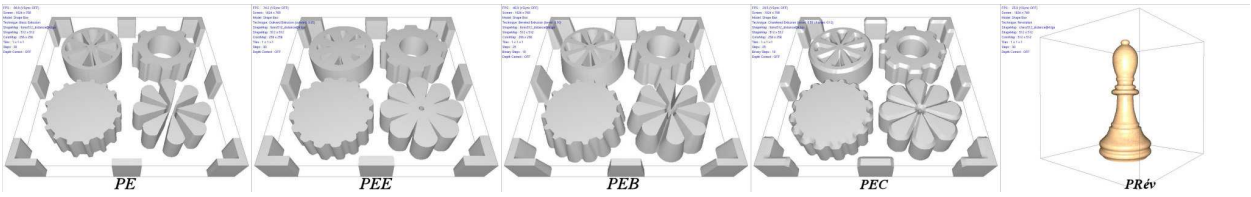


Fig. 4.17 - Captures d'écran pendant le calcul de la vitesse du rendu par placage d'extrusion et de révolution. Notez bien que la boite de forme occupe la majeure partie de l'écran.

La figure 4.19 met en évidence certains défauts liés à l'utilisation d'images pour la modélisation et le rendu d'objets 3D. Les images de gauche montrent des défauts de crénelage qui apparaissent lors d'un large zoom sur l'objet. Ce problème peut être atténué avec les filtres anti-crénelage ou avec l'utilisation des textures de plus grande résolution. Cependant, une approche basée sur les textures vectorielles reste à explorer, en s'inspirant notamment sur les travaux décrits dans [PZ08, NH08, LB08, QMK08, BPMG08]. Les images de droite révèlent l'échec de la recherche de l'intersection, lorsque le rayon de vue passe par plusieurs répliques (particulièrement pour la révolution). Nous pouvons résoudre ce problème en augmentant le nombre d'itérations, ou en utilisant un réseau de boites de forme au lieu d'un placage avec répétition.

Les figures 4.20, 4.21 et 4.22 présentent quelques models créés par placage d'extrusion et de révolution. Nous pouvons y observer la grande variété d'objets pouvant être modélisés à l'aide de ces techniques. Le nombre de primitives graphiques (sommets et polygones) qui peuvent être ainsi évitées est très important, et dépend naturellement de la complexité de la forme de base. Signalons également que la méthode de placage de texture peut être adaptée à la nature de l'objet. Il suffit pour cela de modifier la fonction de texturation dans le nuanceur de pixels.

Notons aussi que le placage de révolution produit une géométrie très fine, sans l'aspect anguleux visible sur les bords du maillage traditionnel. Enfin, nous observons que l'utilisation d'une échelle non-uniforme permet de produire des révolutions elliptiques correctes.

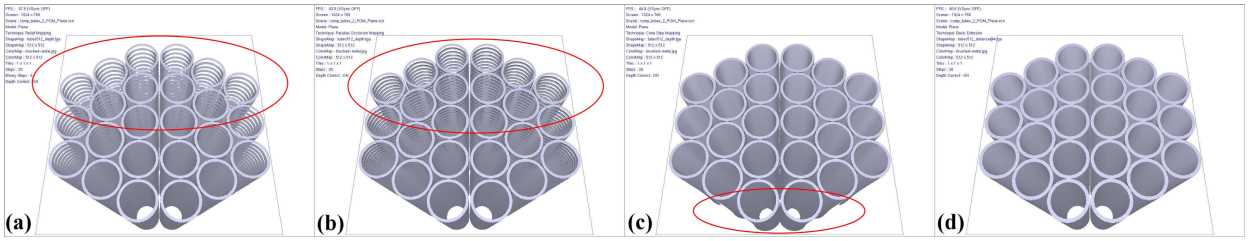


Fig. 4.18 - Comparaison entre la technique du placage d'extrusion et différentes techniques de placage de déplacement. (a) Placage de relief (57 IPS). (b) Placage de parallaxe avec occlusion (43 IPS). (c) Traçage de cônes (44 IPS). (d) Placage d'Extrusion (80 IPS). La taille de la carte de forme est de 512x512. La résolution de l'écran est de 1024x768. Le nombre total d'itérations est 30.

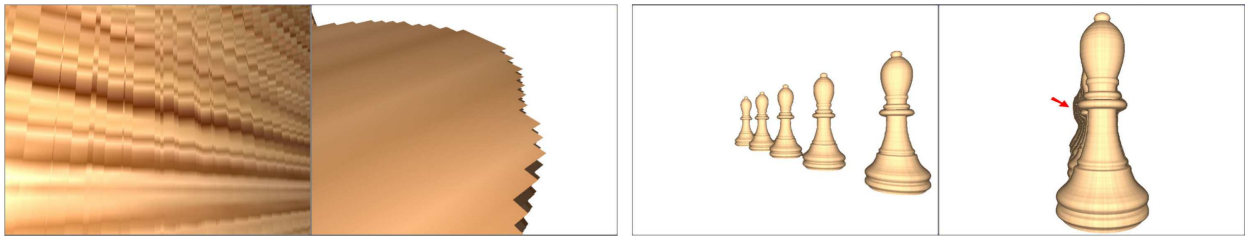


Fig. 4.19 - Défauts dus à l'utilisation des images pour le rendu de géométrie. (En haut) des défauts de crénelage visibles lors d'un large zoom sur un objet crée avec le placage d'extrusion. (En bas) l'échec de la recherche d'intersection lorsque le rayon de vue traverse plusieurs répliques.

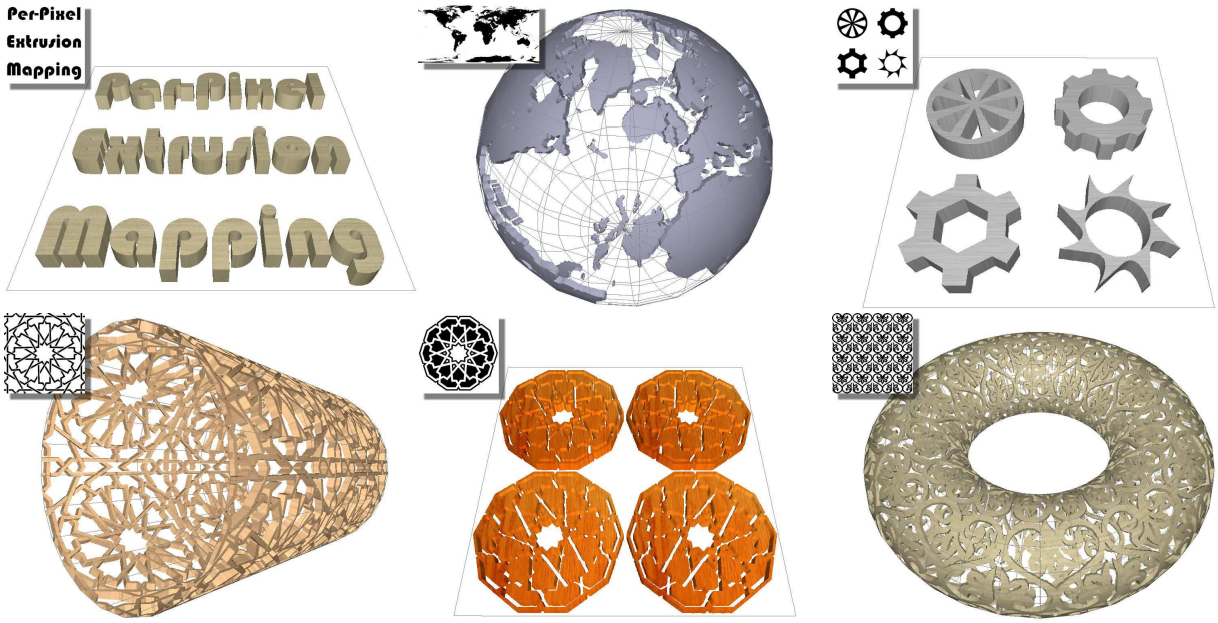


Fig. 4.20 - Quelques objets créés à l'aide du placage d'extrusion. Ces objets sont obtenus à partir d'un maillage de faible densité, sur lequel une carte de forme a été plaquée. La technique de placage d'extrusion permet de restituer une géométrie très détaillée, évitant ainsi de recourir à un maillage beaucoup plus dense.

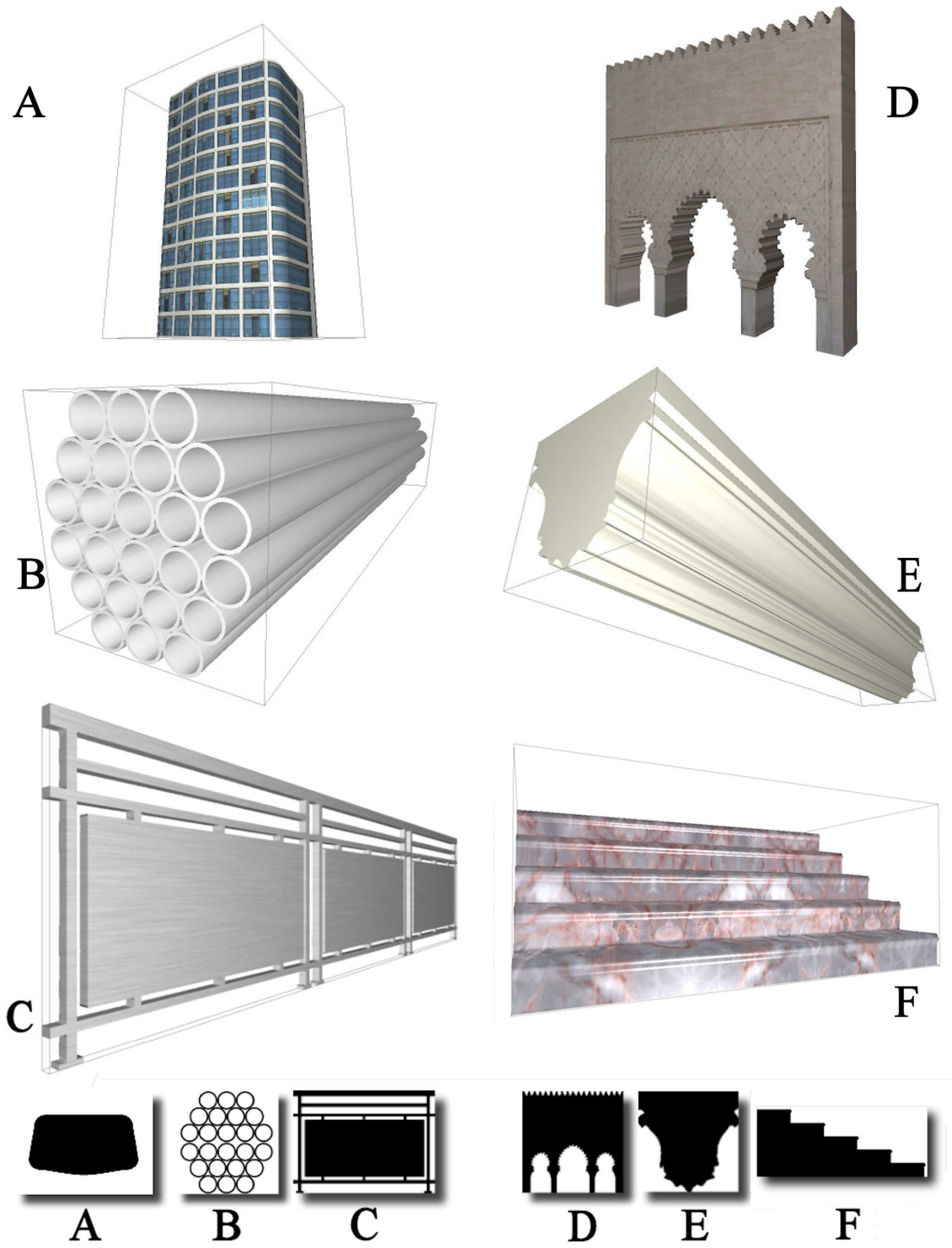


Fig. 4.21 - Collection de modèles créés à l'aide du placage d'extrusion appliquée à une boite de forme. La qualité visuelle de ces modèles, leur vitesse de traitement et leur faible occupation mémoire, les destinent naturellement à remplacer leurs équivalents en maillages polygonaux.

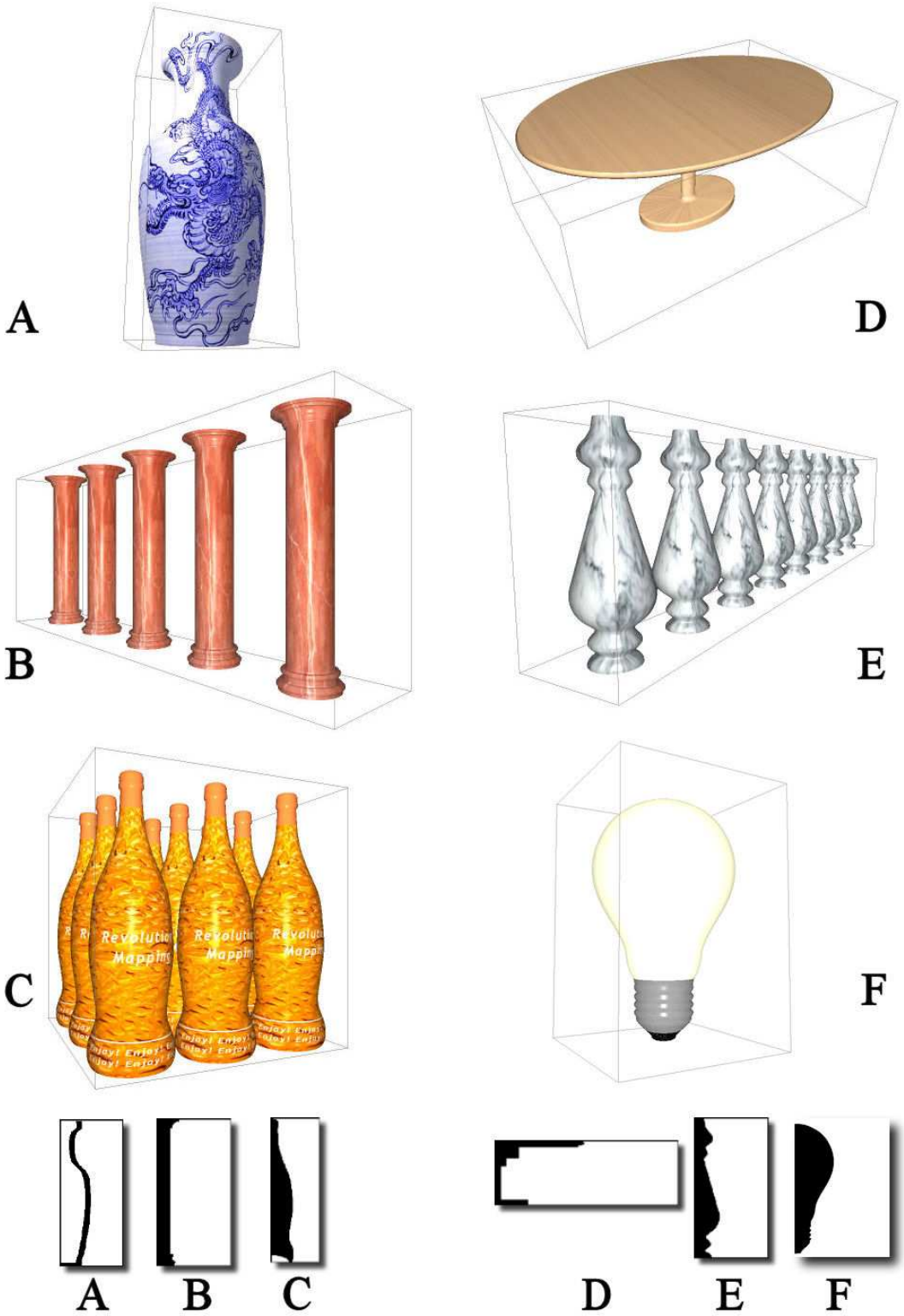


Fig. 4.22 - Collection d'objets créés en utilisant le placage de révolution. Nous pouvons remarquer que la silhouette des objets est parfaitement lisse, contrairement aux maillages dont les bords sont plus ou moins anguleux en fonction de leur densité. Notez également que l'échelle non-uniforme appliquée à la boite de forme (D) produit une révolution elliptique correcte.

4.8 Conclusion

Dans ce chapitre, nous avons introduit une nouvelle approche pour le rendu de géométrie créée à partir d'opérations d'extrusion ou de révolution. Nous avons utilisé une seule texture, qui regroupe à la fois: Une image binaire représentant une forme 2D, sa transformation distance Euclidienne, et le gradient unitaire de cette dernière. La distance est utilisée pendant la phase de la recherche d'intersection entre l'axe de vue et la géométrie virtuelle, générée par l'extrusion ou la révolution de la forme de base. Cette distance permet de sauter l'espace vide, et de converger ainsi plus rapidement vers le point d'intersection. Notre méthode sert non seulement à afficher des motifs extrudés sur des maillages polygonaux, mais également à créer des objets entiers extrudés ou révolus.

Nous sommes convaincus que pour les applications temps réel, le rendu basé sur les images constitue une alternative très sérieuse au traditionnel maillage de polygones. Car ce dernier sera toujours limité par le nombre de primitives graphiques que le matériel pourra traiter.

Les formes 2D sont généralement créées à l'aide de splines. Or, la plupart des logiciels 3D intègrent la création de splines, qui servent par la suite à générer des maillages par des opérations d'extrusion ou de révolution. Il serait alors très intéressant de pouvoir créer des cartes de forme à partir de ces splines, et ainsi pouvoir les manipuler en temps réel, afin d'obtenir la forme souhaitée. Le placage d'extrusion et de révolution peut également être utilisé par le moteur de rendu temps réel des logiciels 3D et d'architecture, et qui intervient pendant la phase de modélisation. Cette technique peut aussi servir à générer des rendus « brouillons » (c-à-d très rapides), pour des scènes très complexes.

Plusieurs améliorations potentielles peuvent être apportées à la technique de placage d'extrusion et de révolution. Notamment pour varier les objets pouvant être créés par cette technique, et pour améliorer l'interaction des boites de forme avec les autres objets de la scène. Citons par exemple :

- Des extrusions non linéaires, ainsi que des biseaux et des chanfreins arrondis.
- Opérations booléennes entres différentes boites de forme.
- Combinaison entre le placage d'extrusion/révolution et le placage de bosselures afin d'ajouter un effet de microreliefs.
- Un algorithme interactif de détection de collision qui prend en compte les géométries créées à l'aide des boites de forme.

Conclusion et Perspective

Cette thèse expose nos contributions dans le domaine du rendu 3D en temps réel. Nous nous sommes particulièrement intéressés à la représentation des mesostructures, ainsi qu'à la création de modèles 3D à base d'images par opposition à la géométrie. Dans cette partie, nous commencerons par exposer un bilan de l'ensemble de nos travaux, avant de présenter une perspective de travail reliant cette thèse au domaine de la numérisation du patrimoine bâti.

5.1 Bilan

5.1.1 Représentation des mesostructures

La représentation des mesostructures est une problématique majeure dans le domaine du rendu 3D. En effet, les mesostructures ne peuvent ni être représentées par un maillage traditionnel, pour des problèmes de performances notamment, ni à l'aide de formules d'ombrage, à cause de la pauvreté visuelle. Il s'agit alors de trouver des solutions capables à la fois de fournir une qualité de rendu appréciable, et en même temps, de conserver l'interactivité des applications 3D.

Nous avons commencé par proposer une synchronisation temps réel entre la mise à jour de l'échelle de profondeur des mesostructures, et le calcul d'ombrage. En effet, les techniques existantes supposent que l'échelle de profondeur est fixée d'avance. Cependant, la possibilité de pouvoir varier l'échelle peut se révéler indispensable. Comme dans le cas où l'on souhaite utiliser la même carte de déplacement avec des échelles différentes, ou lorsque la profondeur doit être animée et constamment mise à jour. Afin d'intégrer cette fonctionnalité importante, nous avons proposé deux solutions : La première consiste à recalculer la normale en fonction de la nouvelle échelle et de la normale déjà calculée pour une échelle fixe. La deuxième approche consiste à stocker les dérivées partielles de la carte de profondeurs au lieu de ses normales, puis de calculer celles-ci pendant la phase de rendu en fonction de l'échelle de profondeur souhaitée. C'est cette deuxième solution que nous avons adoptée dans la suite de nos travaux, car elle est plus simple et plus intuitive.

Nous avons ensuite travaillé sur l'une des meilleures solutions apportée au problème de la représentation des mesostructures, à savoir : *la technique de traçage de cônes*. Cette technique permet de restituer les mesostructures d'une manière très convaincante visuellement, tout en respectant la vitesse exigée par le rendu temps réel. Toutefois, cette technique souffrait de quelques problèmes, liés notamment au temps du prétraitement des textures et à la gestion des textures non-carrées.

La méthode de traçage de cônes existe en deux versions : La première est dite *conservative*, tandis que la deuxième est dite *relaxée*. Ces deux approches adoptent des algorithmes à complexité quadratiques. Ces derniers s'avèrent très lents, particulièrement avec des textures à moyenne et grande résolution. Par conséquent, l'algorithme de traçage des cônes relaxés peut

prendre plusieurs heures pour calculer une carte de 256x256 de résolution ! Nous avons alors proposé une adaptation de l'algorithme linéaire HDDT, utilisé dans la visualisation des reliefs géographiques. L'usage de la transformation distance le rend particulièrement rapide pour le traitement des cartes de cônes conservatifs. Cependant, l'HDDT n'est pas applicable aux cartes de cônes relaxés. Nous avons alors proposé un nouvel algorithme qui se base sur le calcul des maxima locaux, en plus de la transformation distance. L'algorithme obtenu est très rapide, et permet ainsi de passer d'un temps de prétraitement de quelques heures à quelques secondes. Nous estimons que la phase de prétraitement joue un rôle très important dans l'adoption d'une technique donnée. Nous espérerons alors que les algorithmes linéaires proposés favoriseront un peu plus la technique de traçage de cônes.

Le second point est relatif à la mauvaise gestion des textures rectangulaires. Ce problème ne concerne pas seulement la technique de traçage de cône, mais également toutes les méthodes qui utilisent la distance pendant une phase de prétraitement. Comme par exemple, le traçage de sphères ou le placage d'extrusion et de révolution. Ce problème est lié directement au pipeline graphique, qui effectue d'une manière systématique, une normalisation de l'espace texture. La solution traditionnelle effectue cette même normalisation avant de procéder aux calculs de prétraitement. Cette approche permet effectivement de résoudre le problème de distorsion, mais au prix d'une perte de performance pendant la phase de rendu. Nous avons alors adopté une nouvelle approche, qui consiste à ignorer la normalisation pendant le prétraitement, et à effectuer une correction elliptique pendant la phase de rendu. Cette approche ne ralentit pas la vitesse du rendu, car elle est effectuée une seule fois pour chaque rayon de vue. Les résultats obtenus par notre méthode sont nettement meilleurs par rapport à l'approche traditionnelle.

La technique de traçage de cônes posait également une limite concernant l'angle d'ouverture des cônes. En effet, le stockage du ratio des cônes imposait un angle maximal de $\pi/4$. Nous avons alors proposé de stocker le rayon des cônes, cela permet de porter l'angle maximal à $\pi/2$, et ainsi influer positivement sur la vitesse de convergence du traçage de cônes.

Toujours concernant les mesostructures, nous avons pu introduire une nouvelle technique spécifique aux motifs extrudés ou biseautés. En effet, ce type de mesostructures est très mal géré par les techniques existantes. Nous avons utilisé pour cela une image binaire représentant le motif de base. Puis, nous avons calculé sa transformation distance Euclidienne et son gradient. Pendant la phase de rendu, nous avons utilisé la distance pour converger rapidement vers

l'intersection entre l'axe de vue et la géométrie d'extrusion, créée à partir du motif de base. Nous avons obtenu ainsi des mesostructures très nettes avec une vitesse de rendu très rapide.

5.1.2 Modélisation et rendu à base d'images

En plus du problème de la représentation des mesostructures, le rendu 3D de scènes complexes pose un autre problème. Il s'agit du goulot d'étranglement causé par le très grand nombre de primitives graphiques envoyées au processeur graphique (polygones, point 3D, normales, coordonnées de texture…). A défaut de pouvoir réduire le nombre d'objets constituant la scène, une meilleure solution serait de pouvoir créer certains d'objets en évitant les structures maillées. Cette solution, appelée *modélisation et rendu à bases d'images*, s'appuie sur des données stockées dans des textures, qui seront par la suite extraites pendant la phase de rendu, afin de simuler l'apparence d'un objet 3D complexe.

Partant du fait que plusieurs objets architecturaux et manufacturés sont créés, totalement ou partiellement, avec des outils d'extrusion ou de révolution, nous avons pu étendre la technique du placage de motifs extrudés afin de pouvoir créer des objets entiers, obtenus par l'extrusion ou la révolution d'une forme de base, stockée sous forme binaire dans une texture. Pour cela, Nous avons créé une boite, que nous avons appelée *boite de forme*, permettant d'englober la géométrie virtuelle déduite à partir de la carte de forme. La qualité des objets créés par notre méthode est comparable à une approche maillée. Le placage de révolution est même meilleur, car la silhouette obtenue est parfaitement lisse, à la différence de la silhouette anguleuse des maillages.

5.2 Application à la reconstitution 3D du patrimoine bâti

Cette thèse a été effectuée au sein du laboratoire d'Informatique, Imagerie et Analyse Numérique (LIIAN), qui fait partie du Pôle de compétences Patrimoine Culturel (2PC). Ce pôle a pour objectif d'initier des projets scientifiques visant la sauvegarde et la valorisation de notre patrimoine. La numérisation 3D du patrimoine bâtis se trouve au cœur de cet objectif, et permettra une continuité et une mise en application des recherches effectuées durant cette thèse, notamment en ce qui concerne la simplification du maillage et les visites virtuelles. Cette partie expose les résultats d'une étude préliminaire que nous avons menée sur ce sujet.

5.2.1 Introduction

Les nouvelles technologies, dans les domaines du multimédia et des réseaux de communication, ouvrent au monde culturel de nouvelles perspectives en terme de création, de présentation, d'éducation et de partage des connaissances. Elle lui permet ainsi de renouveler et d'élargir l'accès à la culture et au patrimoine. Les avancées technologiques récentes en matière de photographie numérique et de scanner 3D, permettent aujourd'hui d'envisager, à un coût raisonnable, la réalisation de maquettes numériques 3D de grands monuments. La numérisation 3D permet de satisfaire à deux exigences:

- **La conservation et l'archivage :** L'acquisition du modèle géométrique et de son aspect visuel constituera une véritable maquette numérique pouvant être diffusée sous plusieurs formats. Cette maquette servira par la suite aux professionnels pour la gestion du monument et pour les travaux de restauration et de réhabilitation.
- **La valorisation :** Par leur capacité à attirer l'attention, les images de synthèses offriront aux monuments reconstitués une forte valeur ajoutée et un gain d'intérêt évident. Le modèle numérique permettrait par exemple, la visite d'un monument tel qu'il était lors de son édification ou la visite d'endroits inaccessibles ou interdits au public. La diffusion des données sera effectuée sur des supports variés et sous différents formats.

5.2.2 Objectifs

Trois grands objectifs sont visés par ce projet :

- La création d'une base de données nationale, constituée de monuments du patrimoine marocain et regroupant toutes les données techniques et historiques les concernant

(maquette 3D, plans, matériaux, photos, historique…). Cette base de données doit représenter une archive du patrimoine bâti, et constitueras une source de données incontournable pour tout projet ou action portant sur l'un des monuments répertoriés.

- La création d'un site web multilingue dédié aux monuments nationaux. Le visiteur doit pouvoir ainsi accéder d'une manière simple, rapide et conviviale, à toute information concernant les monuments répertoriés (histoire, images, visite virtuelle, événements, carte d'accès…). Le site sera relié aux différents organismes chargés du tourisme, de la sauvegarde du patrimoine et de l'éducation nationale.
- En génie civil, la technique du Scanner 3D de bâtiments et d'infrastructures est de plus en plus utilisée en génie civil pour les travaux de réaménagement, de consolidation et de restauration de bâtiments et d'infrastructures.

5.2.3 Processus de la reconstitution 3D

La reconstitution 3D vise la création d'une maquette numérique en trois dimensions d'un objet (bâtiment ou environnement) qu'il soit existant, partiellement ou totalement détruit, en projet de construction ou complètement imaginaire. Bien évidemment, la méthode de reconstitution dépend fortement des contextes précédemment cités.

Dans notre cas, nous nous intéressons plutôt au patrimoine bâti, ce qui nous ramène à l'étude de monuments existants qui sont soit en bonne état de conservation, soit partiellement détériorés. La démarche est souvent composée de quatre grandes étapes: L'acquisition des données, la modélisation 3D, la texturation du modèle, et enfin la représentation et la diffusion des résultats.

5.2.3.1 L'acquisition des données

C'est l'une des étapes les plus délicates, car le résultat en dépend directement. Elle est réalisée par la combinaison de différentes techniques:

- **Données existantes:** plans, cartes, dessins ou descriptions déjà établies de l'édifice (essentiellement si l'édifice est particulièrement détérioré).
- **Lasergrammétrie (Scanner 3D):** Technologie novatrice, elle repose sur le principe du balayage de l'objet à mesurer par un faisceau laser. L'émission du laser est effectuée à des fréquences élevées, permettant l'acquisition de plusieurs milliers de points dans un laps de temps très court. Il en résulte un nuage de points tridimensionnels, décrivant complètement

et précisément l'objet observé. La combinaison de plusieurs nuages permet de décrire des objets de grandes dimensions. Les nuages de points sont traités par la suite au moyen de logiciel permettant, entre autres : des mesures directes sur ces nuages, la génération de documents techniques (D.A.O), et la modélisation 3D.

- **Topographie terrestre:** Pour regrouper et géoréférencer les nuages de points obtenus, il faut poser des cibles sur l'objet à lever. Les coordonnées de ces cibles relevées par tachéométrie serviront au calcul de similitude spatiale qui permet le géoréférencement.
- **Photogrammétrie terrestre et aérienne:** Elle consiste à prendre plusieurs photographies du bâtiment sous différents angles, en se basant sur des points de calage et de géoréférencement présents et à définir sur chaque image. Les images obtenues seront projetées par la suite sur le modèle reconstruit dans la phase de texturation.

5.2.3.2 La modélisation 3D

Un modèle infographique est une version simplifiée du modèle topographique. Chaque entité architecturale est modélisée à l'aide de surfaces et d'objets réguliers (plans, cylindres, cônes, surfaces paramétriques). C'est donc un modèle beaucoup plus léger, exploitable en imagerie de synthèse. Il rend compte de la géométrie globale du monument et de ses principaux composants architecturaux à l'aide d'objets 3D simples et beaucoup plus maniables.

Cette étape demande un temps conséquent, et dépend fortement de l'architecture de l'objet à reconstruire. Elle est souvent semi-automatisée par des logiciels de post-traitement dans le cas du relevé photogrammétrique et lasergrammétrique. Cependant, elle demeure largement dépendante de l'intervention d'opérateurs humains

5.2.3.3 La texturation du modèle

La modélisation 3D permet l'obtention d'une représentation géométrique de l'objet, mais ne donne pas d'information sur son aspect visuel. Cette phase est réalisée avec la technique de texturation, qui permet de plaquer les images prises du bâtiment sur les surfaces du modèle géométrique. Cela nécessite la mise en correspondance des photographies et du nuage de points ou du modèle 3D, d'où l'importance du calibrage des caméras. Dans le cas ou les photographies ne sont pas disponibles, les textures seront également reconstituées en se basant sur les informations acquises sur l'aspect visuel de l'édifice.

5.2.3.4 La représentation et la diffusion du modèle

En plus du nuage de points, du modèle géométrique et du modèle texturé, plusieurs représentations peuvent être déduites, comme par exemple :

- Relevés topographiques, cartographies d'élévations, de façades, de coupes…
- Simulations d'éclairages et de travaux de restauration.
- Simulation des hypothèses établies sur l'état original de l'édifice.
- Images et animations à haute définition.
- Modèle simplifié permettant des visites virtuelles interactives.

La diffusion de ces différentes représentations pourrait se faire sur support multimédia, sur base de données accessible via un serveur réseau, ou via des sites Internet dédiés.

5.2.4 Impact du projet

La numérisation 3D du patrimoine bâti aura certainement un grand impact sur plusieurs secteurs d'activités, citons notamment :

- **La sauvegarde du patrimoine** : outre l'aspect conservation et archivage numérique, le projet pourra servir aux professionnels pour la gestion des monuments et pour les travaux de restaurations et de réhabilitations.
- **Le tourisme** : Les images de synthèses offriront aux monuments reconstitués une forte valeur ajoutée et un gain d'intérêt évident, ce qui aura un impact direct sur le coté attractif de notre pays, et contribuera à appuyer la politique gouvernementale en matière de tourisme
- **L'éducation** : Ce projet contribuera au renouvellement et à l'élargissement de l'accès à la culture et au patrimoine. Il offrira certainement aux enseignants, des outils de formation plus ludiques et plus conviviaux.
- **Le renforcement du contenu multimédia** : le gouvernement a adopté récemment une politique visant à appuyer l'élargissement du contenu multimédia du royaume. Ce projet s'inscrit clairement dans cette voie et pourra susciter d'autres initiatives similaires.

BIBLIOGRAPHIE

[BD06a] L. Baboud, and X. Décoret, Realistic Water Volumes in Real-Time, *In Proc. of Euroraphics*, 2006.

[BD06b] L. Baboud, and X. Décoret, Rendering Geometry with Relief Textures, *In Proc. of Graphics Interface*, Vol. 137, pp. 195-201, 2006.

[Bid90] H. B. Bidasaria, Ray Tracing Surfaces of Revolution Using a Simplified strip Tree Method (Abstract), *In Proc of the ACM annual conference on Cooperation CSC'90*, p. 427, 1990.

[BJ01] G. Baciu, and G. Jinyuan Jia Lam, Ray Tracing Surfaces of Revolution: An Old Problem with a New Perspective, *In Proc. of Computer Graphics International 2001*. pp. 215-222, 2001

[Bli77] J.F. Blinn, Models of Light Reflection for Computer Synthesized Pictures. *In Proc of SIGGRAPH'77*, pp. 192-198, 1977.

[Bli78] J.F. Blinn, Simulation of wrinkled surfaces, *In Proc. of SIGGRAPH'78*, pp. 286-292, 1978.

[BN76] J.F. Blinn, and M. E. Newell, Texture and Reflection in Computer Generated Images, *Communications of the ACM*, Vol. 19(10), pp. 542-547, 1976.

[BPMG08] C. Bosch, X. Pueyo, S. Mérillou, and D. Ghazanfarpour, A Resolution Independent Approach for the Accurate Rendering of Grooved Surfaces, *Computer Graphics Forum*, Vol. 27(7), pp. 1937-1944, 2008.

[BT04] Z. Brawley, and N. Tatarchuk, Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing, *ShaderX3*, pp. 135-154, 2004.

[Cat74] E.E. Catmull, A Subdivision Algorithm for Computer Display of Curved Surfaces, *PhD thesis, University of Utah*, 1974

[CMRS98] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno, A General Method for Preserving Attribute Values on Simplified Meshes, *In Proc. of IEEE Visualization'98*, pp. 59-66, 1998.

[Coo84] R.L. Cook, Shade Trees, *In Proc. of SIGGRAPH'84*, pp. 23-231, 1984.

[CTW+04] Y. Chen, X. Tong, J. Wang, S. Lin, B. Guo, and H.Y. Shum, Shell Texture Functions, *In Proc. of SIGGRAPH'04*, Vol. 23, pp. 343-353, 2004

[Dan80] P.E. Danielsson, Euclidean Distance Mapping, *Computer Graphics and Image Processing*, Vol. 14, pp. 227-248, 1980.

[DH00] M. Doggett, and J. Hirche. Adaptive View Dependent Tessellation of Displacement Maps, *In SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pp. 59-66, 2000.

[DLP05] J.F. Dufort, L. Leblanc, and P. Poulin, Interactive Rendering of Mesostructure Surface Details Using Semi-Transparent 3d Textures, *In Proc. of Vision, Modeling and Visualization*, pp. 399-406, 2005.

[Don04] W. Donnelly, Per-pixel Displacement Mapping with Distance Functions; *GPU Gems 2, Addison-Wesley*, 2004.

[Dum06] J. Dummer, Cone Step Mapping: An Iterative Ray-Heightfield Intersection Algorithm, *Technical Report*, Available online at: www.lonesock.net/files/ConeStepMapping.pdf, 2006.

[DvGNK99] K.J. Dana, B. van Ginneken, S.K. Nayar, and J.J. Koenderink, Reflectance and Texture of Real-World Surfaces, *ACM Transactions on Graphics*, Vol. 18(1), pp. 1-34, 1999.

[EBAB05] F.J. Espino, M. Boo, M. Amor, and J.D. Bruguera, Adaptive Tessellation of Bezier Surfaces Based on Displacement Maps, *In Proc. of WSCG'05*, pp. 29-32, 2005.

[ERSW98] I. Ernst, H. Russeler, H. Schulz, and O. Wittig, Gouraud Bump Mapping, *In Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp. 47-53, 1998.

[FB81] M.A. Fischler, and R.C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, *Communication of the ACM*, Vol. 24(6), pp. 381-395, 1981.

[Gat03] J. Gath, Derivation of the Tangent Space Matrix, *Blacksmith Studios*, 2003, Available online at: www.blacksmith-studios.dk/projects/downloads/tangent_matrix_derivation.php

[GH99] S. Gumhold, and S. Huttnert, Multiresolution rendering with displacement mapping, *In Eurographics Workshop on Graphics Hardware*, pp. 55-66, 1999.

[HDKS00] W. Heidrich, K. Daubert, J. Kautz, and H. Seidel, Illuminating Micro Geometry Based on Precomputed Visibility, *In Proc. of SIGGRAPH'00*, pp. 455-464, 2000.

[HEGD04] J. Hirche, A. Ehlert, S. Guthe, and M. Doggett, Hardware Accelerated Per-Pixel Displacement Mapping, *In Proc. of Graphics Interface 2004*, vol. 62, pp. 153-158, 2004.

[HM91] P.S. Heckbert, and H.P. Moreton, Interpolation for Polygon Texture Mapping and Shading, *State of the Art in Computer Graphics: Visualization and Modeling*, pp. 101-111, 1991.

[HS88] C. Harris, and M. Stephens, A Combined Corner and Edge Detector, *In Proc. of the 4th Alvey Vision Conference*, pp. 147-151, 1988.

[JMW07] S. Jeschke, S. Mantler, and M. Wimmer, Interactive Smooth and Curved Shell Mapping, *Rendering Techniques 2007, Proc. of the Eurographics Symposium on Rendering*, pp. 351-360, 2007.

[JTJ04] J. Jia, K. Tang, and A. Joneja, Biconic Subdivision of Surfaces of Revolution and its Applications in Intersection Problems, *The Visual Computer*, Vol. 20(7), pp. 457-478, 2004.

[JWP05] S. Jeschke, M. Wimmer, and W. Purgathofer, Image-Based Representations for Accelerated Rendering of Complex Scenes, *In Proc. of EUROGRAPHICS'05*, pp. 1-20, 2005.

[Kaj83] J.T. Kajiya, New techniques for ray tracing procedurally defined objects, *ACM Transactions on Graphics (TOG)*, Vol. 2(3), pp. 161-181, 1983.

[Kaj86] J.T. Kajiya, The Rendering Equation, *In Proc of SIGGRAPH'86*, pp. 143-150. 1986.

[Kil00] M.J. Kilgard, A Practical and Robust Bump-mapping Technique for Today's GPUs, *In Game Developers Conference GDC'00*, 2000.

[KLP01] J. S. Kim, J. H. Lee, and K. H. Park, A Fast and Efficient Bump Mapping Algorithm by Angular Perturbation, *Computers and Graphics*, Vol. 25, pp. 401-407, 2001.

[KO07] H. Ki, and K. Oh, Accurate Per-Pixel Displacement Mapping using a Pyramidal Structure, *Technical Report 2007*, available online at: http://ki-h.com/archive/KiH-TA07-IPDM.pdf.

[KRS05] A. Kolb, and C. Rezk-Salama, Efficient Empty Space Skipping for Per-Pixel Displacement Mapping. *In Vision, Modeling and Visualization*, 2005.

[KTI+01] T. Kaneko, T. Takahei, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda, and S. Tachi, Detailed Shape Representation With Parallax Mapping, *In Proc. of the ICAT'01*, pp. 205-208, 2001.

[LB05] C. Loop, and J. Blinn, Resolution Independent Curve Rendering using Programmable Graphics Hardware, *ACM Transactions On Graphics*, Vol. 22(3), pp. 1000-1009, 2005.

[Len04] Eric Lengyel, Mathematics for 3D Game Programming & Computer Graphics (Second Edition), *Charles River Media*, 2004.

[LKG+03] H.P.A. Lensch, J. Kautz, M. Goesele, W. Heidrich, and H.P. Seidel, Image-Based Reconstruction of Spatial Appearance and Geometric Detail. *ACM Transactions On Graphics*, Vol. 22(2), pp. 234-257, 2003.

[LMH00] A. Lee, H. Moreton, and H. Hoppe, Displaced Subdivision Surfaces, *In Proc. of the 27th annual conference on Computer graphics and interactive techniques*, pp. 85-94, 2000.

[LPLY07] S.G. Lee, W.C. Park , W.J. Lee, S.B. Yang, and T.D. Han, An Effective Bump Mapping Hardware Architecture Using Polar Coordinate System, *Journal of Information Science and Engineering*, Vol. 23(2), pp. 569-588, 2007.

[LQ02] M. Lhuillier, and L. Quan, Quasi-Dense Reconstruction from Image Sequence, *Proceedings of the 7th European Conference on Computer Vision-Part II ECCV'02*, pp.125-139, 2002.

[Max88] N. Max, Horizon mapping: shadows for bump-mapped surfaces, *The Visual Computer*, Vol. 4(2), pp. 109-117, 1988.

[McM97] L. McMillan, An Image-based Approach to Three-dimensional Computer Graphics, *PhD. Dissertation. University of North Carolina at Chapel Hill*, 1997.

[MGW01] T. Malzbender, D. Gelb, and H. Wolters, Polynomial Texture Maps, *In Proc. of SIGGRAPH'01*, pp. 519-528, 2001.

[MJW07] S. Mantler, S. Jeschke, and M. Wimmer, Displacement Mapped Billboard Clouds, *Technical Report, Institut für Computergraphik und Algorithmen, TR-186-2-07-01*, 2007.

[MM05] M. McGuire, and M. McGuire, Steep Parallax Mapping, *In proc. of I3D'05 Poster*, 2005.

[Mor77] J.J. More, The Levenberg-Marquardt Algorithm: Implementation and Theory, Numerical Analysis, *G.A. Watson, editor. Lecture Notes in Mathematics 630, Springer-Verlag*, 1977.

[MS04] J.L. Mitchell, and P.V. Sander, Applications of Explicit Early-Z Culling, *Real-Time Shading Course, SIGGRAPH'04*, 2004.

[NH08] D. Nehab, and H. Hoppe. Random-Access Rendering of General Vector Graphics. *ACM Transactions on Graphics*, Vol. 27(5), pp. 1-10, 2008.

[NRH+77] F.E. Nicodemus, J.C. Richmond, J.J. Hsia, I.W. Ginsberg, and T. Limperis, Geometric Considerations and Nomenclature for Reflectance, *Monograph 161, National Bureau of Standards* (*US*), 1977.

[OBM00] M.M. Oliveira, G. Bishop, and D. McAllister, Relief Texture Mapping, *In Proc. of SIGGRAPH'00*, pp. 359-368, 2000.

[OKL06] K. Oh, H. Ki, and C.H. Lee, Pyramidal Displacement Mapping: A GPU-Based Artifacts-Free Ray Tracing Through an Image Pyramid, *ACM Symposium on Virtual Reality Software and Technology* (*VRST'06*), pp. 75-82, 2006.

[Oli00] M.M. Oliveira, Relief Texture Mapping, *PhD thesis, University of North Carolina*, 2000.

[OP05] M.M. Oliveira, and F. Policarpo, An Efficient Representation for Surface Details, *UFRGS technical report RP-351*, 2005.

[PAC97] M. Peercy, J. Airey, and B. Cabral, Efficient Bump Mapping Hardware, *In Proc. of SIGGRAPH'97*, pp. 303-306, 1997.

[PBFJ05] S.D. Porumbescu, B. Budge, L. Feng, and K.I. Joy, Shell Maps, *In Proc. of SIGGRAPH'05*, Vol. 24(3), pp. 626-633, 2005.

[Pet02] S. Petitjean, A Survey of Methods for Recovering Quadrics in Triangle Meshes. *ACM Computing Surveys*, Vol. 34(2), pp. 1-61, 2002.

[PHL91] J.W. Patterson, S.G. Hoggar, and J.R. Logie, Inverse Displacement Mapping, *Computer Graphics Forum*, Vol. 10(2), pp. 129-139, 1991.

[PO06] F. Policarpo, and M.M. Oliveira, Relief Mapping of Non-Height-Field Surface Details, *In Proc. of the 2006 Symposium on Interactive 3D Graphics and Games I3D'06*, pp. 55-62, 2006.

[PO07] F. Policarpo, and M.M. Oliveira, Relaxed Cone Stepping for Relief Mapping, *GPU Gems 3*, pp. 409-428, 2007.

[POC05] F. Policarpo, M.M. Oliveira, and J.L.D. Comba, Real-Time Relief Mapping on Arbitrary Polygonal Surfaces, *In Proc. of the Symposium on Interactive 3D Graphics and Games I3D'05*, pp. 155-162, 2005.

[PP94] D.W. Paglieroni, and S.M. Petersen, Height Distributional Distance Transform Methods for Height Field Ray Tracing, *ACM Transactions on Graphics* (*TOG*), Vol. 13(4), pp. 376-399, 1994.

[Pre06] M. Premecz, Iterative Parallax Mapping with Slope Information, *Central European Seminar on Computer Graphics CESCG'06*, Available online at: www.cescg.org/CESCG-2006/papers/ TUBudapest-Premecz-Matyas.pdf, 2006.

[PZ08] E. Parilov, and D. Zorin, Real-Time Rendering of Textures with Feature Curves. *ACM Transactions on Graphics*, Vol. 27(1). pp. 1-15, 2008.

[QMK08] Z. Qin, M. McCool, and C. Kaplan, Precise Vector Textures for Real-Time 3D Rendering. *Symposium on Interactive 3D Graphics and Games I3D'08*, pp. 199-206, 2008.

[Rit06] N. Ritsche, Real-Time Shell Space Rendering of Volumetric Geometry, *In Proc of GRAPHITE'06*, pp. 265-274, 2006.

[RSP06] E.A. Risser, M.A. Shah, and S. Pattanaik, Interval Mapping, *Symposium on Interactive 3D Graphics and Games* (*I3D'06 Poster*), 2006.

[SC00] P. Sloan, and M.F. Cohen, Interactive Horizon Mapping, *Eurographics Workshop on Rendering*, vol. 2, pp. 281-286, 2000.

[SKU08] L. Szirmay-Kalos, and T. Umenhoffer, Displacement Mapping on the GPU - State of the Art, *Computer Graphics Forum*, Vol. 27(6), pp. 1567-1592 *,* 2008.

[ST90] P. Shirley, and A. Tuchmann, A polygonal Approximation to Direct Scalar Volume Rendering, *ACM Computer Graphics*, Vol. 24, pp. 63-70, 1990.

[STM04] P. Sander, N. Tatarchuk, and J.L. Mitchell, Explicit Early-Z Culling for Efficient Flow Simulation and Rendering, *ATI Research Technical Report*, 2004.

[Tat06a] N. Tatarchuk, Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows, *In Proc. of I3D'06*, pp. 63-69, 2006.

[Tat06b] N. Tatarchuk, Practical Parallax Occlusion Mapping with Approximate Soft Shadows for Detailed Surface Rendering, *ShaderX5*, pp. 75-105, 2006.

[TI07] I. Tisevich, and A. Ignatenko, Displacement and Normal Map Creation for Pairs of Arbitrary Polygonal Models Using GPU and Subsequent Model Restoration, *In Proc. of GraphiCon'2007*, pp. 61-68, 2007.

[TIS08] A. Tevs, I. Ihrke, and H.P. Seidel, Maximum Mipmaps for Fast, Accurate, and Scalable Dynamic Height Field Rendering, *In Proc. of Interactive 3D Graphics and Games*, pp. 183-190, 2008.

[TL08] R. de Toledo, and B. Levy, Visualization of Industrial Structures with Implicit GPU Primitives, *International Symposium on Visual Computing ISVC'08*, pp. 139-150, 2008.

[VT04] M.A.O. Vasilescu, and D. Terzopoulos, Tensor Textures: Multilinear Image-Based Rendering, *In Proc of SIGGRAPH'04*, pp. 336-342, 2004

[Wel04] T. Welsh, Parallax Mapping with Offset Limiting: A Per-Pixel Approximation of Uneven Surfaces, *Technical Report, Infiscape Corp.*, 2004.

[WTL+04] X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.Y. Shum. Generalized Displacement Maps, *In Proc. of the Eurographics Symposium on Rendering*, pp. 227-234, 2004.

[WWT+03] L. Wang, X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.Y. Shum, View-Dependent Displacement Mapping, *ACM Trans. Graphics*, vol. 22(3), pp. 334-339, 2003.

[YJ04] K. Yerex, and M. Jagersand, Displacement Mapping with Ray-Casting in Hardware, *In Proc. of SIGGRAPH'04 Sketches*, 2004.

[ZTCS99] R. Zhang, P.S. Tsai, J.E. Cryer, and M. Shah, Shape from Shading: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 21(8), pp. 690-706, 1999.

LEXIQUE FRANÇAIS/ANGLAIS

Auto-ombrage	Self-shadowing
Axe/Rayon de vue	View Ray
Biseau	Bevel
Captures d'écran	Screenshots
Carte de hauteurs	Height map
Carte de profondeurs	Depth map
Chanfrein	Chamfer
Cônes conservatifs	Conservative cones
Cônes relaxés	Relaxed cones
Coordonnées de texture	Texture coordinates
Coquille	Shell
Coupe	Slice
Crénelage / Anti-crénelage	Aliasing / Anti-aliasing
Densification (maillage)	Tessellation
Détermination de visibilité	Culling
Espace homogène de fenêtrage	homogeneous clip space
Extrusion étendue	Offset extrusion
Forme / Carte de forme / Boite de forme	Shape / Shape Map / Shape Box
Image à niveaux de gris	Grayscale/Greyscale image
Images Par Seconde IPS	Frame Per Second FPS
Lancer de rayons	Raytracing/Raycasting
Maillage	Mesh
Niveau / Lignes de niveau	Level / Level set
Nuanceur	Shader
Nuanceur de géométries	Geometry shader
Nuanceur de pixels	Pixel shader / Fragment shader
Nuanceur de sommets	Vertex shader
Ombrage	Shading
Parallaxe	Parallax
Pente	Slope
Placage de bosselures	Bump mapping
Placage de déplacement	Displacement mapping
Placage de textures	Texture mapping
Prétraitement	Pre-processing
Pyramide tronquée	frustum
Rastérisation	Rasterization / Rasterisation
Rendu différé	offline rendering
Rendu temps réel	Real-time rendering

Révolution	Revolution - Lathe
Sélection de faces	face culling
Sommet / sommets	Vertex / Vertices
Tampon	Buffer
Taux de rafraîchissement	Framerate
Test ciseaux	Scissor test
Traçage de cônes / sphères / cylindres	Cone / sphere / cylinder tracing
Traçage de rayons	Raytracing / Ray-marching

PUBLICATIONS

Journaux internationaux

Akram Halli, Abderrahim Saaidi, Khalid Satori, and Hamid Tairi, **Extrusion and Revolution Mapping,** *ACM Transactions on Graphics (ACM TOG),* ISSN 0730-0301, accepted for publication pending revisions, 2009.

Akram Halli, Abderrahim Saaidi, Khalid Satori, and Hamid Tairi., **Per-Pixel Extrusion Mapping**, *International Journal of Computer Science and Network Security (IJCSNS),* ISSN 1738-7906, Vol. 9(3), pp. 118-124, March 2009.

Akram Halli, Abderrahim Saaidi, Khalid Satori, and Hamid Tairi, **Per-Pixel Displacement Mapping Using Cone Tracing**, *International Review on Computers and Software (IReCoS), Praise Worthy Prize Editor,* ISSN 1828-6003, Vol. 3(5), September 2008.

Conférences internationales avec comité de lecture

Akram Halli, Khalid Satori, Hamid Tairi, and Abderrahim Saaidi, **Relief Mapping with Real Time Depth Scaling Support**, *In Proceeding of Information and Communication Technologies International Symposium ICTIS'07, IEEE Morocco section*, ISBN 9954-8577-0-2, pp. 476-479, April 2007, Fez, Morocco.

Conférences nationales

Khalid Satori, Akram Halli, Hamid Tairi et Abderrahim Saaidi, **Reconstitution 3D du Patrimoine Architectural,** *Recherche sur le patrimoine: état des lieux et perspectives, Journée d'étude organisée par le Pole de compétences Patrimoine Culturel (2PC),* 2006, Fès, Maroc.

Akram Halli, Abderrahim Saaidi, Khalid Satori, Hamid Tairi, **Placage de relief pour le rendu temps réel**, *Compte-rendu du Workshop sur les Technologies de l'Information et de la Communication (WOTIC'07),* pp. 136-139, juillet 2007, Rabat, Maroc.