REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences et Sciences de l'ingénieur
Département d'informatique

Mémoire en vue de l'obtention du diplôme de

Magister en Informatique

Option : Intelligence Artificielle et Images

Rendu Volumique efficace par une représentation à base de couches d'images

Réalisé par :
Djalel HEMIDI

Soutenue devant le jury composé de :

Batouche Mohamed Chaouki	Professeur	Université de Constantine	Président
Djedi Noureddine	Maître de conférence	Université de Biskra	Rapporteur
Kholladi M^{ed} Khiereddine	Maître de conférence	Université de Constantine	Examinateur
Bellatar Brahim	Maître de conférence	Université de Batna	Examinateur
Boukerram Abdellah	Chargé de cours (Docteur N.T.)	Université de Sétif	Examinateur

2004/2005

A mes parents, ma famille et à tout mes amis …

A tous ceux qui adorent le savoir …

Remerciements

« Avant tout chose, je remercie Dieu de m'avoir donner toute cette force, qui m'à permit d'arriver à bon port …»

Je tiens à exprimer mes vifs remerciements au Dr. DJEDI Nour Eddine d'avoir accepté d'être le rapporteur de ce mémoire, pour m'avoir bien suivi durant la réalisation de ce travail, pour ces conseils et ses remarques qui m'ont aidé à élaborer et améliorer ce mémoire.

Je remercie également Mr. BABHENINI M^{ed} Chaouki, pour son aide, pour ses orientations et ses remarques qui m'ont permet d'améliorer les nouveaux résultats à chaque étape de développement de ce projet …

Je tiens aussi de dire merci à Mr. Alexandre MEYER pour avoir répondu à mes questions, pour ses orientations et ses conseils …

Mes sincères remerciements s'adressent à tous les enseignants du département d'informatique, qui m'ont formé durant la période d'étude à l'université Mohamed Khider – Biskra, surtout Mr. CHERIF Foudil et Mr. ZERARKA M^{ed} Faouzi …

Mes vifs remerciements s'adressent également, au président et aux membres de jury pour avoir consacré de leurs temps si précieux, un moment d'attention vis-à-vis de ce projet.

Merci, à tous …

Djalel Hemidi.

Table des matières

Table des figures

Introduction générale

La reproduction de la réalité est l'un des objectifs de la synthèse d'images. Le terme de *photo-réalisme* décrit des techniques et des formes d'art qui tentent de créer des images de synthèse pouvant être confondues à des images réelles. Dans ce cadre, le *réalisme* d'une image est dépendant de la précision avec laquelle on définit la scène qui la représente, ainsi que de la précision avec laquelle on simule la propagation de la lumière. En d'autres termes, pour obtenir une image visuellement réaliste d'une scène, il faudra, d'une part, la définir le plus précisément possible et d'autre part, appliquer les modèles physiques de propagation de la lumière les plus performants et les plus complets.

L'une des caractéristiques du monde réel, est sa complexité due à une multitude de détails présents dans la nature. Cette complexité est caractérisée par la variété et les différentes échelles des motifs présents dans une scène à représenter. Ainsi, si l'on souhaite modéliser une scène où intervient une forme de complexité en utilisant les outils graphiques de modélisation des objets simples, on se trouve confronté à un certain nombre de problèmes pratiques :

- Au moment de la modélisation, il y'a lieu de donner, de façon répétitive, les informations permettant de générer ces motifs, alors qu'en évidence nous avons besoin d'une méthode générale et intuitive facilitant ce processus.
- En terme d'espace mémoire utilisé, l'utilisateur s'apercevra que la taille occupée par les modèles associés est excessivement élevée.
- Le processus de rendu va présenter une lenteur caractérisée à cause du nombre très grand de formes géométriques présentes dans la scène.

L'une des solutions intéressantes, apportée pour pallier ces problèmes est l'utilisation des textures volumiques, introduites en 1989 par *Kajiya* et *Kay* et dont l'objectif initial était de simuler la fourrure. Ce modèle, généralisé par la suite par *F. Neyret* en 1996, génère de l'intérêt du fait qu'il représente une approche pertinente permettant de résoudre le problème de représentation de la complexité, surtout dans sa forme répétitive. *Neyret* a déployé tout ce que la méthode avait de potentiel pour la rendre plus générale, plus complète et plus rapide.

Malgré toutes les avancés que les textures volumiques ont apportées, et malgré la monté en puissance des machines de calcul spécialisées telles que les stations graphiques, le problème du temps pour avoir un résultat final qui reste relativement élevé. De plus, l'implantation de ce modèle sur une machine individuelle pose, en plus, le problème de l'espace mémoire utilisé pour stocker un volume de référence à haute résolution.

La solution apportée à ce problème est l'utilisation d'une représentation alternative du volume de référence, la texture volumique à base de couches d'images, qui permet de profiter des performances

du matériel graphique. La méthode a était inspirée, par *Meyer*, à partir des travaux de *Lacroute* et *Levoy* pour la résolution du problème du temps très élevé pour l'obtention d'un rendu volumique classique. Elle consiste à créer un volume de référence en utilisant des couches d'images représentées comme étant des polygones transparents texturées. Ainsi, le rendu de la surface texturée consiste à afficher ces polygones décalés à la surface de chaque boîte constituant un élément épais de la surface à habiller.

Organisation du mémoire.

Après cette introduction, le corps de ce mémoire se décompose en quatre chapitres. Le premier chapitre expose un ensemble de techniques basées sur des images pour effectuer l'étape de rendu dans des temps raisonnables, surtout celles qui sont classées comme des représentations alternatives au approches classiques, où intervient la technique des textures volumiques interactives de *Meyer* qui est la base de notre travail.

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

Le deuxième chapitre présente, donc, une description détaillée de l'étape de modélisation d'une surface et d'un volume de référence par cette dernière approche, en plus d'une nouvelle méthode permettant d'avoir un ensemble de volumes de références à partir des images de synthèse avec ou sans l'utilisation de l'information de profondeur de chaque pixel.

Le troisième chapitre traite l'étape de rendu d'une texture volumique basée sur cette nouvelle représentation. Nous allons présenter, ainsi, la technique utilisée pour rendre une telle représentation en utilisant la mémoire vidéo comme un support de stockage de notre volume de référence représenté en tant que couches d'images, et les optimisations apportées en introduisant la notion de multi-résolution.

Le quatrième chapitre décrit l'ensemble des résultats générés en se basant sur cette nouvelle technique. Ce chapitre se termine par un ensemble de perspectives à atteindre dans des travaux futurs.

La conclusion générale récapitule cette représentation et met en évidence un ensemble de nouvelles voies à exploiter ultérieurement pour enrichir ce travail.

Le mémoire se termine par une série d'annexes :

- L'annexe A présente l'ensemble des techniques qu'essayent de traiter la complexité, géométrique ou temporelle, que se soit les approches classiques ou celles appelées émergentes.
- L'annexe B rassemble tous les détails concernant la vision 3D sur lesquels se base la technique de coloration de voxels utilisée comme un outil permettant d'avoir une nouvelle technique de génération d'un volume de référence à base de couches d'images.

- L'annexe C décrit les étapes utilisées pour charger une image, par le modèle de la texture volumique, en mémoire vidéo, et les fonctions permettant par la suite de rendre (afficher) une image chargé dans la mémoire vidéo à un emplacement spécifique dans l'espace 3D.

Les contributions de cette nouvelle approche concernent les améliorations techniques apportées au modèle original des textures volumiques :

- L'utilisation des images comme des couches constituant un volume de référence, au lieu d'un arbre octale, ce qui permet de générer des volumes de référence plus fins - résolution plus grande donnant par conséquent plus de détails pour des motifs plus proches - sans avoir à traiter les problèmes posés par des machines individuelles concernant l'utilisation d'espace de travail dans une mémoire.
- L'exploitation des capacités des cartes graphiques en utilisant la bibliothèque graphique normalisée *OpenGL* permet d'avoir des temps de calcul très raisonnables et une fréquence d'affichage très grande, ce qui permet d'atteindre le temps réel.

Techniques de rendu basées sur les couches d'images : Une vue générale

1. Introduction.

Plusieurs axes novateurs dans le placage de texture sont attribuables aux travaux classiques de *Catmull*, *Blinn* et *Newell* [OLI00, HQU00]. Dans sa forme de base, le placage de textures place une image (la texture) sur un objet dans une scène. Toutes les approches de placage de texture comportent deux étapes :

- Le placage de la texture partant de l'espace de texture vers l'espace écran, et l'échantillonnage pour éviter l'aliassage.
- Après que le placage ait été effectué et la texture déformée, l'image doit être re-échantillonnée sur la grille de l'écran.

Les techniques de placage de textures comptent sur des fonctions de placage pour spécifier le rapport des coordonnées spatiales d'image de la texture avec leurs positions correspondantes sur un modèle tridimensionnel. La spécification de ce placage est difficile, consomme du temps et exige souvent une intervention humaine considérable. En conséquence, les méthodes de placage utilisées généralement sont limitées aux descriptions géométriques très simples, comme des facettes polygonales, des sphères et des cylindres [LEO97, HQU00].

Récemment, des techniques de modélisation et de rendu basées sur l'image (IBMR) ont gagné une attention considérable à cause de leur potentiel de création d'images très réalistes. L'un des bénéfices principaux de ces techniques est la capacité de capturer des effets subtils du monde réel et des détails liés aux imperfections du monde réel que les chercheurs graphiques ne savent pas comment modéliser et rendre. En utilisant des images[1] comme primitives de modélisation et de rendu, ces approches peuvent aider à résoudre deux problèmes importants et permanents dans l'infographie [OLI00] :

- Le besoin de techniques de modélisation plus simples appropriées à la représentation de scènes complexes,
- et le besoin de l'accélération de rendu.

Le premier point peut être atteint en remplaçant des modèles conventionnels (géométriques) avec des représentations basées sur l'image. L'accélération de rendu est obtenue en séparant le temps de rendu de la complexité de la scène et en ré-échantillonnant des images pré-ombrées.

[1] Les images peuvent être prises du monde réel ou rendues d'objets synthétiques.

Malgré leurs potentiels, les techniques de rendu basées sur l'image (IBR)[2] ne sont qu'à leur début et plusieurs défis doivent toujours être surmontés. Ce travail présente une classification des dernières méthodes de rendu basé sur l'image, et discute leurs forces et leurs limitations.

La structure de la classification est organisée autour du diagramme présenté dans la figure 1.1. Les techniques sont classifiées selon leurs positions relatives le long du spectre géométrie/image (indiqué par un point) [OLI00].

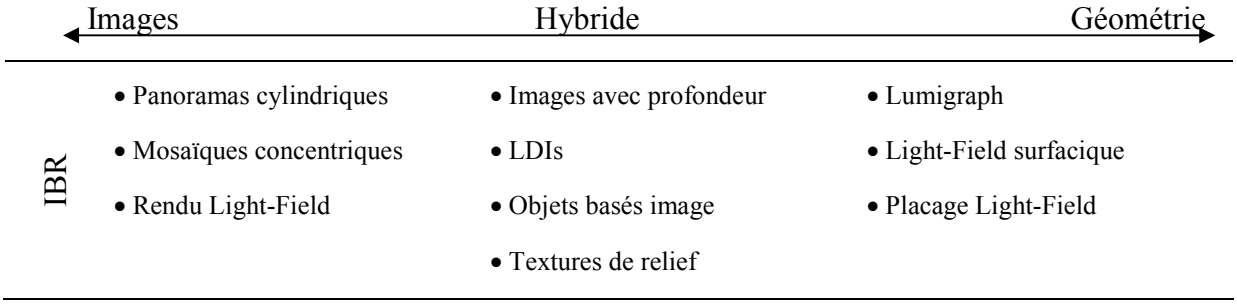


Figure 1.1 : Quelques techniques de rendu basées sur l'image sur le spectre géométrie/image.

2. Techniques de rendu basées purement sur l'image.

Les techniques IBR pures utilisent seulement quelques images de scènes pour le rendu de nouvelles vues. Les échantillons de l'environnement sont capturés comme un ensemble de photographies (ou des séquences vidéo) puis ré-échantillonnés pendant le rendu. Dans ce cas, la modélisation 3D n'est pas exigée et la vitesse de rendu n'est pas affectée par la complexité de la scène [OLI00].

2.1. Movie-Maps.

Les *movie-maps* ont été introduites en 1980 par *Lippmann* [LEO97, OLI00, POR04]. Le principe de cette approche est de créer une base de données de milliers d'images de la scène (images de référence) et de les stocker sur un support interactif. Une méthode spéciale permet d'accéder à un de ces points de vue en fonction du point de vue virtuel (l'observateur) et d'afficher l'image la plus proche dans la base de données. Ainsi, le processus de rendu est remplacé par une interrogation de base de données dans un ensemble énorme d'images de référence. Le système pourrait aussi accommoder une simple rotation, inclinaison, ou changement d'échelle à partir de ces positions d'observation fixées. L'utilisateur peut contrôler sa vitesse et son chemin parmi ceux autorisés.

Le *Movie-Map* [LEO97] était incapable de reconstruire toutes les vues désirables possibles. Même avec les capacités de stockage énormes actuellement disponibles comme les disques laser et malgré le développement rapide des médias de stockage de plus haute capacité, l'espace de stockage de toutes les images désirables possibles apparaît si grand que n'importe quelle approche purement orientée base de données continuera à être peu pratique dans le proche avenir.

2.2. Imposteurs.

Quelques techniques basées sur l'image utilisent des images pour représenter une partie de la scène. Cette sorte d'images, appelées imposteurs ou *sprites*, peut être utilisées comme des textures et plaquées sur un plan pour représenter la partie réelle de la scène. Une simple image ne peut pas fournir

[2] Image Based Rendering.

la parallaxe appropriée, les effets de parallaxe génèrent, par exemple, l'apparition de certaines parties de la scène qui ne sont pas dans l'image de base (**effets d'occlusion** ou **de masquage**, voir la figure 1.2) [POR04]. Pour cela, deux extensions ont été introduites [HQU00] :

- Une approche ajoute l'information de profondeur à l'image.
- L'autre approche utilise une maille approximative sous-jacente et plaque des images sur la géométrie. Les imposteurs sont d'habitude utilisés pour accélérer le rendu de la scène géométrique.

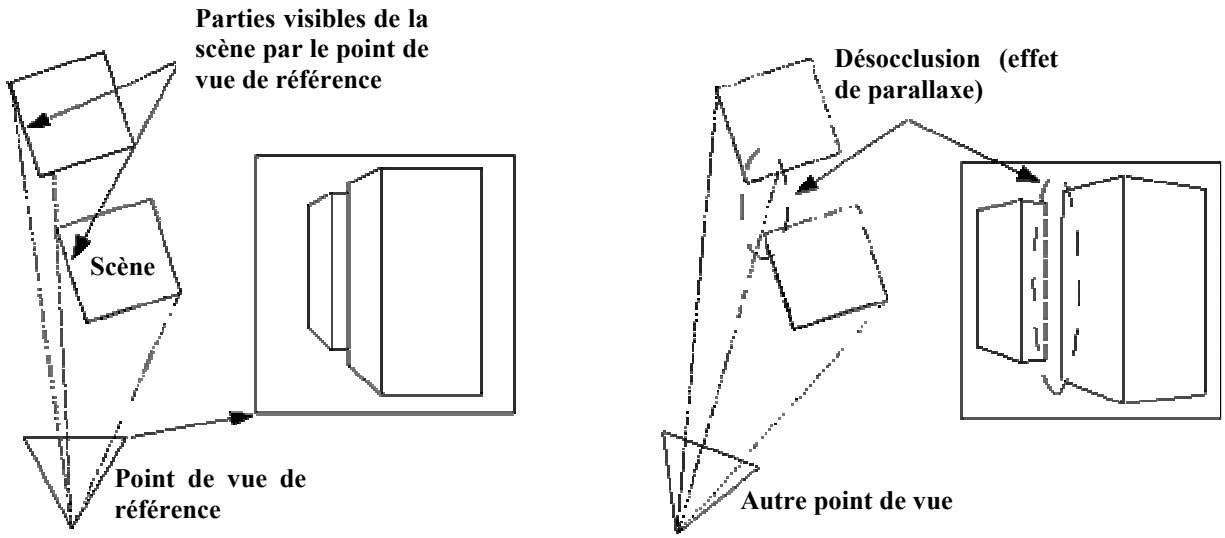


Figure 1.2 : Illustration d'un effet de parallaxe : la désocclusion. L'image de référence utilisée pour afficher la scène selon un autre point de vue ne contient pas assez d'informations et des trous apparaissent dans l'image à l'endroit où on devrait voir une partie de la scène.

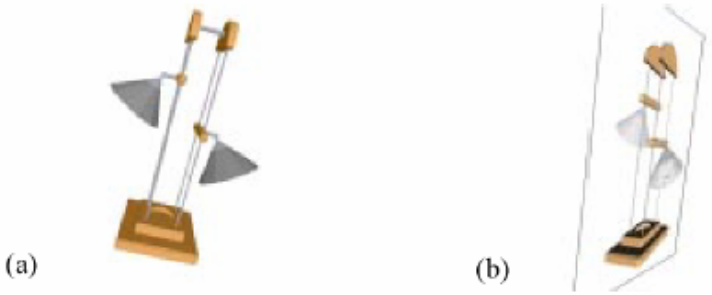


Figure 1.3 : Un objet (a) et son imposteur (b). L'imposteur est vue d'un point de vue différent que la géométrie originale.

2.2.1. Imposteurs statiques.

Maciel et *Shirley* [HQU00, POR04] ont introduit les imposteurs pour la navigation visuelle dans de grands environnements. Ils ont défini l'imposteur comme une entité qui est plus rapide à afficher que l'objet réel, mais il conserve les caractéristiques visuelles importantes des objets. Ainsi, ils remplacent des parties complexes de la scène par un modèle (une primitive) englobant simplifié et texturé permettant d'obtenir un taux d'affichage élevé et approximativement constant. En utilisant en plus la technique des niveaux de détails, ces modèles simplifiés texturés sont appelés *Imposteurs*.

2.2.2. Imposteurs dynamiques.

A la différence des imposteurs statiques, tous pré-générés, *Schaufler* a introduit les imposteurs dynamiques. L'approche utilisant des imposteurs précalculés devient rapidement inutilisable dès que le nombre d'objets augmente car l'espace mémoire nécessaire pour stocker les textures est limité. La scène et ses imposteurs peuvent être stockés dans une structure de données hiérarchique appelée "*three dimensional image cache"*. Durant le rendu, l'imposteur produit dynamiquement est utilisé pour remplacer la primitive. Pour déterminer la validité visuelle d'un imposteur, l'angle maximal en dessous duquel tous les points de la surface de l'objet sont aussi dans l'image est évalué. Tant que l'angle de vue de l'imposteur est inférieur à ce seuil, l'imposteur est considéré valide. Si le point de vue n'est pas translaté mais que seule la direction de vue change, l'imposteur reste valide. Ainsi les imposteurs peuvent être réutilisés d'une image à l'autre lors de l'animation, et ne sont régénérés que s'ils deviennent invalides, accélérant ainsi le rendu [HQU00, POR04].

Shade a proposé une méthode semblable à celle de *Schaufler*. Comme une étape de pré-traitement, on construit un BSP-Tree qui partitionne hiérarchiquement les primitives géométriques dans la scène. Durant le rendu, les images des nœuds des différents niveaux de la hiérarchie sont mises en cache pour être réutilisées dans des images suivantes. Une image est réutilisée en la plaquant sur un simple quadrilatère dessiné, qui remplace la géométrie du nœud correspondant [HQU00].

2.2.3. Nailboard et imposteur avec couches

Un système en couche ne résout pas les problèmes de visibilité en général étant donné qu'un imposteur ne peut être que soit devant, soit derrière un autre. L'utilisation de polygones transparents texturés provoque également des problèmes de visibilité car les polygones peuvent s'intersecter ou intersecter la géométrie de la scène. De plus, ces images planes ne prennent pas en compte les effets de parallaxe et d'occlusion quand elles sont vues d'un point de vue différent de celui de la capture [POR04].

Schaufler a proposé un rendu de primitives nommé *Nailboards* pour résoudre le problème de visibilité. Le *Nailboard* est un polygone sur lequel une texture RVBP (Rouge, Vert, Bleu et Profondeur) est plaquée pour représenter un objet complexe. Pour chaque pixel de la texture (texel) le composant complémentaire *P* mesure la distance entre le point de l'objet apparaissant sur ce texel et le polygone. Cette information de profondeur peut être utilisée pour composer le z-buffeur. L'image de l'objet et l'information de profondeur sont dynamiquement produits et réutilisés pour plusieurs images (*frames*) jusqu'à ce que l'erreur soit au-delà d'un seuil [HQU00].

Schaufler a également proposé une autre extension des imposteurs dynamiques appelée *imposteurs en couche (layered impostors)* pour représenter des objets. Un imposteur dynamique remplace un objet complexe par un polygone transparent sur lequel est plaquée une image de l'objet. Un imposteur en couches est composé de plusieurs de ces polygones. Tous les pixels d'une couche sont à une distance identique de l'observateur, c'est à dire que les pixels d'une image sont distribués sur plusieurs calques en fonction de leur profondeur (Figure 1.4). Comme les imposteurs en couche contiennent une profondeur (approximative), il est possible de résoudre correctement les problèmes de visibilité lors de changement de point de vue. Une mesure d'erreur visuelle identique aux imposteurs standard est utilisée pour savoir quand régénérer l'imposteur [POR04].

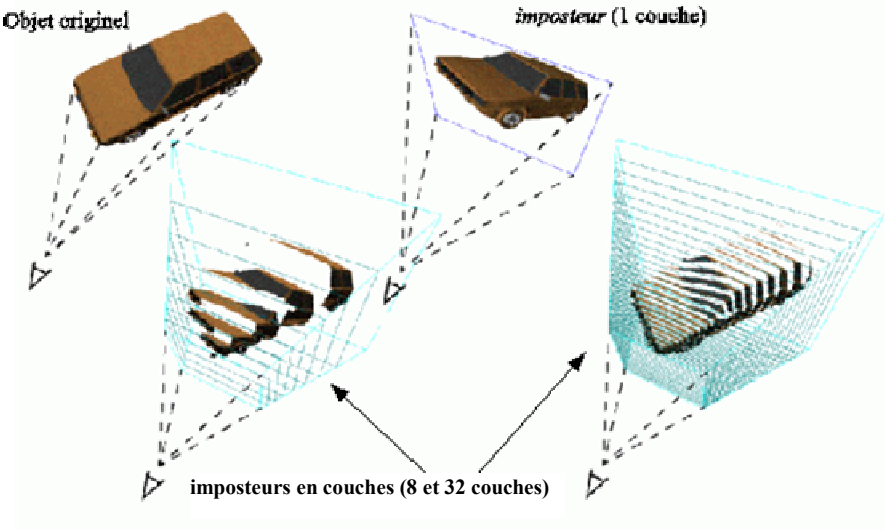


Figure 1.4 : Imposteurs en couche.

2.2.4. Imposteur maillé

Les méthodes d'imposteur précédentes plaquent une image sur un plan (quadrilatère, des polygones). Cependant, le nombre d'images pour lesquels ces imposteurs sont valables est très limité en raison des problèmes de parallaxe. Quelques chercheurs utilisent une information géométrique plus sophistiquée. Ils n'utilisent pas seulement l'image de la géométrie distante comme une texture, mais ils utilisent également un maillage triangulaire pour approximer cette géométrie. Cette méthode, appelée imposteur maillé, triangule la scène en régions approximativement plates dans une étape de pré-traitement. Durant l'exécution, la texture est plaquée sur ces triangles [HQU00].

Darsa et al. ont présenté une technique de rendu basée sur image pour la navigation dans des environnements tridimensionnel [HQU00, POR04] :

- Un ensemble de cartes d'environnement cubiques est pré-rendu à partir d'un certain nombre de points de vue fixés dans le modèle virtuel.
- Une maille polygonale est créée par triangulation des six cartes de profondeurs associées ainsi obtenues.
- Une vue prise d'un point de vue arbitraire peut être reconstruite par le placage de texture sur les maillages créés.
- Des cartes d'environnement multiples avec leurs mailles triangulaires sont utilisées pour remplir les trous dans les images en combinant les images de profondeur issues de la structure.

Sillion et al. ont utilisé la technique d'imposteur pour la visualisation en temps réel de paysages urbains. Le modèle urbain est segmenté en deux sous-ensembles distincts sur chaque *frame* : Le voisinage local et le paysage distant.

- Le voisinage local est rendu d'une façon normale.
- Le paysage distant est remplacé par des imposteurs maillés.

Un des problèmes avec cette approche est que la séparation imposteur/géométrie réelle est visible lorsqu'on se rapproche du modèle [HQU00].

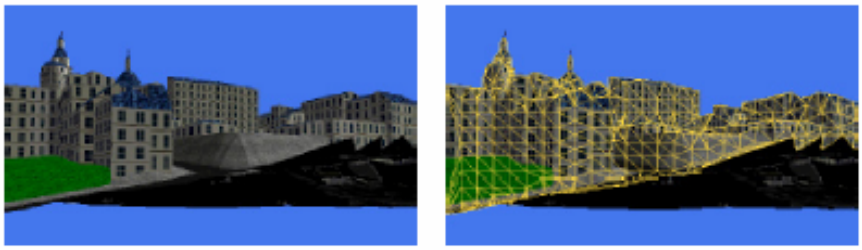


Figure 1.5 : Imposteurs maillées de *Sillion* : (a) texture de l'imposteur et (b) l'imposteur maillé.

2.3. Panoramas Cylindriques.

Les panoramas cylindriques (ou les cartes d'environnement cylindriques) sont utilisés pour fournir une indépendance d'orientation horizontale en explorant un environnement à partir d'un seul point. Des panoramas cylindriques peuvent être créés en utilisant des caméras panoramiques spécialisées, en rassemblant un ensemble de photographies acquises avec une caméra régulière ou en utilisant des interprétations d'ordinateur [OLI00].

Le choix d'une carte cylindrique est basé sur les facteurs suivants [CHE95] :

- Il est plus facile de capturer un panorama cylindrique que d'autres types de cartes d'environnement en utilisant des caméras.
- La carte cylindrique se courbe dans une seule direction, qui rend efficace l'exécution de la déformation de l'image.

QuickTime VR[1], représente une scène sous-jacente par un ensemble d'images cylindriques. Une nouvelle vue est synthétisée en déformant l'image cylindrique dont le centre de projection est le plus proche à celui de la nouvelle vue [OLI00, GENQ, LEO97]. Le rendu est accompli aux taux fortement interactifs (plus de 20 images par seconde) [LEO97].

Le système *QuickTimeVR* est capable de décrire des variations d'image dues aux changements de l'orientation d'observation. Les transitions de la position d'observation peuvent être rapprochées seulement en choisissant l'image cylindrique dont le centre de projection est le plus proche à la position d'observation courante [LEO97].

Le processus de rendu, basé sur image, de l'approche *QuickTimeVR* est un cas spécial de l'équation de déformation "cylindre-plan" dans le cas où tous les points de l'image sont calculés comme s'ils étaient à une distance infinie de l'observateur [LEO97, HQU00].

Figure 1.6 : Une vue perspective créée par déformation d'une région entourée par la boîte jaune dans l'image panoramique.

[1] Un système commercialisé développé par *Apple Computer Incorporate*.

Après *QuickTimeVR*, plusieurs systèmes de rendu de cartes d'environnement cylindriques ont été implantés [OLI00]. Ces systèmes supportent la rotation continue dans la direction horizontale et le changement d'échelle. La rotation dans la direction verticale est limitée par le champ de vision vertical des panoramas cylindriques (d'habitude 50 degrés). Pendant la rotation, la partie visible du panorama est déformée pour produire des images correctes plates et perspectives. La figure 1.7 représente une description d'un panorama cylindrique avec une vue *frustum* associée et illustre la géométrie de base de l'algorithme de déformation. Puisque le panorama cylindrique ne se courbe pas dans la direction verticale, la fonction de déformation se réduit à une opération de changement d'échelle appliquée aux scanlines verticaux (des colonnes) intersectées par la vue *frustum*. Le facteur de changement d'échelle varie d'une colonne à une autre. Pour un champ de vision donné, les facteurs de changement d'échelle peuvent être pré-calculés, stockés dans un tableau et réutilisés pour l'accélération du rendu.

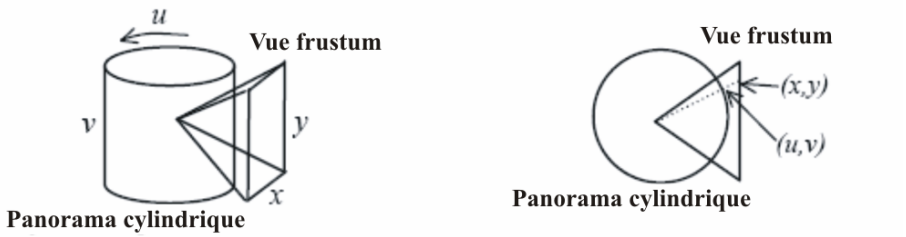


Figure 1.7 : Gauche : panorama cylindrique et vue *frustum*. Droite : géométrie de l'algorithme de déformation (vue supérieure).

La déformation peut être implantée en utilisant une fonction inverse semblable à celle utilisée pour le placage de texture. Ainsi, étant donné les coordonnées *(x, y)* d'un point dans le plan image, on peut directement obtenir les paramètres *(u, v)* correspondant dans la carte d'environnement cylindrique (Figure 1.7 (à droite)). Cela garantit le ré-échantillonnage approprié pour tous les pixels des nouvelles vues. La figure 1.8 représente un panorama cylindrique tandis que la figure 1.9 représente une re-projection plate obtenue en déformant une partie de la figure 1.8. L'anti-aliassage est implanté par le calcul de la moyenne des pixels voisins dans le panorama [OLI00].

Figure 1.8 : Image en Panorama Cylindrique.

Figure 1.9 : Re-projection plate obtenue du panorama montré dans la figure 1.8.

Pendant la rotation verticale, les images subissent deux ré-échantillonnages, qui ont comme tendance une dégradation de la qualité de l'image finale.

Le changement d'échelle est obtenu en changeant le champ de vision du *frustum* pour fournir plus de détail pendant l'agrandissement. Pour éviter l'aliassage pendant le changement d'échelle, une structure pyramidale peut être utilisée pour interpoler entre les niveaux de résolution appropriés, de la même façon que l'utilisation d'une pyramide de mip-map.

La limitation principale des systèmes basés sur des cartes d'environnement (cylindrique, sphérique ou cubique) est de fixer l'observateur à un seul emplacement (le point de vue d'où les images ont été acquises). Bien que l'observateur soit libre de tourner autour de ce point, on ne lui permet pas de se déplacer, puisque la translation produit des changements de visibilité. En conséquence, l'utilisation de cartes d'environnement n'est pas appropriée à la promenade dans des environnements virtuels [OLI00].

2.4. La fonction plénoptique.

Adelson et *Bergen* ont assigné le nom *fonction plénoptique*[1] à l'ensemble de rayons lumineux visibles de n'importe quel point dans l'espace, à tout moment et sur n'importe quelle gamme de longueurs d'ondes [LEO97, HQU00]. En outre, si on limite notre intérêt à la lumière quittant la coque convexe d'un objet englobé, on doit seulement représenter la valeur de la fonction plénoptique à travers quelques surfaces entourant l'objet. Un cube a été choisi pour sa simplicité informatique (Figure 1.10) [GOR96].

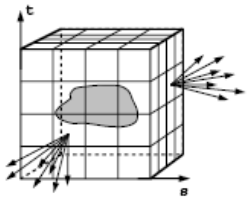


Figure 1.10 : La surface d'un cube tient toute l'information de radiance due à l'objet ci-joint.

Imaginant un œil idéalisé placé dans un point dans l'espace (V_x, V_y, V_z). De ce point, une direction d'observation peut être choisie en définissant un azimut et un angle d'élévation (θ, ϕ) aussi bien qu'une longueur d'ondes λ, dont les mesures seront prises. Dans le cas d'une scène dynamique, on peut choisir de plus le temps t, pour lequel on veut évaluer la fonction [LEO97, JAG03]. Cela aboutit à la forme suivante pour la fonction plénoptique :

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

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

$$\mu = P\ (V_x, V_y, V_z, \theta, \phi).$$

La figure 1.11 représente les éléments géométriques de ce rapport.

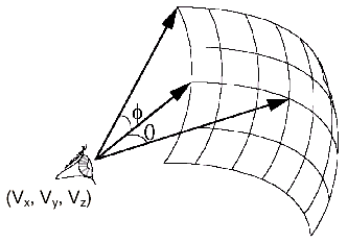


Figure 1.11 : Les Paramètres de la fonction plénoptique.

[1] Dérivé de la racine latine ***plenus***, signifiant la vision complète.

Malheureusement, la fonction plénoptique pour une scène donnée est rarement disponible. Cependant, n'importe quelle image projetée en perspective est un sous-ensemble de la fonction plénoptique d'un environnement. Son centre de projection définit un point d'observation et son champ de vision définit un angle solide sur lequel les valeurs de θ et ϕ sont définies. Les mesures photométriques représentées dans l'image indiquent les valeurs connues de la fonction plénoptique. Ainsi, n'importe quel ensemble d'images de référence d'un environnement représente un ensemble d'échantillons dispersés de la fonction plénoptique de cette scène [LEO97].

McMillan et *Bishop* ont affirmé que toutes les approches de rendu basées sur l'image peuvent être considérées comme des tentatives pour reconstruire la fonction : En ignorant le temps et la longueur d'onde, la dimension de la fonction plénoptique peut être réduite de 7 à 5. Si la scène peut être contenue dans une boîte englobante, le *Light-Field* et *Lumigraph* utilisent une fonction plénoptique 4D pour caractériser le flux de lumière dégagé dans l'espace dans une scène statique avec une illumination fixée. Si l'espace de vision est à l'intérieur d'un cercle 2D, les mosaïques concentriques peuvent être utilisées pour simuler la fonction plénoptique. Si le point de vue est fixé, la fonction plénoptique devient simplement une panorama 2D [HQU00].

2.4.1. Modélisation Plénoptique.

McMillan et *Bishop* [HQU00, POR04] ont choisit d'utiliser une projection cylindrique pour représenter un échantillonnage de la fonction plénoptique.

Leur méthode consiste en trois étapes clefs : échantillonnage, reconstruction et ré-échantillonnage. La projection cylindrique peut être acquise par une caméra vidéo et un trépied capable de faire une rotation continue. La reconstruction de la fonction plénoptique de ces échantillons exige l'estimation du flux optique des points quand le point de vue est translaté. Étant donné deux ou plusieurs images de référence panoramiques cylindriquement projetées, prises de différentes positions dans une scène statique, la projection cylindrique à partir d'un nouveau point de vue peut être obtenue par un placage de type cylindre $\mapsto$ cylindre. Alors, la projection cylindrique peut être re-projetée comme étant une image plate. La visibilité peut être résolue en utilisant un algorithme d'ordonnancement qui garantit un ordre approprié de front vers l'arrière [HQU00, POR04].

2.5. Light-Field et Lumigraph.

Le rendu *Light-Field* et *Lumigraph* utilise un paramétrage 4D de la fonction plénoptique si la scène est limitée par une boîte englobante pour créer des nouvelles vues de scènes/objets en ré-échantillonnant une base de données d'images représentant un échantillon discret de la fonction plénoptique. Dans cette représentation, un rayon est paramétré par ses intersections avec deux plans parallèles, une structure connue par *bloc de lumière* (Figure 1.12 (à gauche)) [OLI00, JAG03].

Le rendu *Light-Field* est introduit par *Levoy* et *Hanrahan*. Ils ont proposé de paramétrer des rayons par leurs intersections avec deux plans arbitraires. Par convention, le système de coordonnées sur le premier plan est (u, v) et sur le deuxième plan est (s, t). Des *Light-Field* (champs de lumière) peuvent être produits en assemblant une collection d'images. Pour des scènes synthétiques, un tableau 2D d'images peut être rendu en plaçant le centre de projection de la caméra virtuelle à une position (u, v) sur le premier plan. Dans leurs travaux, ils ont construit une caméra commandée par ordinateur pour l'acquisition du champ de lumière d'une scène réelle. La caméra est translatée à travers un réseau

régulier de positions sur le plan (u, v) et une image est acquise à chaque position. Chacune de ces images est ensuite projetée (par placage de texture) sur le second plan (s, t) [HQU00].

Gortel et al. ont développé le système *Lumigraph* en même temps que les *Light-Fields*. Dans leurs travaux, la fonction plénoptique est représentée sur la surface d'un cube englobant la scène. Ils ont utilisé la même méthode de paramétrage d'un rayon dans l'espace 3D du *Light-Field*. Les échantillons de *Lumigraph* peuvent être acquis en utilisant une caméra régulière prise à la main. Le calibrage de la caméra est l'estimation de la position de la caméra et un modèle géométrique grossier de l'objet peuvent être obtenus en utilisant quelques techniques de vision par ordinateur (Annexe B). Puisqu'une caméra arbitrairement positionnée est utilisée, les positions d'échantillonnage ne peuvent être ni spécifiées ni contrôlées ; ce qui ne garantit pas que ces échantillons de la fonction plénoptique soient régulièrement répartis. Ils ont proposé un nouvel algorithme en trois phases qui peut convertir ces échantillons vers un *Lumigraph* [HQU00].

Pour créer la base de données des images, un treillis régulier orthogonal est utilisé pour définir les positions de la caméra pour la capture. Un tel treillis est associé au plan *uv*, appelé *"plan de caméra"*. A chaque position du treillis, une image est prise en utilisant le plan *st* comme un plan image (Figure 1.13 - gauche). Puisque les positions des deux plans sont fixées, le champ de vision devient progressivement incliné quand la caméra se déplace du centre du plan de la caméra vers ses frontières (Figure 1.13 - à droite).

Conceptuellement, le rendu de nouvelles vues est illustré par la figure 1.12 (à droite). Pour chaque pixel de la nouvelle vue, on calcule les intersections de rayon de vision correspondant avec le plan caméra et le plan focal [OLI00].

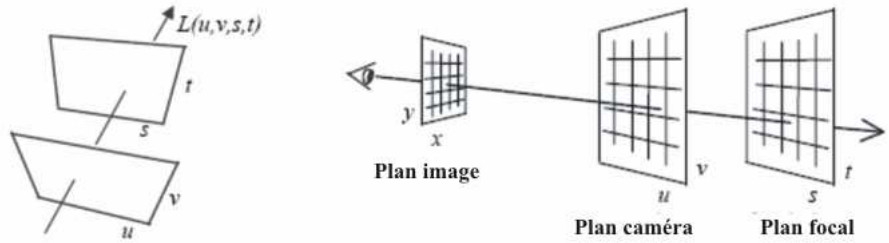


Figure 1.12 : Gauche : bloque de lumière. Un rayon lumineux est paramétré par ses intersections avec deux plans parallèles. Droite : représentation géométrique de la base de données d'image ré-échantillonnée.

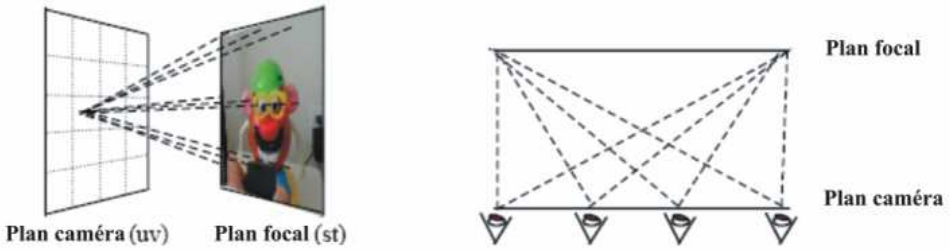


Figure 1.13 : Gauche : chaque élément de la grille sur le plan de la caméra correspond à une image. Droite : quand le point de vue se déplace du centre vers les frontières du plan de la caméra, la vue frustum devient progressivement inclinée.

Les coordonnées *(u, v)* de l'intersection sont utilisées pour choisir l'image(s) qui devant être utilisée pour le ré-échantillonnage; les coordonnées *(s, t)* de l'intersection sont utilisées pour choisir le pixel(s) réel de l'image(s) choisie. La stratégie de ré-échantillonnage peut s'étendre des plus proches voisins à l'interpolation quadra-linéaire. On note que les transformations de coordonnées du plan image

vers le plan caméra et le plan focal sont des transformations projectives plates. Elles peuvent donc être efficacement implantées en utilisant le matériel graphique [OLI00].

En voyant un objet de plusieurs points de vue, les rendus *Light-Field* et *Lumigraph* peuvent représenter des effets dépendants de la vue. Cependant, le grand nombre d'images exigées pour éviter le flou excessif (dû au pas d'interpolation utilisé pendant le ré-échantillonnage) augmente les exigences de stockage comparativement à d'autres techniques IBR. Le *Light-Field* et *Lumigraph* ne sont pas efficaces pour la représentation des environnements vus de l'intérieur vers l'extérieur.

Le rendu *Light-Field* est plus simple et plus robuste que les techniques de rendu précédentes basées sur l'image parce qu'il n'exige pas de valeurs de profondeurs, des informations de flux optique ou des correspondances de pixel. Par ailleurs, il n'est pas limité pour le rendu de scènes diffuses. Cependant, il présente aussi plusieurs limitations significatives [LIS98]:

(i) Des *Light-Field* de haute fidélité exigent une très grande capacité de stockage;

(ii) Le calcul du *Light-Field* d'une scène synthétique peut prendre une très longue période de temps; et

(iii) Le *Light-Field* 4D décrit le flux de lumière dans une région limitée d'espace sans objets et il n'est pas encore clair comment étendre des *Light-Fields* pour permettre la navigation libre à l'intérieur d'une scène.

2.6. Mosaïques Concentriques.

Shum et al. ont représenté la fonction plénoptique par des mosaïques concentriques, qui représentent une généralisation des panoramas cylindriques et peuvent être créées en contraignant le mouvement de la caméra aux cercles concentriques en composant des petits fragments d'images prises à partir d'emplacements différents le long de chaque cercle.

Dans ce cas, au lieu de l'utilisation d'une image cylindrique, des caméras de "fente"[1] se déplacent le long des cercles concentriques. Une série de mosaïques concentriques diversifiées est créée en composant les images acquises par chaque caméra le long de leurs chemins circulaires (Figure 1.14). Ainsi, un panorama cylindrique est équivalent à une mosaïque pour laquelle l'axe de rotation passe par le centre de la caméra de projection, comme le cas de la caméra C_0 dans la figure 1.14. Dans un ensemble de mosaïques concentriques, toutes les images associées à une colonne donnée sont acquises au même angle [OLI00, HQU00].

Les rayons de l'image d'entrée sont caractérisés par trois paramètres : rayon, angle de rotation et élévation verticale. Ainsi, c'est une fonction plénoptique 3D. Les nouvelles vues sont rendues en combinant les rayons appropriés capturés durant le rendu (Figure 1.15). Aucun effet de parallaxe vertical n'est capturé pour cette technique parce que les positions de la caméra sont contenues dans une région horizontale plate et une seule image est prise à chaque point de vue. Cependant, lors du rendu, l'absence d'information de profondeur cause des distorsions verticales dans les images obtenues. Plusieurs méthodes de correction de profondeur ont été proposées pour réduire la distorsion verticale [OLI00, HQU00, POR04].

[1] Une caméra capturant une seule ligne vertical d'une image. Il peut être simulé avec une caméra régulière en tenant seulement la colonne centrale des pixels de chaque image.

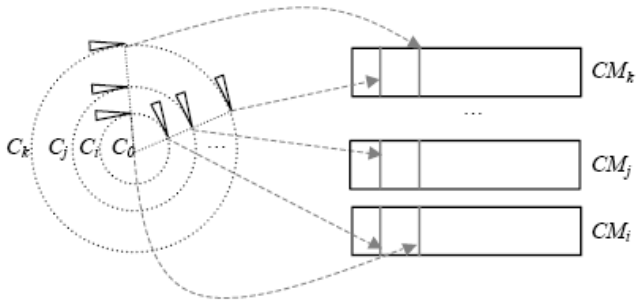


Figure 1.14 : Une mosaïque concentrique CM_l est créée en composant toutes les images de fente acquises par la caméra C_l le long de son chemin circulaire.

Comme la distance radiale entre deux cercles adjacents augmente, le nombre d'images disponibles pour la reconstruction d'une nouvelle vue donnée est réduit, en exigeant plus d'interpolation des données originales pour remplir le plan image de la nouvelle vue (Figure 1.15).

Comparativement avec le *Light-Field* ou *Lumigraph*, les mosaïques concentriques utilisent des fichiers de petite taille à cause de l'utilisation des images de fente, ce qui réduit significativement la quantité de données exigées pour approximer la fonction plénoptique.

Une autre limitation [OLI00] de l'approche de mosaïques concentriques est le manque de parallaxe verticale, résultant du fait que toutes les vues originales sont obtenues d'une région horizontale plate. Cependant, la parallaxe horizontale, semble être un indice assez fort pour une perception 3D, probablement en raison du fait que nos yeux ont tendance à rester relativement à un niveau constant quand on marche.

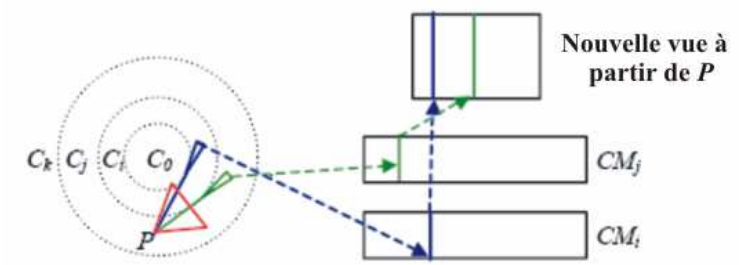


Figure 1.15 : Rendu avec mosaïques concentriques. Pour un point de vue donné P, la nouvelle vue est créée en composant des images capturées par des caméras C_l, $l=0..K$, aux angles pour lesquels une ligne passant par P est la tangente au cercle définit par C_l.

2.7. Image morphing.

L'*image morphing* est une technique intéressante qui peut produire des transitions lisses entre deux images. Les techniques de *morphing* d'image produisent des vues intermédiaires par une interpolation des couleurs et des formes. Les nouvelles vues sont donc géométriquement incorrectes (la figure 1.16) [JAG03, HQU00, POR04]. Il y a une variété de méthodes de *morphing* dans la littérature. Ces méthodes consistent d'habitude en deux étapes [HQU00, POR04]:

- D'abord, des correspondances et des éléments de contrôle sont établis entre les deux images.
- Ensuite, les positions et les couleurs de pixels dans deux images sont linéairement interpolées pour produire les images de transition. Les autres points de la nouvelle image sont interpolés de façon plus complexe (bilinéaire, *splines*) entre les éléments de contrôle les plus proches.

2.7.1. Interpolation de vue.

Chen et *Williams* ont présenté une méthode d'interpolation de vue pour synthétiser la nouvelle vue de quelques images de référence. Ils ont utilisé une carte pour stocker les vecteurs de compensation « *offset vectors* », représentant le placage en avant d'une image sur une autre image. Durant le rendu, n'importe quelle vue intermédiaire peut être produite par l'interpolation linéaire des vecteurs de compensation et ensuite le déplacement des pixels de l'image source par le vecteur interpolé vers leur destination. D'habitude, les nouveaux emplacements de pixel ne sont pas exactement corrects. Mais, tout à fait, cette interpolation donne une bonne approximation tant que le changement du point de vue est petit. Les trous sont remplis en interpolant les couleurs des pixels adjacents [HQU00].

2.7.2. Morphing de vue.

Seitz et *Dyer* ont introduit le *morphing* de vue, qui consiste en une pré-déformation de deux images avant le calcul d'un morphe et ensuite la post-déformation des images interpolées. Ils ont démontré que les techniques de morphing d'image utilisant l'interpolation linéaire pour calculer des positions de particularité dans des images intermédiaires peuvent causer des distorsions 3D sévères. La figure 1.16 représente une interpolation linéaire de deux vues perspectives d'une horloge (à droit et à gauche) qui cause un effet de courbure géométrique dans les images intermédiaires. La ligne discontinue présente le chemin linéaire d'une particularité pendant la transformation. Cet exemple indique les types des distorsions qui peuvent surgir dans les techniques de morphing d'image [HQU00].

Figure 1.16 : Un *morphe* déformant une forme.

3. Techniques hybrides de rendu basé sur image.

Les techniques hybrides d'IBR utilisent quelques informations géométriques, en plus des images, pour synthétiser les nouvelles vues de scènes. La quantité d'information géométrique utilisée varie de la profondeur de chaque pixel (utilisée dans des techniques basées sur la déformation 3D d'image) aux mailles polygonales raisonnablement détaillées (utilisées dans le placage *Light-Field*). Les techniques hybrides d'IBR sont les plus nombreuses. Dans cette section on va présenter quelques unes de ces techniques, en considérant la quantité d'information géométrique exigée.

3.1. Techniques basées sur des images de profondeur.

Les images de profondeur ou avec profondeur, sont utilisées par une grande classe de techniques IBR, incluant la déformation 3D d'image, la déformation post-rendu, des images de profondeur en couches (LDIs[1]), des objets basés sur image (IBOs[2]), les LDI-tree et le placage de texture de relief [OLI00].

[1] Layred Depth Image.
[2] Image Based Object.

Une image avec profondeur [OLI00] est une paire {*id, K*}, où *id* est une image digitale et *K* est un modèle de caméra associé *à id*. A chaque élément de l'espace de couleur d'*id* est associée une valeur scalaire représentant la distance par pixel (dans l'espace Euclidien) entre le point échantillonné et une entité de référence. *Si K* est un modèle de caméra de projection perspective, l'image est appelée *une image de projection perspective avec profondeur* et l'entité de référence est le centre de projection *K*.

Quand *K* représente un modèle de caméra de projection parallèle, l'image est appelée *une image de projection parallèle avec profondeur* et l'entité de référence est le plan d'image *K*. Dans une image avec profondeur, le contenu géométrique de la scène est représenté implicitement en combinant l'information de profondeur par-pixel avec le modèle de caméra associé à l'image.

La figure 1.17 (à gauche) représente un modèle d'une caméra de projection perspective. Les vecteurs $\vec{a}$ et $\vec{b}$ forment une base pour le plan image. $\dot{C}$ Est le centre de projection de la caméra *et* $\vec{c}$ est un vecteur du centre de projection à l'origine du plan image.

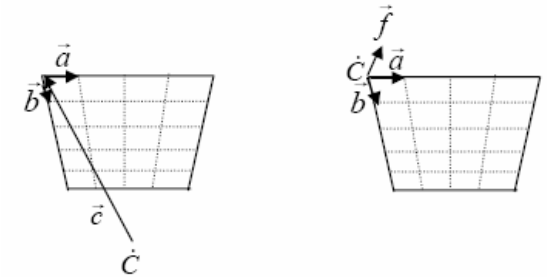


Figure 1.17 : Le modèle de caméra *pinhole* perspective (Gauche). Représentation de la caméra de projection parallèle (Droite).

La figure 1.17 (à droite) représente un modèle de caméra de projection parallèle. Les vecteurs $\vec{a}$ et $\vec{b}$ ont les mêmes définitions que dans le cas perspective. Le vecteur $\vec{f}$ est un vecteur unitaire orthogonal au plan engendré par $\vec{a}$ et $\vec{b}$. Les origines de tous ces vecteurs *sont à* $\dot{C}$, l'origine du plan d'image.

3.1.1. Déformation 3D d'image.

La déformation 3D d'image est une transformation géométrique qui plaque une image source avec profondeur i_s sur une vue cible arbitraire i_t. L'équation de déformation a été donnée par [OLI00] :

$$\vec{x}_t \doteq P_t^{-1} P_s \vec{x}_s + P_t^{-1}(\dot{C}_s - \dot{C}_t)\delta_s(u_s, v_s) \quad (1)$$

Où $\delta_s(u_s, v_s) = 1/t_s(u_s, v_s)$ est appelée la disparité généralisée[1] du pixel source (u_s, v_s) et $\doteq$ est l'équivalence projective.

Cette équation permet de produire les re-projections correctes de la scène échantillonnée par l'image source de points de vue arbitraires.

L'existence de secteurs noirs dans une image résultante est d'habitude mentionnée comme des *artefacts de disocclusion*, correspond aux surfaces non visibles dans l'image source, mais qui sont exposées dans les nouvelles vues. Pour réduire au minimum les artefacts représentés par des

[1] La profondeur projective d'un pixel, dérivé de la profondeur classique, correspondant au rapport de la distance focale de la caméra par la profondeur du point.

événements de désocclusion, on peut déformer des images multiples à la vue désirable ou utiliser des images de profondeur avec l'information de couleur et des valeurs de profondeur le long de chaque rayon d'échantillonnage (des images de profondeur en couches (LDIs)). De l'équation (1), on observe que le placage de texture sur un polygone est un cas spécial d'une déformation 3D d'image pour laquelle tous les pixels dans l'image source ont la même profondeur [OLI00].

Pendant la déformation d'une image, le problème de visibilité peut être résolu en utilisant un algorithme de liste de priorité. L'algorithme spécifie des ordres possibles pour déformer les pixels d'une image de projection perspective avec profondeur, donc la visibilité correcte est calculée pour des points de vue arbitraires sans une comparaison explicite de profondeur. L'algorithme 1 récapitule la procédure, qui est illustrée dans la figure 1.18 [OLI00].

```
Trouvez la projection du centre de projection cible dans le plan d'image source
(l'épipôle);
Divisez l'image source en quatre régions au plus en se basant sur les coordonnées de
l'épipole;
Si le centre de projection cible est derrière le centre de projection source alors
    Pour chaque région résultante
    Déformez ses échantillons de l'épipôle vers les frontières de la région;
Sinon
    Pour chaque région résultante
    Déformez ses échantillons des frontières de la région vers l'épipôle;
```

Algorithme 1 : Ordre compatible occlusion pour des images sources projetées en perspective.

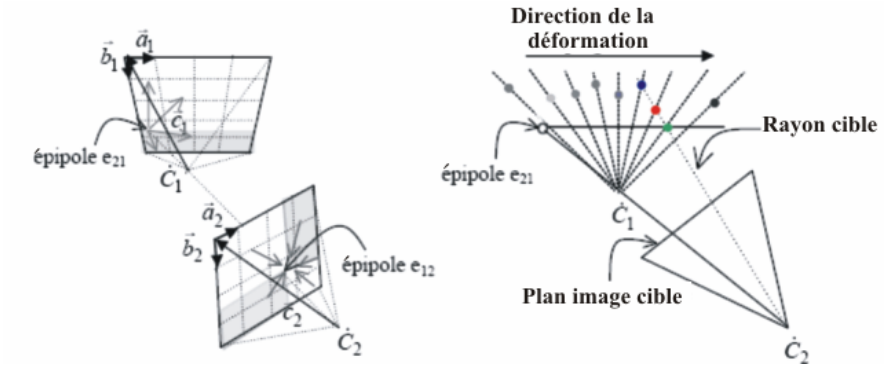


Figure 1.18 : Deux caméras *pinhole* projectives. Gauche : les épipoles divisent les images au plus en quatre régions. Les flèches grises spécifient l'ordre dans lequel les échantillons doivent être déformés. Droite : quand plusieurs échantillons se trouvent le long du même rayon, le plus proche au centre de projection cible est le dernier déformé.

L'intérêt derrière l'algorithme d'ordre compatible occlusion est illustrée dans la figure 1.18 (à droite) pour le cas dans lequel le centre de projection cible est derrière le centre de projection source : Chaque fois que plusieurs échantillons se situent le long d'un rayon cible, celui dont le pixel correspondant est le plus éloigné de l'épipole est le plus proche au centre de projection désiré et, donc, peut remplacer sans risque des échantillons précédemment déformés. Ainsi, des pixels sources doivent être déformés de l'épipole vers les frontières de l'image.

Pour éviter ou réduire au minimum l'occurrence des artefacts de désocclusion, on peut déformer plusieurs images de profondeur à la même vue cible. En général, l'algorithme d'ordre compatible occlusion ne peut pas être utilisé avec des images sources multiples. Dans ce cas, des alternatives pour

la résolution des issues de visibilité peuvent inclure la construction et l'utilisation d'une image avec des échantillons multiples par un rayon (LDIs), ou l'utilisation d'un buffeur de profondeur [OLI00].

3.1.2. Images de profondeur en couches.

L'image de profondeur en couches est une extension importante d'image traditionnelle. A cause de l'occlusion, une image à une seule couche ne peut pas contenir assez d'information pour un nouveau point de vue [HQU00].

Une image de profondeur avec couches (LDI), proposée par *Shade* et al. pour aborder les problèmes d'occlusion, est une vue de la scène d'un seul point de vue de caméra où chaque échantillon soutient deux informations (la couleur et le profondeur du pixel) pour chaque rayon d'échantillonnage (Figure 1.19) [OLI00, JAG03]. Dans ce cas, chaque élément de l'image consiste en une liste ordonnée d'échantillons. Dans ce cas, comme "un pixel" est prêt pour être déformé, tous les échantillons le long du rayon correspondant sont déformés en utilisant l'équation (1). L'échantillon le plus loin du nouveau centre de projection est déformé d'abord et le plus proche est le dernier déformé [OLI00, HQU00].

En échantillonnant des surfaces à plusieurs couches à partir d'un centre de projection, les LDIs peuvent réduire la présence des artefacts de désocclusion en conservant l'efficacité de la déformation 3D d'image (*3D Warping*). Bien que les LDIs peuvent réduire la présence des artefacts de désocclusion, ils ne peuvent pas les éliminent. Comme des images de profondeur à une seule couche, leur efficacité a tendance à être réduite quand il y-a un déplacement du point de vue désirable loin du centre de projection de la LDI. Pour réduire l'aliassage, un peu de filtrage est exigé. L'utilisation du mip-mapping permet de résoudre efficacement le problème d'aliassage. L'équivalent du mip-mapping dans le contexte des LDIs est connu sous le nom de LDI-tree (les arbres LDIs) [OLI00].

Chang et al. ont proposé l'arbre LDI pour résoudre le problème des images LDI causé par des taux d'échantillonnage différents dans les images originales[1]. L'arbre LDI est un *octree* avec une LDI attaché à chaque cellule. Chaque nœud est une boîte englobante où chaque face est utilisée comme un plan de projection (orthographique) de la LDI (la projection utilisée pour la LDI orthographique). La LDI attaché à une cellule *octree* contient seulement les échantillons d'objets qui se trouvent dans la boîte englobante. L'arbre LDI est construit à partir des images de référence en re-projetant chaque pixel de ces images à la LDI d'une cellule octree, et en filtrant chaque pixel affecté d'une LDI dans les LDI des cellules parentes.

Durant le rendu, l'*octree* est traversé de haut en bas. A chaque niveau, on vérifie si la LDI du nœud déformé vers l'image résultante peut fournir assez de détails. Si la cellule actuelle ne fournit pas assez de détail, ses enfants sont traversés [HQU00, POR04].

[1] D'abord, les pixels subissent deux ré-échantillonnage durant leur voyage de l'image d'entrée vers celle résultante. Cela peut dégrader la qualité d'image. Deuxièmement, quelques informations peuvent être perdues si une surface est échantillonnée dans une des images de façon meilleure que celle du point de vue de la LDI.

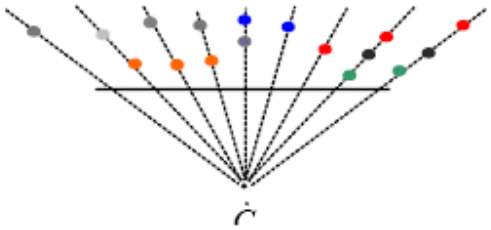


Figure 1.19 : Image de profondeur en couches. Chaque rayon d'échantillonnage peut contenir des échantillons multiples.

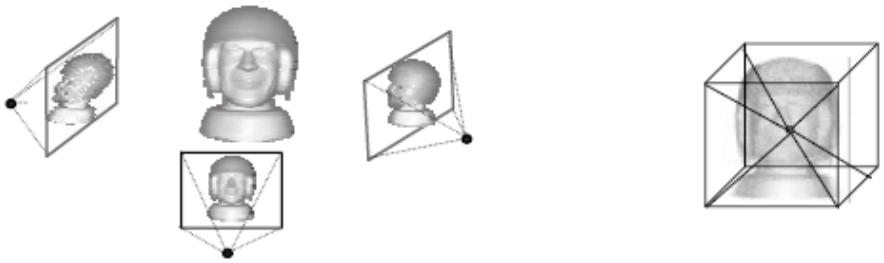


Figure 1.20 : Gauche : objet échantillonné de centres de projections multiples. Droite : les échantillons sont enregistrés et re-projetés sur des plans image perpendiculaires (les faces d'un cube) partageant un seul centre de projection.

Les images de profondeur avec couches ont été aussi utilisées par *Max* pour représenter d'une façon hiérarchique et rendre des modèles photo-réalistes complexes d'arbres botaniques. *Max* a utilisé une représentation semblable à un LDI, pour un anti-alissage de haute qualité. Il y a des ressemblances entre son approche et celui présenté par *Lischinski* : toutes les deux approches utilisent la projection LDIs parallèle et stockent un normal avec chaque échantillon de la scène pour calculer l'ombrage après la re-projection. Cependant, l'approche de *Lischinski* se concentre sur l'issue de traitement des phénomènes d'ombrage dépendant de vue, qui n'ont pas été traitées par *Max* [LIS98, SHA97].

3.2. Objets basés sur image (IBO).

Les objets basés sur images (IBOs) fournissent une représentation compacte basée sur image d'objets 3D qui peuvent être rendus dans un ordre compatible occlusion. Un objet basé sur image est construit en acquérant des vues multiples de l'objet, les enregistrant et les ré-échantillonnant à partir d'un seul centre de projection sur les faces d'un parallélépipède. Ainsi, chaque objet basé sur image est représenté par six LDIs. L'utilisation d'un parallélépipède permet à une telle représentation d'être décomposée en régions plates paramétrées (les faces du parallélépipède) pour lequel un processus de déformation peut être efficacement implanté [OLI00].

Oliveira et *Bishop* ont représenté des objets par six images de profondeur en couches partageant un seul centre de projection. Les objets basés sur image peuvent subir des opérations de changement d'échelle, des translations et des rotations libres, pour être utilisés comme primitives de construction des scènes plus complexes. Ils présentent aussi un nouvel algorithme de liste de priorité basé sur l'ordonnancement compatible occlusion de *McMillan* et *Bishop* pour le rendu de telles scènes [HQU00].

Après l'acquisition des vues multiples de l'objet, ces échantillons sont enregistrés et re-projetés sur des plans images perpendiculaires partageant le même centre de projection qui est mis à l'intérieur de l'objet [HQU00].

3.2.1. Objets Basés sur image par des imposteurs avec couches.

Schaufler a utilisé les imposteurs avec couches pour représenter des objets. Un imposteur avec couches est constitué de plusieurs polygones transparents. Sur chaque polygone tout les texels dessinés représentent les parties de la surface de l'objet qui sont à une distance semblable de l'observateur au polygone. Au lieu de rendre un polygone transparent, comme avec des imposteurs, plusieurs couches polygonales sont rendues comme étant une pile pyramidale ayant comme point de vue son apex. Dans chaque couche, les secteurs dessinés de l'image sont seulement celles ayant une valeur z correspond étroitement à la distance de la couche par rapport au point de vue. Puisque les imposteurs avec couches fournissent des valeurs de profondeur approximatives, il est possible de résoudre correctement le problème de la visibilité [HQU00].

3.2.2. Rendu par liste de priorité.

Considérons une image de référence sphérique et un point de vue. Le rayon de vision passant par le centre de l'image sphérique intersècte sa surface en deux points. Le point le plus proche à l'observateur (ou derrière lui, si l'observateur est à l'intérieur de la sphère) est appelé *épipôle positif* (e+); l'autre point est appelé *épipôle négatif* (e-). Un ordre compatible occlusion pour des images de référence sphériques consiste à déformer des échantillons de l'épipôle négatif vers le positif. Puisque la représentation IBO est topologiquement équivalente à une sphère, un IBO peut être déformé dans un ordre compatible occlusion. Dans le cas d'IBOs, les propriétés suivantes peuvent être observées :

- Une fois un ordre relatif entre les LDIs a été établi, chaque image peut être indépendamment déformée en utilisant l'algorithme de déformation 3D des LDIs;
- Les épipôles positif et négatif se situent sur des faces opposées;
- Il n'y a aucune redondance parmi les images, *c'est-à-dire*, on ne voit aucun échantillon dans plus d'une face;
- Le champ de vision entier est couvert.

La ligne connectant le centre de projection voulu et le centre de projection d'IBO intersecte le cube à des faces opposées (Figure 1.21) et définit un ordre compatible occlusion pour déformer l'objet entier. La face contenant l'épipole négatif (e-) doit être déformée d'abord, tandis que la face contenant l'épipole positif (e+) doit être la dernière déformée. Dans la figure 1.21, les faces contenant les épipoles positif et négatif, sont appelés respectivement F et K. Les deux autres paires de faces opposées sont appelées (A, A') et (B, B'). Ainsi, la déformation des faces du cube dans l'ordre (K, B, A, A', B', F), ou (K, B, A', A, B', F) produit une visibilité correcte à partir de la position de vue désirable [OLI00].

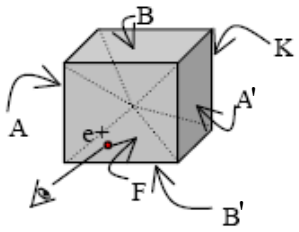


Figure 1.21 : Les faces du parallélépipède sont étiquetées selon la position du point de vue désirable. F contient l'épipole positif (e+), tandis que K contient l'épipole négatif.

L'ordre de déformation relatif pour les différentes lignes épipolaires n'est pas important, mais les points sur une seule ligne épipolaire doivent être déformés selon un ordre particulier (Figure 1.22 (a)). Ainsi, l'image d'entrée est fendue en régions, dont chacune est traitée selon un ordre de parcours différent (Figure 1.22 (b)).

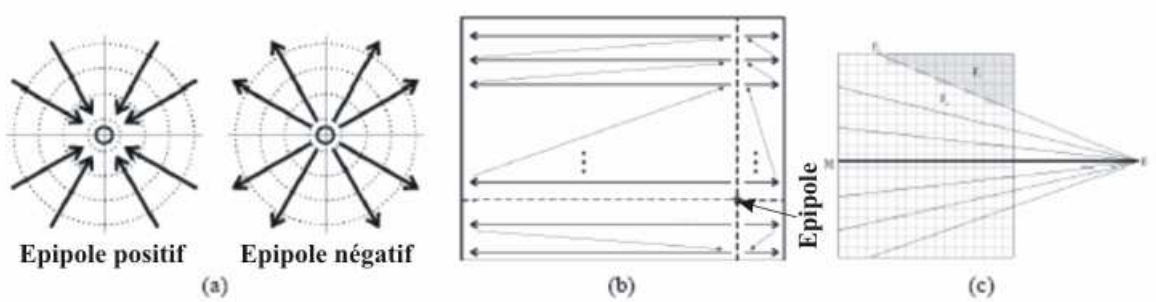


Figure 1.22 : (a) L'ordonnancement compatible occlusion de front vers l'arrière se déplace vers l'épipole positif et s'éloigne de l'épipole négatif. (B) l'image de référence peut être divisé en quatre régions d'occlusions compatibles. (C) les régions sont divisées le long des lignes épipolaires en *p* fragments de surface égales, où *p* est le nombre de processeurs disponibles.

3.3. Texture de Relief.

Le placage de texture de relief est une extension du placage de texture classique qui supporte la représentation des détails de surfaces 3D et la parallaxe de changement de point de vue [OLI00].

Oliveira et *Bishop* ont proposé la texture de relief (une texture étendue par un déplacement orthogonal par texel) pour supporter la représentation de détails superficiels 3D et la parallaxe de mouvement de point de vue. L'approche résulte d'une factorisation exacte de l'équation de déformation 3D de l'image vers une pré-déformation suivie par un placage de texture classique. La texture de relief peut être plaquée sur un polygone en utilisant un processus à deux étapes [HQU00]:

- D'abord, elle est convertie en une texture ordinaire en utilisant une transformation 1D en avant.
- La texture résultante est alors plaquée sur le polygone en utilisant le placage de texture standard.

La figure 1.23 représente le pipeline de placage de texture de relief. Les textures de relief sont des images de projection parallèles avec profondeur, ce qui simplifie la pré-déformation et le rendu des objets 3D complets [OLI00]. La pré-déformation dépend de la position relative de l'observateur en ce qui concerne le polygone qui doit être la texture de relief plaqué et garantit que le rendu final est équivalent à la vue des surfaces 3D réelles. La figure 1.24 illustre les étapes du pipeline d'habillage avec une texture de relief pour le cas d'un polygone simple.

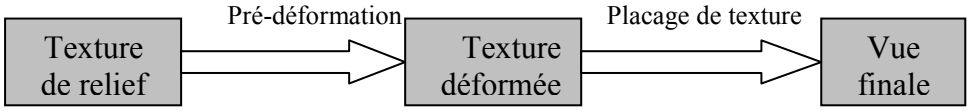


Figure 1.23 : Le pipeline d'habillage avec une texture de relief.

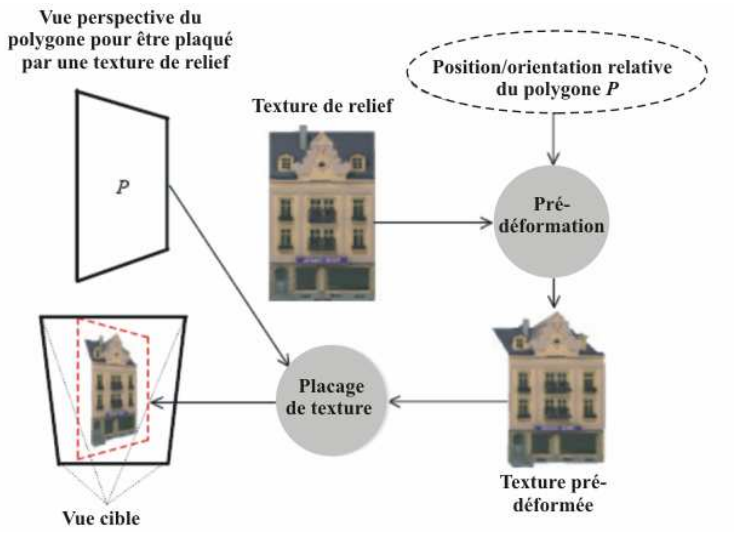


Figure 1.24 : Placage d'une texture de relief.

L'étape de pré-déformation a comme entrée une texture de relief et les paramètres (la position relative et l'orientation) du polygone qui doit supporter la texture de relief. Par conséquent, une texture pré-déformée est générée, ensuite elle va être plaquée sur le polygone produisant une vue correcte de la scène [OLI00].

La pré-déformation.

On a besoin de trouver une pré-déformation p, pour que la composition m o p, où m est une transformation de placage de texture standard, soit cohérente avec les projections des surfaces 3D. Ainsi, les équations de pré-déformation peuvent être obtenues en factorisant les composantes de la fonction de placage de texture à partir de l'équation de déformation 3D. On obtient alors des nouvelles coordonnées (u_i, v_i) après la pré-déformation d'un pixel source (u_s, v_s). Particulièrement, les coordonnées d'un texel dans l'image pré-déformée peuvent être calculées indépendamment l'un de l'autre, *c'est-à-dire*, u_i ne dépend pas de v_s et v_i ne dépend pas de u_s [OLI00].

L'indépendance entre les coordonnées (u_i, v_i) permet à la pré-déformation d'être implantée comme étant un processus à deux étapes en utilisant seulement une opération le long des lignes et des colonnes de la texture et en utilisant seulement deux texels à tout moment (Figure 1.25) [OLI00] :

- La texture de relief originale est transformée vers une représentation intermédiaire en transférant chaque texel vers sa colonne finale le long de sa rangé originale (à droite en haut).
- Dans la deuxième étape, un texels est déplacé à leur rangé finale (à droite au fond). La pré-déformation entière est implantée dans un ordre compatible occlusion en utilisant une adaptation de l'algorithme original de l'ordre compatible occlusion pour des images de projection parallèles avec profondeur. L'implantation de la pré-déformation comme une série d'opérations 1D sur les paires de texels simplifie la tâche de reconstruction. Elle doit aussi mener à une implantation matérielle simple et efficace.

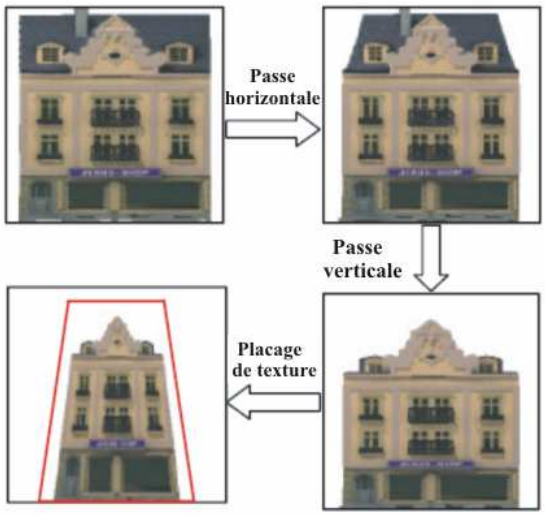


Figure 1.25 : La pré-déformation est implantée comme une séquence de déformations 1D horizontales et verticales.

Figure 1.26 : Un objet représenté par six textures de relief associées aux faces de sa boîte englobant (à gauche). Rendu de la statue comme étant deux polygones habillés par une texture de relief affichée avec frontières (au centre) et sans frontières (à droite).

Les textures de relief peuvent aussi être utilisées pour modéliser et rendre des objets tridimensionnels. Une représentation d'objet consiste en six textures de relief associées aux faces de la boîte englobant l'objet. La figure 1.26 (à gauche) est une représentation de texture de relief d'une statue modélisée avec 35,280 polygones. Les nouvelles vues d'un objet peuvent être obtenues en pré-déformant les textures de relief et en plaquant les images résultantes sur les faces de la boîte. La figure 1.26 (au centre) et (à droite) représente le rendu du modèle avec une texture de relief pour un point de vue donné avec et sans mettre en évidence les frontières de polygone habillé avec cette texture [OLI00].

Une texture de relief est une représentation à couche unique, elle est donc non appropriée au rendu de structures multicouches. Dans ces situations, une LDI est probablement un meilleur choix pour éviter l'occurrence des artefacts de disocclusion.

3.4. Textures volumiques interactives.

Meyer [MEY98] a introduit une représentation simple des textures volumiques en couches. Dans ce modèle, le volume de référence contient l'ensemble des images 2D qui représentent les couches. Ce volume se plaque sur une surface composée uniquement de quadrilatères.

Il a généralisé ce modèle pour que le volume contenant l'échantillon de texture se plaque sur la surface de manière complètement indépendante du maillage de celle-ci. En suite, il a amélioré progressivement ce modèle de façon à le rendre plus général, plus souple et plus rapide. En utilisant notamment trois directions de couches pour limiter au maximum l'effet visuel que donne la

superposition de couches. Ainsi que l'introduction d'un critère de qualité qui indique le nombre minimum de couches qu'il faut tracer pour une erreur donnée.

En ce qui concerne la génération du texel de référence, il a utilisé essentiellement deux techniques [MEY98] :

- La première, consiste à transformer des objets polygonaux (au format Inventor) en texel en utilisant simplement le rendu d'*OpenGL* pour chaque couche. En plaçant les deux plans de *clipping* (proche et lointain) très près l'un de l'autre on obtient une vue en coupe de l'objet qui correspond dans cette représentation à une couche du texel.
- La deuxième technique de création utilise les textures de *Perlin* : une image 2D de *Perlin* est utilisée comme champ de hauteur (i.e. que le niveau de gris de l'image donne la hauteur de la colonne correspondante).

Pour ces deux techniques se pose le problème de transformer des tranches d'objets surfaciques en solides pleins. Pour remédier ce problème, *Meyer* a utilisé un algorithme de remplissage des parties internes de l'objet de manière à ne pas voir des "trous" dans l'objet.

Pour l'animation de scènes à base de texels, il se limite au cas où la couche de texels suit passivement les déformations de la surface sous-jacente. Pour l'animation de cette surface, il a implémenté des techniques déjà connues : la déformation suit soit une spécification cinématique (par exemple une fonction sinus) soit une spécification dynamique (un modèle physique d'un drapeau au vent par exemple).

Meyer a utilisé une nouvelle représentation du volume de référence, au lieu de celle utilisée par *Neyre*t. Ce nouveau volume de référence peut être vus comme un ensemble de N images 2D de taille $X \times Y$ au format *RGBA* (les trois composantes de couleur *Red*, *Green*, *Blue* et la transparence *Alpha*). On peut aussi le voir comme un volume de $N \times X \times Y$ voxels ayant chacun quatre composantes (les trois de couleurs et la transparence). Ce volume de référence ne contient plus d'informations sur la réflectance puisque la couleur pré-calculée est stockée à la place. A noter que le coefficient *alpha* pour la transparence est indispensable sinon la face du dessus cacherait une partie des faces d'en dessous [MEY98].

4. Les applications de RBI.

4.1. Le RBI dans des systèmes de promenade.

Le rendu basé sur image a été utilisé dans des systèmes de promenade. Certains d'entre eux sont basés sur l'algorithme de déformation 3D de *McMillan* et *Bishop*. *Rafferty* et al. ont proposé d'utiliser la déformation 3D d'image pour l'accélération des promenades dans des systèmes architecturaux. Quelques images de référence ou des images de profondeur en couches sont stockées à chaque entrée d'une étape de pré-traitement. Durant le rendu, les entrées sont remplacées de ces images déformées au point de vue actuel [HQU00].

Mark et al. ont développé un système de déformation 3D post-rendant. Les images de référence sont rendues d'une façon conventionnelle à quelques points de vue clefs. Les images tirées sont produites en déformant les images de référence. L'algorithme déforme toujours deux images de référence différentes et compose les résultats pour éviter des artefacts liés à l'occlusion (figure 1.27). Un point de vue d'une image de référence est placé près de la position précédente de l'observateur et le

point de vue de l'autre image est placé près d'une position future de l'observateur. Ils ont utilisé un maillage triangulaire pour la reconstruction. Les méthodes de reconstruction et composition sont basées sur le calcul de connexion qui détermine si trois sommets d'un triangle de maille d'une image de référence sont placés sur une surface cohérente. La connexion est calculée en évaluant si vraiment les normales de sommets du triangle sont compatibles avec les valeurs Z des sommets du triangle. L'algorithme peut augmenter le taux d'affichage d'un système de 5Hz à 60 Hz [HQU00].

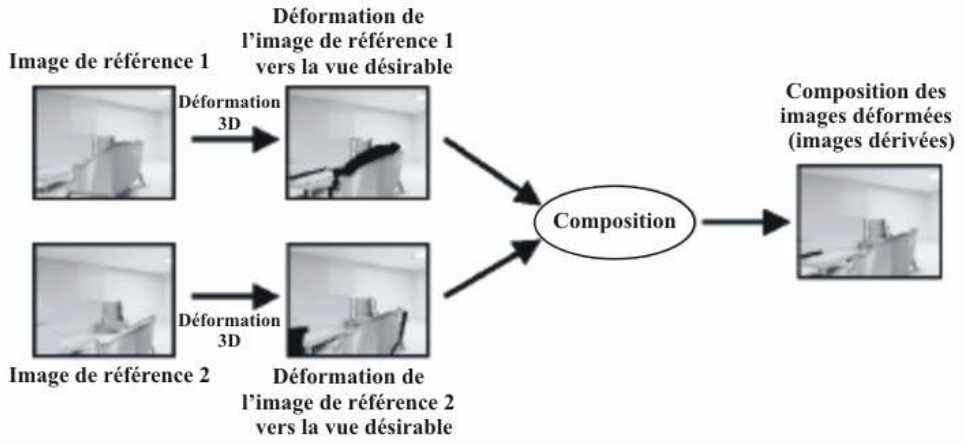


Figure 1.27 : Une simple image déformée manquera d'information sur des secteurs occlus dans son image de référence. Des images de référence multiples peuvent être composées pour produire des images dérivées plus complet.

4.2. L'accélérant de rendu de volume avec RBI

Mueller et al. ont utilisé le RBI pour accélérer le rendu de volume de la manière suivante :

- Ils décomposent d'abord le volume en blocs et chaque bloc est rendu dans une image de quelques points de vue clefs.
- Pour chaque bloc image, une maille intermédiaire des données de volume est construite et le bloc image respective peut être utilisé comme une texture pour la maille.
- Ces polygones habillés de texture peuvent alors être transformés selon les nouveaux paramètres de vue et l'image finale au nouveau point de vue peut être rendue par leur composition.

Chen a proposé une structure de rendu de volume basé sur image pour rendre la surface contenue dans les données volumétriques. Cette approche profite de la cohérence entre les images voisines pendant la navigation (La figure 1.28 expose sa méthode). Au lieu de lancer un rayon pour chaque pixel de l'image du nouveau point de vue, la méthode rend seulement les pixels qui sont absolument nécessaires. Le reste de l'image est produit à partir d'une vue de rendu de volume précédemment construite (*keyframe*) par un placage de texture. En se basant sur la fonction de transfert d'entrée, un modèle géométrique d'iso-surface est extrait à partir du volume. Ce modèle géométrique est habillé avec la texture de la vue originale et projeté ensuite à la nouvelle vue. La partie rendue de l'image représente la région visible, dans la vue originale et la nouvelle vue ; le secteur rendu par un lancer de rayon représente la région cachée dans la vue originale, mais qui devient visible dans la nouvelle vue. La dernière partie occupe seulement un petit pourcentage de la nouvelle vue quand la caméra se déplace légèrement de la position de vision de la vue originale; par conséquent, une amélioration substantielle de rendu peut être obtenue [HQU00].

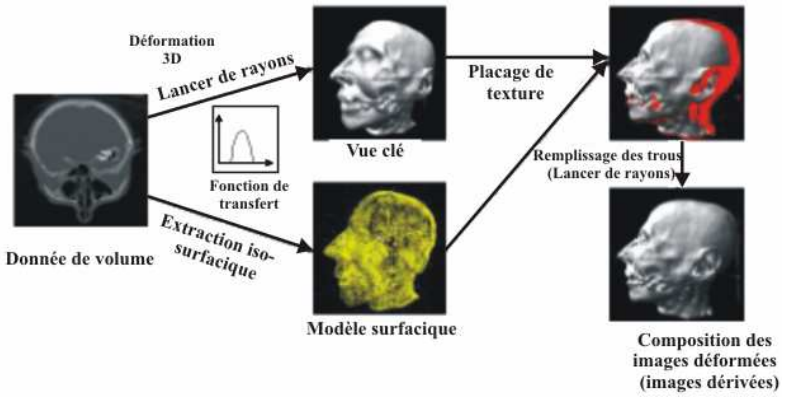


Figure 1.28 : La structure de rendu d'un volume basé sur image.

5. Bilan.

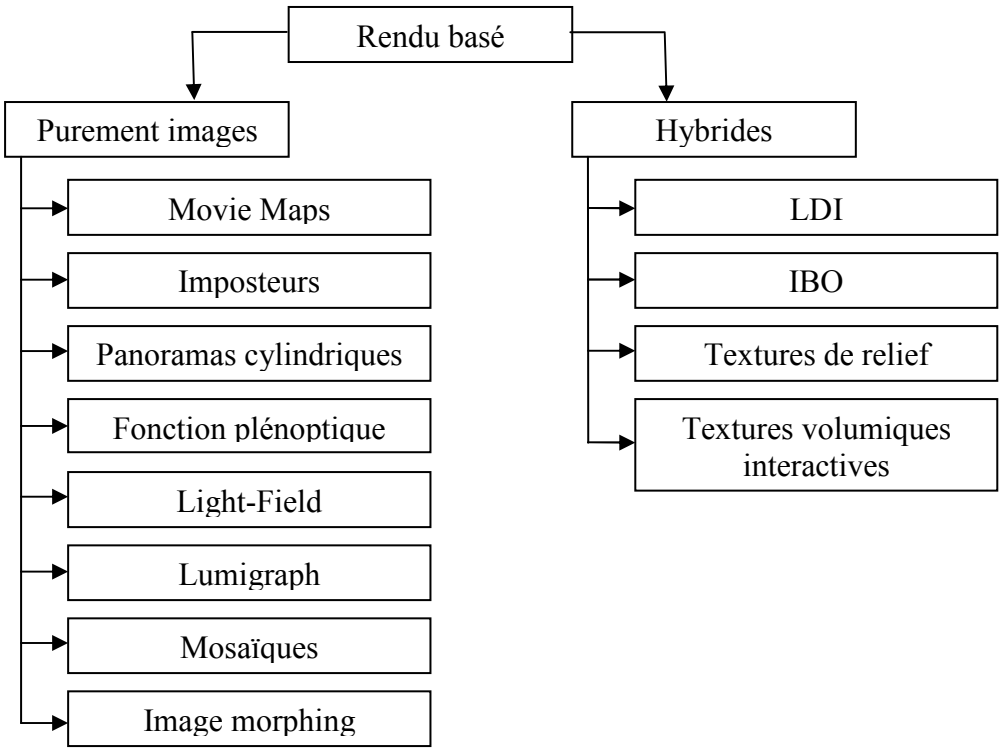


Figure 1.29 : Les techniques de rendu basé sur images.

Nous avons présenté ci-dessus une vue générale sur des techniques de rendu basées sur des images classées en deux grandes catégories. Le Movie-Map, est l'une des premières techniques purement basées sur des images présente dans ce domaine. Cette technique permet d'effectuer une promenade dans une ville en se basant sur une base de données d'images, ce qui nécessite une grande capacité de stockage. Les imposteurs représentent une famille de techniques permettant une accélération de rendu, car un imposteur représente une entité plus rapide à afficher qu'un objet réel, tout en conservant les caractéristiques visuelles des objets et en maintenant un taux d'affichages élevé et constant.

Le problème de visibilité, imposé par ces techniques, est résolu par l'utilisation des *Naillboards* et les imposteurs en couches. Par la suite, l'apparition des imposteurs maillés permet de résoudre le problème de parallaxe en utilisant des mailles triangulaires. Mais le problème des trous entre le modèle local et l'imposteur reste présent.

La technique des panoramas cylindriques permet d'explorer un environnement à partir d'un seul point, mais cette méthode n'est pas appropriée à la promenade dans des environnements virtuels parce qu'elle fixe l'observateur à un seul point.

Les de techniques de rendu *Light-Field* et *Lumigraph*, simples et robustes, peuvent représenter des effets dépendants de vue. L'inconvénient majeur de ces deux techniques est qu'elles exigent une grande capacité de stockage, et qu'elles ne permettent pas la navigation à l'intérieur d'une scène. Ce problème est résolu par la technique des mosaïques concentriques en utilisant "des images de fente". Mais les problèmes de manque d'un parallaxe verticale et l'apparition des distorsions verticales à cause de l'absence d'information de profondeur restent présents.

Les images de profondeur, une des techniques hybrides, permettent de générer une nouvelle vue de la scène selon un nouvel point de vue. Le problème d'occurrence des trous (artefacts de désocclusion) dans les images résultantes est remédié par l'utilisation des couches des images de profondeur. Mais un problème de perte d'information est présent à cause des taux d'échantillonnage différents (plus grand que celui de l'image de référence). De plus, comme les pixels subissent deux ré-échantillonnages, une dégradation apparaît dans la qualité de l'image finale.

6. Conclusion.

Nous avons présenté quelques techniques de rendu basé sur image qui sont principalement utilisées pour accélérer le rendu ou aider d'autres méthodes de rendu comme le rendu de surface ou le rendu de volume (Dans le cas où la scène est constituée d'objets géométriques ou d'objets de volume). Les méthodes de rendu basé sur image sont basées sur la fonction de plénoptique, comme le *Light-Field*, *Lumigraph*, la modélisation plénoptique … etc. Elles utilisent directement des images comme primitives de modélisation et de rendu, ce qui facilite la modélisation et accélère le rendu.

Comparé aux techniques de modélisation et de rendu 3D traditionnel, ces approches peuvent éviter l'étape consistant à créer explicitement un modèle 3D de la scène, et peuvent modéliser des objets qu'il est difficile de les modéliser par des surfaces. Le temps de rendu de la scène est d'habitude indépendant de la complexité de la scène.

Texture volumique à base de couches d'images : La modélisation

1. Introduction.

La méthode des textures volumiques permet de représenter plusieurs types d'objets complexes naturels, qui n'ont pas une surface bien définie (pré, fourrure, certains organes humain internes, … etc.), par une zone volumique au voisinage d'une surface. Mais cette méthode fonctionne uniquement en lancer de rayons, ce qui donne au mieux des temps de calcul de l'ordre de dix minutes.

Notre objectif est d'utiliser les fonctionnalités des cartes graphiques pour mettre au point un modèle inspiré des textures volumiques, à base de couches de textures transparentes superposées et décalées à la surface de l'objet à habiller. Un motif de texture représentant, dans son volume, un objet de plusieurs milliers de faces ne coûtera ainsi que le prix de quelques faces, à savoir les tranches du volume. Une scène composée de multitudes de texels déformés offrira donc une complexité géométrique apparente énorme, tout en ne requérant en fait que le rendu de quelques milliers de faces.

Notre travail s'est essentiellement concentré sur 2 aspects :

- Méthodes de spécification du motif 3D de texture et de sa conversion en volume.
- Choix d'une représentation et mise au point de l'algorithme de rendu qui en découle.

2. Motivations pour une nouvelle représentation.

Le monde réel est très riche de scènes et d'objets complexes répétitifs (La ferrure, une prairie, une chevelure, …etc.) qui est difficile à les modéliser et à les rendre. La généralisation du modèle de texture volumique de *Kajiya* et *Kay*[1] par *F. Neyret* a généré de l'intérêt. *Neyret* [NEY96] a introduit une nouvelle représentation de la fonction de réflectance en utilisant des ellipsoïdes, une primitive paramétrable qui est capable de modéliser de nombreux types de formes, comme un support des normales à la surface contenue dans le volume de référence. D'autre part la méthode originale est très lente : le rendu de Kajiya et Kay peut parfois considérer inutilement beaucoup de voxels (par exemple dans le cas où le volume est loin de l'observateur). *F. Neyret* a donc introduit une approche multi-échelle similaire utilisant les octrees qui comporte beaucoup d'avantages. Le niveau de détails affichés est ainsi modulé en fonction de la distance qui sépare le texel de l'observateur, ce qui procure un gain de vitesse appréciable, en gardant la même qualité d'image (i.e. avec très peu d'aliassage) (Figure 2.1).

[1] Dédié à la construction de ferrure pour l'utiliser comme une peluche d'un ours.

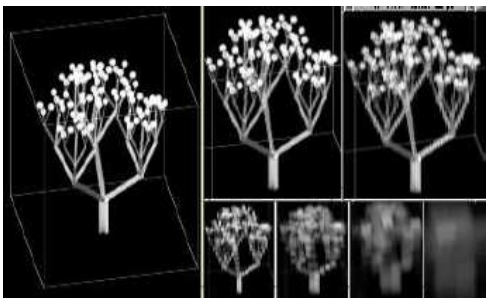


Figure 2.1 : Exemple d'un texel à différents niveaux de détails.

Avec ce modèle enrichi de texture volumique, *F. Neyret* arrive à rendre des images de bonne qualité dans un temps tout à fait raisonnable en lancer de rayons (5 à 20 minutes). Les textures volumiques peuvent être alors vues de deux façons :

- Une représentation des scènes complexes facile à contrôler par l'utilisateur en introduisant la multiéchelle.
- Une représentation permettant un rendu très efficace en temps et en qualité des scènes.

En effet, cette technique, malgré leur puissance de rendu de très bonne qualité visuelle, comporte les inconvénients suivants :

- Le temps de rendu d'une image, apparaît raisonnable, reste toujours lent. Réciproquement, les images en réalité virtuelle sont très épurées, et tout gain en complexité de l'apparence est un grand progrès, même sans atteindre le réalisme permis par le lancer de rayons (pour un simulateur médical, par exemple, le réalisme est important, et en même temps le temps réel est essentiel).
- L'espace nécessaire au stockage du volume de référence, représenté par un arbre octal, est très grand, ce qui nécessite une mémoire de travail de très grande capacité. L'évolution de la taille de stockage est une fonction exponentielle (Dans le cas générale : la taille d'un volume de référence = taille d'un voxel $\times 2^{3\times\text{niveau de résolution}}$).

Notre travaille se base sur la technique de rendu d'une image de données volumiques présentée par *Meyer* et inspirait des travaux de *Lacroute* et *Levoy* sur l'accélération d'un rendu volumique, dont notre objectif est de rendre des textures volumiques par une superposition de couches.

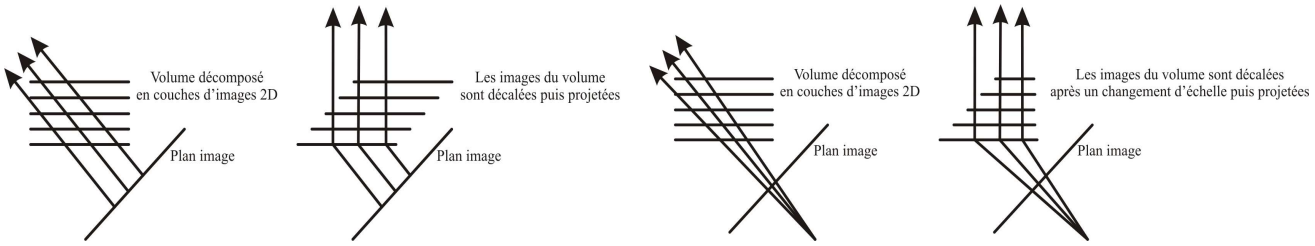
Le rendu volumique se calculait usuellement en lançant des rayons à travers l'espace voxelisé, ce qui se traduisait par des temps de calcul très long. *Lacroute* et *Levoy* [LAC94] ont proposé un algorithme intéressant pour rendre ces images de données volumiques en temps réel[1]. Ils traitent leurs données volumiques comme des couches. Pour le rendu, ils proposent de projeter successivement les couches sur le plan image et ainsi obtenir l'image finale. Chaque couche projetée doit être combinée avec le résultat précédent en tenant compte de la densité ; on peut donc interpréter une couche comme une texture transparente. La projection d'une couche s'effectue plus vite que de faire les calculs de

[1] A partir d'une vingtaine d'images par seconde dans certains contextes comme la réalité virtuelle, où il est nécessaire de calculer et d'afficher une suite d'images à une cadence suffisamment élevée.

projection pour chaque voxel individuellement. Cette factorisation permet donc de gagner beaucoup de temps.

La figure 2.2.a [LAC94] illustre la transformation de l'espace objet à l'espace objet décalé pour une projection parallèle. On suppose que le volume est échantillonné sur une grille rectilinéaire. Les lignes horizontales dans la figure représentent les tranches des données de volume vues dans la section parcourue. Après la transformation, les données de volume ont été décalées parallèlement à l'ensemble des tranches qui sont perpendiculaire à la direction d'observation et les rayons de vision sont perpendiculaires aux tranches. Pour une transformation perspective, il faut changer l'échelle de chaque tranche qui doit être ensuite décalé comme indiqué schématiquement dans la figure 2.2.b.

L'implémentation de *Lacroute* et *Levoy*, exécuté sous une station de travail **SGI Indigo**, donne un temps de calcul d'une seconde pour le rendu de 256^3 voxels d'une donnée médicale.



a. Projection parallèle. b. projection perspective.

Figure 2.2 : Principe de *Levoy* et *Lacroute* pour le rendu volumique.

Pour des textures volumiques, on considère le texel comme une série d'images 2D plaquées au dessus de la surface avec un décalage progressif. A noter que le hardware graphique permet de tracer très rapidement un polygone texturé dans une scène 3D, d'où le gain de temps colossal que l'on escompte par rapport aux méthodes existantes de rendu des texels.

3. Représentation.

L'idée d'avoir une couche épaisse sur un objet est une notion qui a été introduite par *Kajiya* et *Kay* en 1989, en proposant un modèle fourrure. Par la suite ce modèle a largement été amélioré par *F. Neyret* [NEY96], pour avoir un modèle général plus efficace en évitant les contraintes soulevées par *Kajiya*. Une primitive de réflectance particulière, comme c'est le cas dans le modèle de *Kajiya* et *Kay*, n'autorise que des objets particuliers. *F. Neyret* a donc proposé une primitive paramétrable (l'ellipsoïde) qui est capable de modéliser de nombreux types de formes.

Cependant, le modèle des textures volumiques introduit par *Neyret*, valable pour le lancer de rayon, n'était pas adapté à l'utilisation avec un algorithme comme le z-buffer et une représentation du volume de référence en couches. On l'a donc adapté au traitement en couche et au z-buffer de manière à pouvoir utiliser les fonctionnalités d'***OpenGL*** [MEY98].

Dans cette partie, on va présenter la méthode utiliser pour générer la surface à habiller d'une part, et celle pour représenter le volume de référence d'autre part.

4. La modélisation de la surface.

La surface utilisée comme support de notre texture volumique (l'ensemble des texels) est décrite par des polygones quelconques (en se basant sur un maillage surfacique de la surface à habiller). Pour la souplesse et la généralité de notre modèle, il faut que le nombre et l'orientation des texels que l'ont plaque sur la surface ne dépendent pas du maillage. Pour respecter ceci on va introduirai la notion de boîte.

4.1. Notion de boîte.

Pour chaque polygone décrivant la surface, on appelle boîte la portion de couche volumique qui lui correspond. Dans le cas du modèle de *Kajiya* et *Kay* où la surface est décrite par des quadrilatères et où chaque texel se plaque exactement sur un quadrilatère (voir figure 2.3 à gauche) ; la boîte contient en fait tout le texel. Mais dans le cas général un texel est placé sur la surface de manière complément indépendante du maillage de celle-ci (voir figure 2.3 à droite). Le texel va se déformer pour être collé exactement à la surface. Donc, chaque texel sera en fait composé de plusieurs boîtes, éventuellement même de fractions de boîtes [MEY98]. Une boîte est donc le volume se trouvant sur chaque polygone de la surface (voir figure 2.3 à droite).

Les arêtes verticales d'une boîte sont en fait des vecteurs réglables par l'utilisateur, initialement confondues avec les normales à la surface (que l'on obtient en faisant par exemple une moyenne des normales des faces adjacentes au point). Ces arêtes verticales peuvent être perturbées, elles ne sont pas contraintes à rester les normales à la surface. L'utilisateur définit une épaisseur qui correspond à la longueur de ces normales et donc aussi à la distance entre la base et le sommet de la boîte.

Une boîte est ainsi définie [MEY98] par $2N$ points : N points pour la base et N points pour le sommet. Pour indiquer comment se situe le boîte dans le texel il faut ajouter aux points de la base des coordonnées textures U, V. Ces coordonnées textures s'obtiennent suivant le même principe que pour le plaquage d'une texture 2D sur une surface. Le nombre de boîtes dépend directement du maillage de la surface (Le nombre de boîtes est égal au nombre de polygones de la surface). Donc la complexité, en nombre de faces texturées à afficher, est en fonction du maillage et non pas du nombre de texels affichés.

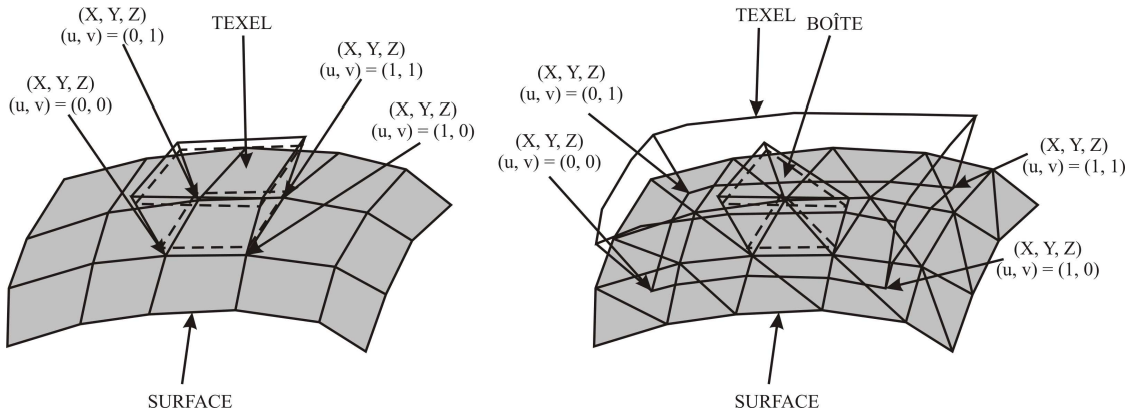


Figure 2.3 : Plaquage d'un texel sur une surface.

5. Méthode de *Meyer* pour une nouvelle représentation du volume de référence.

Le nouveau volume de référence peut être vus comme un ensemble de N images 2D de taille $X \times Y$ au format RGBA (les trois composantes de couleur *Rouge*, *Vert*, *Bleu* et la transparence *Alpha*). On peut aussi le voir comme un volume de $N \times X \times Y$ voxels ayant chacun quatre composantes (les trois de couleurs et la transparence) [MEY98]. Ce volume de référence ne contient plus d'informations sur la réflectance puisque la couleur précalculée est stockée à la place, ce qui implique que le rendu des ombres et de l'éclairage est fixé lors de sa construction. C'est une limitation importante, mais c'est le prix à payer pour accélérer le temps de rendu. A noter que le coefficient alpha pour la transparence est indispensable sinon la face du dessus cacherait une partie des faces d'en dessous.

La taille d'un texel de dimension 32×64×64 est de 512Ko ce qui reste acceptable pour une machine actuelle. Comme on le verra un peu plus loin, il est stratégique que toutes les données tiennent dans la mémoire texture, sinon les performances se dégradent fortement.

Pour le moment, on vient de présenter l'information essentielle que l'on stocke dans ce nouveau volume de référence mais, on verra ensuite que d'autres informations sont nécessaires.

5.1. La génération du volume de référence.

Il existe plusieurs méthodes de création de volume de référence. *Meyer* [MEY98] s'intéresse plus particulièrement à deux méthodes pour générer des volumes de références :

- La conversion d'objets polygonaux vers la représentation en couche ; Comme la nouvelle représentation du volume de référence est très différente de celle utilisée avec l'algorithme de lancer de rayon, on va proposer une nouvelle technique basée sur le rendu avec l'algorithme du z-buffer et l'utilisation des plans de ***clipping***.
- L'utilisation des textures 3D de *Perlin* comme champs de hauteurs. En utilisant une image 2D comme champ de hauteurs on génère un volume de référence. On utilise une image de *Perlin* parce qu'elle donne un bon résultat.

La représentation en couche introduite ici conduit à montrer l'intérieur des objets sous certain point de vue. Pour palier à cet inconvénient on décrit donc un algorithme simple de remplissage.

5.1.1. Conversion des représentations polygonales.

La représentation du volume de référence introduite dans ce travail étant différente de celle utilisé en lancer de rayons. On va donc présenter ici une nouvelle méthode basée sur l'utilisation des plans de ***clipping***.

a. La "coupe" de l'objet en des tranches.

Nous allons construire des texels où chaque tranche sera obtenue en calculant le rendu d'un plan de coupe d'un objet polygonal classique. Pour cela on va utiliser les fonctionnalités graphiques

permettant le rendu d'un objet en utilisant des plans de *clipping*[1]. En plaçant deux plans de *clipping* de manière très rapprochée et parallèle on obtient ainsi une vue en "coupe" de l'objet [MEY98].

On calcule les N images de l'objet polygonal en utilisant simplement le rendu d'***OpenGL*** avec ces 2 plans de *clipping* en les translatant d'un pas fixe pour chaque image. On obtient ainsi N coupes de l'objet ; chacune de ces coupes est une couche du texel.

Cette méthode pose un problème [MEY98] : les objets polygonaux sont creux par définition, donc la coupe de l'objet forme un contour avec un "trou" au milieu. Par exemple un tronc d'arbre sera creux, seulement les bords du tronc seront visibles sur l'image de coupe. Cela pose des problèmes à l'affichage du texel dès que l'angle de vue s'écarte un peu de la normale du texel (voir figure 2.4). Il faut donc utiliser un algorithme de remplissage.

b. Transformation des contours en sections pleines.

Le but de ce remplissage est de rendre "plein" l'intérieur des objets, puisque l'on va voir à l'intérieur de l'objet. Il ne faut pas simplement remplir mais colorier de façon à ce que les pixels de l'intérieur se confondent le mieux possible avec les pixels du contour [MEY98].

Ce remplissage va s'effectuer en deux étapes. La première étape consiste à marquer l'intérieur de l'objet, et la deuxième étape consiste à remplir l'intérieur de l'objet avec une couleur obtenue en interpolant celle des bords.

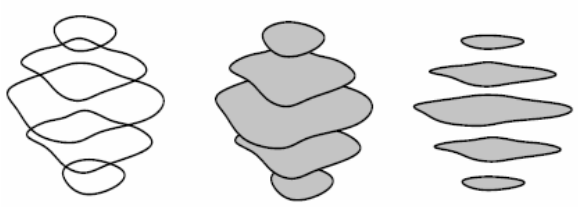


Figure 2.4 : Gauche : contour. Milieu : contour rempli. Droite : lorsque l'angle de vue est grand on voit à travers le texel.

Le marquage de l'intérieur de l'objet.

En effet, le marquage de l'intérieur d'un objet n'est pas une chose triviale dans le cas général. Pour les texels, on est contenté de gérer le cas des objets dont la coupe verticale donne un contour convexe. On effectue ce qui suit pour toutes les colonnes du texel : on va parcourir la colonne de bas en haut, dès qu'on traverse un bord opaque, on vérifie que l'objet a bien un bord qui se trouve au dessus dans la colonne (S'il n'y en a pas c'est que cette colonne est tangente au contour de l'objet et donc on reste à l'extérieur de l'objet). S'il existe un bord plus haut dans la colonne, on est sûr d'être dans l'objet et donc on marque les points entre le bord inférieur et le bord supérieur comme étant à l'intérieur [MEY98].

Cette technique est limitée mais elle a l'avantage d'être rapide et simple à implémenter. Pour une implémentation plus complète il faudrait utiliser un algorithme plus complexe, inspiré des méthodes classiques en 2D de remplissage de formes.

[1] Un plan de *clipping* sépare l'espace en deux parties : une partie qui sera visible au rendu et une autre non visible.

Le remplissage de l'intérieur de l'objet.

Cette partie va travailler sur chaque image 2D (i.e. chaque tranche) séparément [MEY98]. Ce remplissage se fait itérativement jusqu'à ce que tous les points marqués aient été coloriés. Lorsque l'on tombe sur un point à colorier on lui donne comme couleur la moyenne des couleurs de ses voisins (au sens de la quatre connexité) qui sont colorés. Si les quatre voisins n'ont pas de couleur (parce qu'ils sont encore marqués "à colorier") la couleur du point n'est pas changée et il reste marqué "à colorier".

Le résultat de cette coloration donne une couleur interne qui est influencé par tous les bords, c'est-à-dire que plus on se trouve près du bord plus la couleur est proche de celle du bord. Quand on regarde un texel de côté on voit donc bien en moyenne la couleur du bord et donc même si l'on voit à l'intérieur la légère variation de couleur n'est pas choquante.

5.1.2. Utilisation des textures de *Perlin* pour générer un texel.

Les textures de *Perlin* vont nous servir à créer des texels qui ont une forme très complexe, tordue et détaillée mais dont la forme globale n'est pas très bien connue à priori. Pour construire le volume de référence on utilise une image de *Perlin*. Pour obtenir ces données on va implémenter un moteur de texture 3D de *Perlin*. Les détails du moteur de *Perlin* sont donnés dans ce qui suit.

a. Textures 3D de *Perlin* : les deux fonctions de bruit.

Le modèle de texture procédurale de *Perlin*, introduit en 1985, est un modèle volumique [MIN01] : il s'agit d'une fonction permettant d'associer une couleur à tout point de l'espace. Les textures procédurales sont modélisées en utilisant une équation mathématique et ne nécessitent pas d'acquisition et de stockage d'image. *Perlin* a utilisé deux fonctions basées sur le bruit pour construire ces textures [MEY98] :

- La fonction *Noise* produit une courbe aléatoire continue dérivable à valeurs dans [0,1].
  Pour tous les points à coordonnées entières (x, y, z) de l'espace
  - prendre au hasard a, b, c, d.
  - calculer d' = d - (a × x + b × y + c × z).

  Pour tous les points à coordonnées réels (x, y, z) de l'espace :

  Si (x, y, z) ont trois entiers **alors** *Noise*(x, y, z) = a × x + b × y + c × z + d'

  Sinon on fait une interpolation cubique de a, b, c, d' avec les points de coordonnées entières les plus proches et *Noise*(x, y, z) = a × x + b × y + c × z + d'.
- Ainsi que la fonction *Turbulence*(P) qui produit un aspect fractal :

$$\sum \frac{1}{2^i} \times Noise(2^i \times P).$$

Cette fonction a des variations aléatoires mais douces : deux points proches ont des valeurs de couleurs proches tandis que deux points éloignés sont décorrélés [MIN01].

b. Hypertextures de *Perlin.*

Perlin a utilisé la fonction sphère qui renvoie la densité en (x, y, z) d'une hypertexture représentant une sphère de rayon *r* et d'épaisseur de bord *s* (la densité de cette hypertexture ne passe pas de 1 à 0 brutalement mais décroît linéairement sur l'épaisseur *s*) :

Fonction sphère(x, y, z, rayon r, soft s)

rayon ← $(x^2 + y^2 + z^2)^2$

Si $rayon < (r - \frac{s}{2})^2$ **Alors** 1

Sinon Si $rayon > (r - \frac{s}{2})^2$ **Alors** 0

Sinon $\frac{(r - \frac{s}{2})^2 - rayon}{(r - \frac{s}{2})^2 - (r + \frac{s}{2})^2}$

On peut imaginer la construction de toutes sortes d'autres objets (cube, cylindre, ...) de façon similaire.

La fonction de bruit *Noise* va servir à perturber la surface de la sphère. La fonction de densité de la sphère perturbée est :

$$D(X) = sphere(X + \frac{1}{f} \times \overrightarrow{Noise}(f \times X)).$$

Où *f* est la fréquence des perturbations et 1/*f* l'amplitude (le fait de lier ainsi l'amplitude à la fréquence est une caractéristique d'une fonction fractale).

Sur le même principe *Perlin* a créé une boule de feu à l'aide de la fonction *Turbulence* :

$$D(X) = sphere(X + \overrightarrow{Turbulence}(f \times X)).$$

La couleur de la sphère va dépendre de la densité, rouge pour les faibles densités et jaunes pour les hautes densités.

Pour agir sur l'opacité, *Perlin* a définit deux fonctions qui vont rendre les transitions plus ou moins brusques dans une hypertexture :

- une fonction *bias* qui vérifie *bias*(0) =0, *bias*(1) =1 et *bias*(0.5) = b $bias_b(t) = t^{frac(ln(b \times ln(0.5)))}$.
- une fonction *gain* du type :
  $gain_g$(t) = si $t < 0.5$ alors $bias_{1-g}(2t)/2$ sinon 1 - $bias_{1-g}(2-2t)/2$.

Des fonctions booléennes vont permettre de combiner ces hypertextures :

$A \cap B = a(x) \times b(x)$.

$\overline{A} = 1 - a(x)$.

$A - B = A \cap \overline{B} = a(x) - a(x) \times b(x)$.

$A \cup B = \overline{\overline{A} \cap \overline{B}} = a(x) + b(x) - a(x) \times b(x)$.

Perlin a construit ainsi un cube érodé aux coins en faisant l'intersection d'une sphère perturbée et d'un cube.

Cette texture de *Perlin* 2D est utilisée comme champ de hauteur. Le niveau de gris de chaque pixel va nous donner la hauteur de la colonne dans le texel. Dans un premier temps on va utiliser le niveau de gris comme couleur de la colonne. Ceci donne les vallées sombres et les sommets clairs (texel de gauche de la figure 2.7).

Dans un deuxième temps, on peut utiliser une deuxième image de *Perlin* (voir figure 2.6 à droite) pour "mixer" la couleur et ainsi obtenir un texel qui a des variations de couleurs différentes selon le coté par lequel on le regarde (texel de droite de la figure 2.7). En fait on effectue une moyenne géométrique entre la couleur de la première image et la couleur de la deuxième image.

Pour le remplissage, on utilise la deuxième partie de l'algorithme vu au paragraphe précédent (5.1.1-b) pour colorier l'intérieur du volume de référence. Au préalable, il faut d'abord rendre continu les bords (voir la figure 2.5) car la forte pente présente dans ces données génère des contours incomplets. Pour cela on parcourt tous les points du texel et, quand un voxel n'est pas invisible on complète la colonne en dessous de lui jusqu'au voisin le plus bas en interpolant la couleur. D'autre part, on peut générer un deuxième modèle de volume de référence par une coloration de la colonne en dessous jusqu'à la base de la texture.

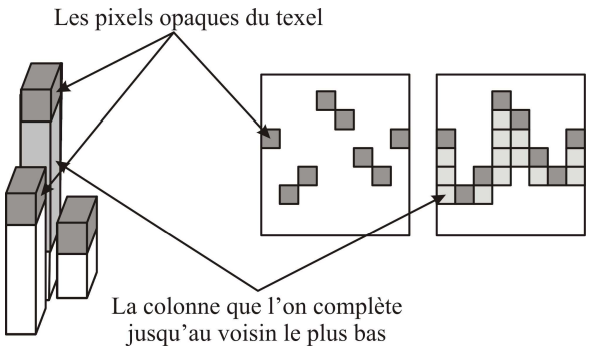


Figure 2.5 : Algorithme de remplissage de la colonne. Gauche : vue en 3D. Droite : Vue en coupe verticale.

Figure 2.6 : Gauche : Image de *Perlin* utilisée pour la profondeur. Droite : Image de *Perlin* utilisée, pour l'ombrage.

Figure 2.7 : Texels construits à partir des images de *Perlin*. Gauche : sans ombrage. Droite : avec ombrage.

6. La coloration de voxel pour la reconstruction d'une scène 3D.

Cette méthode, introduite par *Seitz* en 1997, permet la reconstruction d'une scène 3D en utilisant des images acquises du monde réel, et en utilisant la discrétisation de l'espace 3D en un ensemble de voxel comme une structure de donnée permettant de reconstruire la scène.

6.1. Le problème de "Coloration de Voxel".

Dans cette section on va présenter une technique, introduite par *Seitz* [Sei97], de représentation de l'espace scène pour avoir une information de correspondance dense à partir de vues multiples V_0, …, V_n. On va reconstruire une scène 3D colorée, à base de voxels et qui est cohérente en utilisant toutes les vues de base. Cette scène peut être reprojetée pour synthétiser de nouvelles vues. Cette approche a une particularité garantissant une scène cohérente quand toutes les suppositions sont rencontrées.

Le problème de coloration de voxel est de faire assigner des couleurs (radiances) à des voxels (points) dans un volume 3D afin de garantir une cohérence[1] avec un ensemble d'images de base (Figure 2.8). Plus formellement, une scène 3D *S* est représentée par un ensemble de voxels *Lambertiens* opaques (des éléments de volume), dont chacun occupe un volume fini et homogène de la scène, centré à un point $V \in S$ et a une couleur fixe [Sei97].

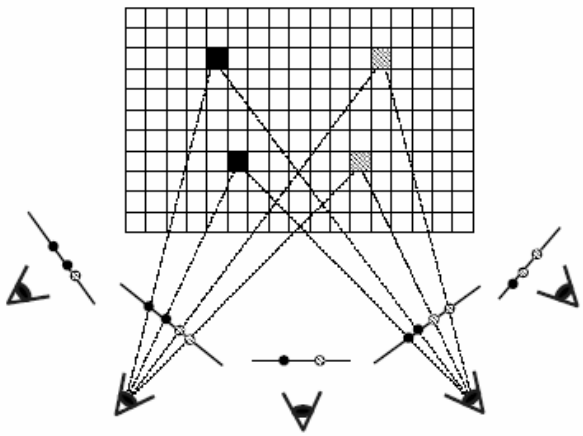


Figure 2.8 : La coloration de voxel ; étant donné un ensemble d'images de base et une grille de voxels, on va attribuer les valeurs de couleurs aux voxels de telle façon que la scène soit cohérente avec toutes les images.

[1] Le rendu des voxels colorés de chaque point de vue de base doit reproduire l'image originale aussi étroitement que possible

On suppose que la scène est entièrement contenue dans un volume englobant. Un "*espace voxel*" noté par **v** représente l'ensemble de tout les voxels dans le volume englobant. Étant donné un pixel $p \in I$ et une scène S, on fait référence au voxel $V \in S$, qui est visible dans I et se projette à p par $V = S(p)$. On dit qu'une scène S est *complèt*e en ce qui concerne un ensemble d'images si, pour chaque image I et chaque pixel $p \in I$, il existe un voxel $V \in S$ tel que $V = S(p)$. On dit qu'une scène complète est *cohérente* avec un ensemble d'images si, pour chaque image I et chaque pixel $p \in I$,

$$\text{Couleur}(p, I) = \text{couleur}(S(p), S)$$

En supposant que les images représentent les projections de la même scène *Lambertienne*, il est clair qu'il *existe* une coloration cohérente de voxel, correspondant à l'ensemble de points et des couleurs sur les surfaces de la scène. Un ensemble d'images peut être compatible avec plus qu'une scène rigide. En déterminant l'occupation spatiale d'une scène, c'est-à-dire, un voxel contenu dans une scène cohérente ne peut pas être contenu dans une autre (voir la figure 2.9).

En outre, un voxel peut être contenu dans deux scènes cohérentes, mais avoir des couleurs différentes dans chacune (voir la figure 2.10). Par conséquent, des contraintes complémentaires sont nécessaires pour que le problème soit bien posé. Le calcul de coloration de voxel pose un autre défi. Observez que l'espace sous-jacent est combinatoire : Une grille de N × N × N voxels, chacun avec M assignements possibles de couleurs rapporte 2^{N^3} scènes possibles et M^{N^3} assignements possibles de couleurs [Sei97].

6.2. Couleur invariante.

Étant donné une multiplicité de solutions du problème de coloration de voxels, la seule façon de récupérer l'information d'une scène intrinsèque est par *les propriétés invariantes* qui sont satisfaites par c*haque* scène cohérente. Par exemple, considérant l'ensemble de voxels qui sont contenus dans chaque scène cohérente. Ces propriétés fournissent une information absolue sur la scène réelle, mais qui sont relativement rares; quelques images ne peuvent rapporter aucune (voir, par exemple, la figure 3.9).

> Un voxel V a **une couleur invariable** en ce qui concerne un ensemble d'images si : (1) V est contenu dans une scène cohérente avec les images et (2) pour chaque paire de scènes cohérentes S et S', $V \in S \cap S'$ implique que *couleur* (V, S) = *couleur* (V, S').

Contrairement à l'invariance de la forme, l'invariance de couleur n'exige pas qu'un point soit contenu dans chaque scène cohérente. Particulièrement on peut montrer que l'union de toutes les couleurs invariantes eux-mêmes rapporte une scène cohérente, c'est-à-dire, une coloration complète de voxel (la figure 2.11). Des contraintes complémentaires sont nécessaires pour que le problème soit plus facile.

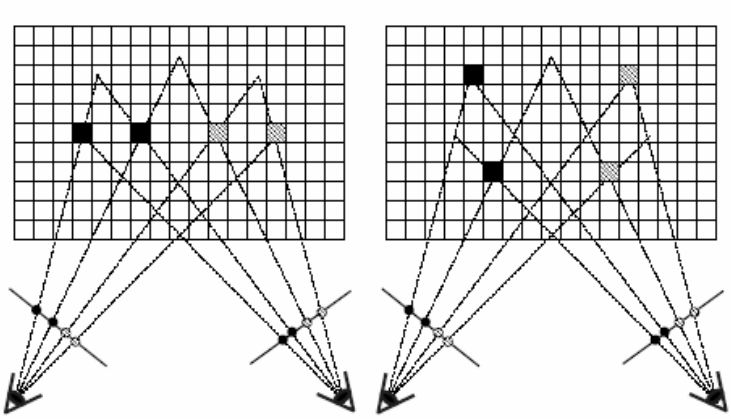


Figure 2.9 : Ambiguïté Spatiale.

L'ambiguïté spatiale correspond au cas où deux colorations de voxel apparaissent identique de ces deux points de vue, malgré qu'ils n'ont pas des voxels colorés en commun.

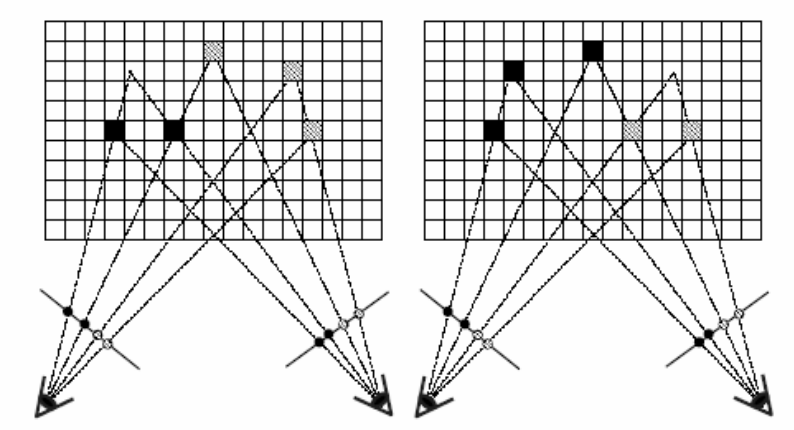


Figure 2.10 : Ambiguïté de couleur.

L'ambiguïté de couleur correspond au cas de présence d'un voxel (la deuxième rangée au centre) qui a deux assignements différentes de couleur dans les deux scènes. Cependant, les deux colorations de voxel apparaissent identique des deux points de vue différentes.

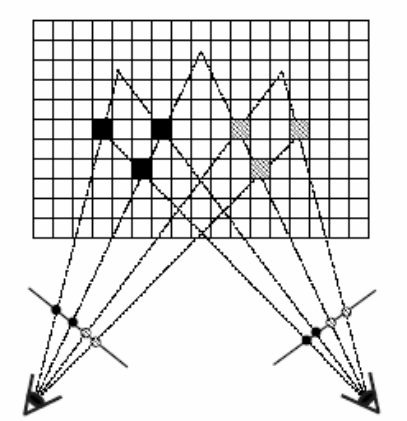


Figure 2.11 : Couleur Invariante.

Chacun des six voxels de la figure 2.11 a une seule couleur dans la scène cohérente dans laquelle il est contenu. La collection de toutes telles couleurs invariantes forme une coloration cohérente de voxels que l'on dénote par $\overline{S}$. Notez que le voxel ayant deux assignements de couleur de la figure 2.11 n'est pas contenu dans $\overline{S}$.

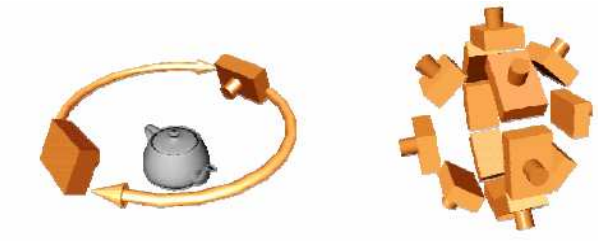
6.2.1. La contrainte de visibilité ordinale.

Notez que la couleur invariante est définie pour l'ensemble $\aleph$ de toutes les scènes cohérentes (définissant un espace combinatoire). D'un point de vue quantitative, une recherche explicite dans cet espace n'est pas faisable. Pour rendre ce problème plus facile, on va présenter une nouvelle contrainte géométrique de positionnement de la caméra par rapport à la scène pour simplifier l'analyse. Cette c*ontrainte de visibilité ordinale* permet l'identification de l'ensemble de couleurs invariantes comme un point de limite de l'ensemble $\aleph$. Donc, elles peuvent être calculés directement, via un passage simple à travers l'espace voxel.

Soit P et Q deux points de l'espace scène et soit *I* l'image d'une caméra centrée à C. On dit que P *occulte* Q si P se trouve sur le segment de ligne $\overline{CQ}$. Nous exigeons que les caméras d'entrée soient placées à des positions pour qu'elles satisfassent la contrainte suivante :

Contrainte de visibilité ordinale : Il existe une norme tel que pour toutes points P et Q de la scène et des images d'entrée *I*, P occulte Q dans *I* seulement si $\|P\| < \|Q\|$.

On appel une telle norme *compatible occultation*. Supposant que les caméras sont distribuées sur un plan et la scène est entièrement au-dessous de ce plan, comme il est indiqué dans la figure 2.12 (a). Pour chaque point de vue, la visibilité relative de deux points quelconques de la scène dépend entièrement du point le plus proche au plan, donc nous pouvons définir cette norme pour être la distance au plan. Plus généralement, la contrainte de visibilité ordinale est satisfaite chaque fois qu'aucun point de la scène n'est contenu dans la coque convexe *C* des centres des caméras. Ici on va utiliser la norme compatible occultation $\|P\|_C$, définie comme étant la distance *Euclidienne* de *P* à *C* (*C* est définit comme le "*volume des caméras*"). La figure 2.12 représente deux configurations utiles[1] de caméra qui satisfont cette contrainte.



a. regard vers l'intérieur b. regard vers l'extérieur

Figure 2.12 : Configurations compatibles de la caméra : (a) une caméra aérienne faisant face vers l'intérieur et qui se déplacé de 360 degrés autour d'un objet et (b) un ensemble de caméras faisant face vers l'extérieur distribuées autour d'une sphère.

[1] La scène doit être en dehors de l'enveloppe convexe des centres de caméras.

6.3. Algorithme de coloration de voxel.

On va décrire maintenant comment calculer $\overline{S}$, l'ensemble de voxels ayant une couleur invariable pour chacun, en utilisant une discrétisation du volume de la scène, et en exploitant la contrainte de visibilité ordinale. Cette contrainte limite les configurations possibles d'une vue de base, mais l'avantage est que ces rapports de visibilité sont très simplifiés. Particulièrement, la division de la scène en une série d*es couches* de voxels qui obéissent au rapport de visibilité précédemment définit devient possible : pour c*haque* image d'entrée, des voxels occultent seulement d'autres voxels qui sont dans les couches suivantes. Par conséquent, les rapports de visibilité sont résolus en traversant les voxels une couche à la fois [Sei97].

6.3.1. Décomposition de la scène en couches.

Pour formaliser cette idée, on va définir une subdivision de l'espace 3D en couches de voxels de distance uniforme au volume de caméra :

$$\nu_d = \{V \mid \|V\| = d\}$$

$$\nu = \bigcup_{i=1}^{r} \nu_{d_i}$$

Où d_1, …, d_r est une séquence de nombres croissants et $\|.\|$ est une norme compatible occultation.

Comme illustration, considérant un ensemble de vues placées le long d'une ligne faisant face à une scène bidimensionnelle (la figure 2.13 (a)). Le choix de $\|.\|$ pour être la distance orthogonale à cette ligne provoque une série de couches parallèles linéaires qui s'éloignent des caméras. Remarquez que pour n'importe quel deux voxels *P* et *Q*, *P* peut occulter *Q* d'un point de vue de base seulement si *Q* est dans une couche plus haute que *P* [Sei97].

Le cas linéaire est facilement généralisé pour n'importe quel ensemble de caméras satisfaisant la contrainte de visibilité ordinale. La figure 2.13 (b) représente une subdivision en couches pour le cas de caméras faisant face vers l'extérieur. Ce type de géométrie de la caméra est utile pour l'acquisition de d'images *panoramiques* d'une scène. Un ensemble valable de couches correspond à une série de rectangles qui s'affichent devant le volume de la caméra. La couche 0 est la boîte englobante *B* alignée sur l'axe des centres des caméras et les couches suivantes sont déterminées en étendant uniformément la boîte d'une unité à la fois. Cet ensemble de couches correspond à une norme donnée par la distance L_1 à *B*.

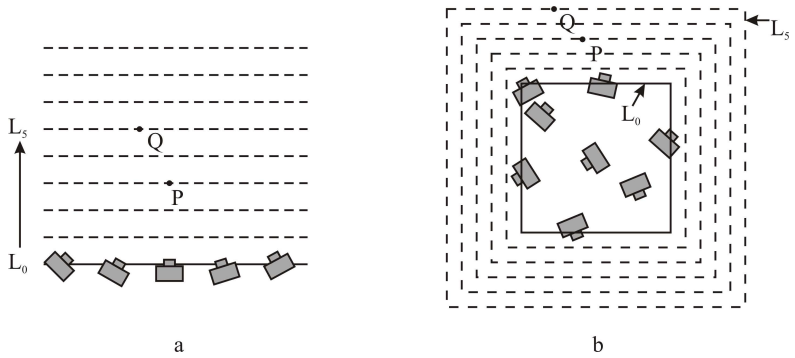


Figure 2.13 : Traversée de scène représentée par couches. Les voxels peuvent être divisés en une série des couches de distances croissantes au volume des caméras. (a) Couches pour des caméras positionnées le long d'une ligne. (b) Couches pour des caméras placées sur un plan.

La décomposition d'une scène 3D en couches peut être faite de la même manière. Dans le cas 3D les couches deviennent les surfaces qui s'affichent devant le volume de la caméra. Cette stratégie de subdivision est particulièrement très utile (la figure 2.13 (b)), dans laquelle chaque couche est un cube aligné suivant un axe. L'avantage de ce choix est l'efficacité du calcul et de la traverse des couches.

6.3.2. La cohérence de voxel.

Supposant maintenant que les images sont discrétisées sur un réseau de pixels non chevauchés et finis :

- Si un voxel *V* n'est pas entièrement occlus dans l'image I_j, sa projection chevauche un ensemble π_j non vide de pixels de l'image [Sei97].
- Un voxel cohérent doit se projeter sur un ensemble de pixels avec des valeurs égales de couleurs.
- Nous évaluons la corrélation λ_V des couleurs de pixel pour mesurer la probabilité de cohérence du voxel. Soit *s* l'écart type et *m* le cardinalité de $\bigcup_{j=0}^{n} \pi_j$. Il faut utiliser un seuil d'erreur pour l'évaluation de la formule : $\lambda_V = s$.

Alternativement, une mesure statistique de cohérence d'un voxel peut être employée. Particulièrement supposant que l'erreur perceptible (l'exactitude de mesure d'irradiance) possède une distribution normale avec l'écart type σ_0. La cohérence d'un voxel peut être estimée en utilisant le test de proportion de probabilité, distribuée comme χ^2 avec n-1 degrés de liberté :

$$\lambda_V = \frac{(m-1)s^2}{\sigma_0^2}$$

Si σ_0 est inconnu, elle peut être estimée en imaginant une surface homogène et en calculant l'écart type s_0 de m' pixels de l'image. Dans ce cas, l'équation précédente doit être remplacée par :

$$\lambda_V = \frac{s^2}{s_0^2}.$$

Cette équation a une distribution F avec *m*-1 et *m*'-1 degrés de liberté.

6.3.3. Un algorithme à un seul passe.

Pour évaluer la cohérence d'un voxel, on doit d'abord calculer, l'ensemble des pixels qui chevauchent la projection de *V* sur I_j. En négligeant des occultations, le calcul de la projection d'un voxel sur une image est directe et se base sur la forme du voxel et la configuration connue de la caméra. On utilise le terme *"empreinte"* pour dénoter cette projection qui correspond à l'intersection avec le plan image de tous les rayons du centre de caméra intersectant le voxel. La représentation des occultations est plus difficile, cependant nous devons faire attention pour inclure seulement les images et les positions de pixels dont *V* doit être *visible*. Cette difficulté est résolue en employant la contrainte de visibilité ordinale pour visiter les voxels dans un ordre compatible occultation et en *marquant* des pixels correspondants [Sei97].

Au départ, tous les pixels sont non marqués. Quand un voxel est visité, π_j est défini comme l'ensemble des pixels *non marqués* qui chevauchent l'empreinte de *V* dans Ij. Quand un voxel est évalué et on trouve qu'il est cohérent, tout les *m* pixels dans π_j sont marqués. Cette stratégie est suffisante pour assurer que π_j contient seulement les pixels dont chaque voxel est visible, c'est-à-dire, S (*p*) = *V* pour chaque pixel *p*. La phase de marquage de pixels peut être retardée jusqu'à ce que tout les voxels dans une couche soient évalués.

L'algorithme complet de coloration de voxel peut maintenant être présenté comme suit [Sei97] :

$\overline{S} \leftarrow \phi$

```
Pour i=1, …, r faire                    //Itérer à travers les couches
  Pour chaque V∈ V_di faire             //Itérer à travers les voxels dans la couche
    Pour j=0, …, n faire                //Projeter le voxel sur chaque image
      Calculer l'empreint ρ de V dans Ij
      πj ← { p ∈ ρ | p est non marqué }
    Fin Pour
    Calculer λV                         //Evaluer la cohérence du voxel
    Si m>0 et λV<seuil alors
      S̄ ← S̄ ∪ {V}                       //Colorer le voxel
      π ← π ∪ (⋃_{j=1}^{m} πj)           //Mémoriser les pixels à marquer
    Fin Si
  Fin Pour
  Marquer les pixels dans π
Fin Pour
```

Le seuil utilisé correspond à l'erreur maximale de corrélation permise. Une petite valeur du seuil résulte en une reconstruction précise mais incomplète. D'autre part, un seuil plus grand rapporte une reconstruction complète, mais qui inclut quelques voxels faux.

7. L'intégration du technique de coloration pour la génération d'un volume de référence.

Nous avons introduit, dans cette partie, une nouvelle méthode de génération du volume de référence, basé sur la technique de coloration de voxels (Voxel Coloring) de *Seitz* [Sei97]. Comme cette technique permet une reconstruction tridimensionnelle d'une scène à partir d'un ensemble d'images en utilisant une discrétisation 3D de l'espace scène (ensemble de voxels), nous pouvons utiliser cet ensemble de voxels pour la génération de notre volume de référence à base de couches. Le processus de passage entre ces deux représentations est illustré dans la figure 2.14.

Il est aisé d'avoir une correspondance entre un espace discrétisé en un ensemble de voxels et un ensemble de couches d'images. Un niveau d'un ensemble de voxels (apparaît en gras sur la figure 2.14 à gauche) correspond à une image (couche sur la figure 2.14 à droite), où l'information de couleur et de transparence de chaque pixel est acquise à partir du voxel correspondant dans le même niveau pour l'ensemble des voxels.

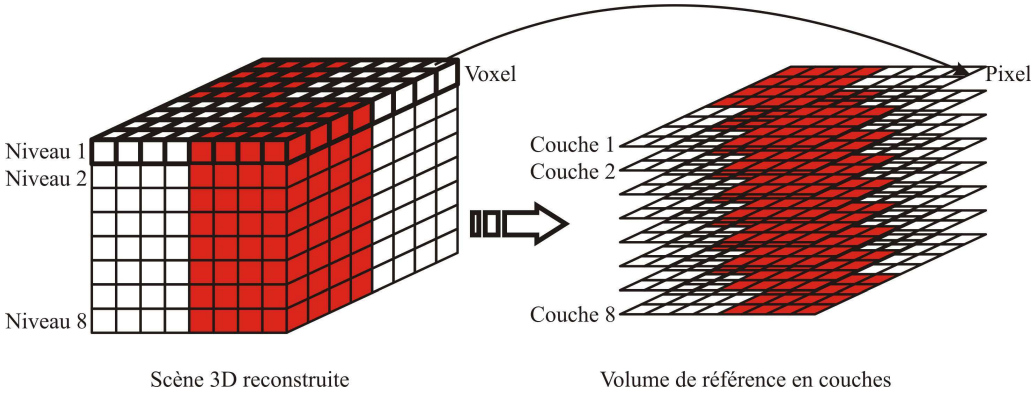


Figure 2.14 : Le passage entre une représentation à base de voxel et une autre à base de couches d'images.

Reconstruction à partir des images de synthèse.

Nous utilisons un ensemble d'images de synthèse, générées en utilisant un lancer de rayon modifié, pour une reconstruction plus aisé de la scène 3D. Un algorithme de lancer de rayon modifié consiste à calculer une image de profondeur (*Depth Image*) de la manière suivante :

- Lancer un ensemble de rayon à travers une grille de pixels (le plan image).
- Calculer pour chaque rayon lancé tous les points d'intersection avec la géométrie de la scène.
- Trier les points d'intersection selon leurs profondeurs.
- Calculer la couleur du point d'intersection le plus proche à l'observateur en utilisant le modèle d'illumination de *Phong*.
- Sauvegarder la profondeur du point d'intersection en plus de l'information de couleur dans une image de profondeur.

Après avoir calculer un ensemble d'images de profondeur autour des objets de la scène, nous pouvant reconstruire cette scène en utilisant ces images accompagnées d'autres informations tel que : la position de la caméra, les angles focaux de vision (verticale et horizontale) et la dimension de chaque image. Le processus de reconstruction est une projection inverse de l'ensemble de pixels de chaque image vers l'espace scène discrétisé en voxels. Cette opération permet de colorer les voxels transparents et d'avoir ainsi la même scène, mais qui est représentée cette fois par une série de voxels. Ce processus (illustré dans la figure 2.15) est effectué de la manière suivante :

- Initialement, nous construisons la grille 3D des voxels en discrétisant l'espace scène. Touts les voxels de la grille sont considérés comme transparents (La valeur de alpha est 0).
- Nous utilisons les informations accompagnant la caméra pour calculer la matrice M de changement de repère entre le repère objet et le repère caméra. $M = R_X \times R_Y \times T$ (où : R_X et R_Y sont les deux matrices de rotation pour ramener le vecteur de vision sur l'axe OZ, et T est la translation ramenant la caméra au centre de la scène.
- La matrice de changement de repère est utilisée pour calculer la direction de chaque rayon lancé à travers la grille des pixels image.
- Une fois la direction du rayon est calculée, nous calculons la position du pixel projeté dans l'espace scène en utilisant l'information de profondeur accompagnant le pixel de la manière suivante : $P = \vec{O} + \lambda \times \vec{D}$. Où : $\vec{O}$ est l'origine du rayon, $\vec{D}$ est le vecteur normé de la direction du rayon et λ correspond au profondeur du point *P*.
- Après avoir calculer la position 3D du point projeté, nous calculons la position du voxel correspondant dans la grille et nous attribuons la couleur du pixel au voxel trouvé, et la valeur alpha du voxel est 255.

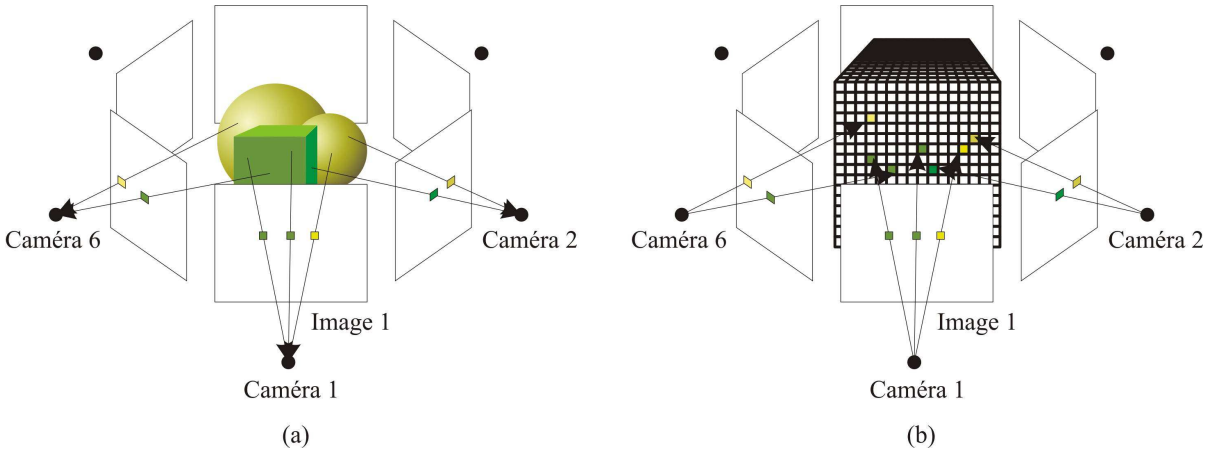


Figure 2.15 : Reconstruction à partir des images de profondeur (le sens du flèche indique le sens de projection) : (a) La création des images de profondeur en utilisant un lancer de rayon modifié (b) La projection inverse des pixels sur l'ensemble des voxels permettant de les colorer.

Après avoir colorer l'ensemble des voxels, nous pouvons générer notre volume de référence en utilisant la même méthode de passage entre la représentation à base de voxels et la représentation à base de couches illustrée dans la figure 2.14.

Pour cette méthode de création d'un volume de référence, il reste un problème qui se pose déjà ; la scène reconstruite est une représentation surfacique, c'est-à-dire que les couches qui vont être générées représentent les frontière des objet reconstruits. La solution est d'utiliser la méthode de transformation des contours en sections pleines vue au paragraphe 5.1.1 (la partie b).

8. Conclusion.

Nous avons présenté une méthode de modélisation de la surface à habiller et la construction des échantillons de référence de ce nouveau modèle de référence en utilisant les deux méthodes décrites par *Meyer*. L'une permettant de convertir n'importe quel objet polyédrique en utilisant des plans de ***clipping***. L'autre est une méthode procédurale basée sur l'utilisation d'une image 2D comme champ de hauteur. On a utilisé un moteur de *Perlin* pour créer les images 2D car celles-ci permettent de créer des motifs de bonne qualité.

Ensuite, nous avons présenté la méthode de coloration de voxels introduite par *Seitz* comme une méthode de reconstruction tridimensionnelle pour la création d'une scène 3D à partir des images réelles prises autour de la scène.

Enfin, nous avons introduit une nouvelle méthode de création d'un volume de référence basé sur celle de *Seitz* en introduisant un processus simple de passage entre une représentation à base de voxels et une autre basée sur un ensemble de couches d'images transparentes.

Dans le chapitre suivant, nous allons présenter la technique de plaquage de texture 3D à base de couches d'images sur une surface modélisée, en utilisant un maillage surfacique permettant de générer un ensemble de boîtes sur la surface.

Texture volumique à base de couches d'images : Le rendu du nouveau modèle

1. Introduction.

Nous allons exposer dans ce chapitre la manière de rendre une scène en utilisant la nouvelle représentation des textures volumiques (un volume de référence représenté comme un ensemble de couches superposées), ainsi que les contraintes qu'impose ***OpenGL*** pour l'utilisation du Z-Buffeur influant sur la façon avec la quelle nous allons rendre une scène.

Le rendu d'une scène passe par le rendu de toutes les boîtes comportant un ensemble de polygones transparents texturés (les couches images). Nous allons voir comment obtenir tous ces polygones dans la scène modélisée. Ensuite, nous discuterons les deux manières de rendre ces polygones.

2. Rendu d'une boîte.

Nous venons de voir dans le chapitre précédent la notion de boîte. En fait, c'est à partir des informations contenues dans une boîte que l'on obtient les coordonnées (dans l'espace scène *X*, *Y*, *Z* et dans l'espace texture *U*, *V*) des faces texturées que l'on affiche avec ***OpenGL*** [MEY98].

2.1. Les coordonnées des faces texturées.

Une boîte est définie par ses points de la base et du sommet. Il est donc aisé de calculer les coordonnées des différentes tranches par une interpolation linéaire entre un point de la base et un point du sommet (voir figure 3.1). Les coordonnées des points de ces faces ne sont recalculées que quand la boîte change, c'est-à-dire quand elle se déforme. Ce qui se produit soit quand la surface bouge, soit quand on applique une déformation sur les boîtes (pour simuler du vent par exemple).

Les coordonnées de la face *i* sont :

$A_i = A_0 + i/n \times \overrightarrow{A_0A_{n-1}}$

$B_i = B_0 + i/n \times \overrightarrow{B_0B_{n-1}}$

$C_i = C_0 + i/n \times \overrightarrow{C_0C_{n-1}}$

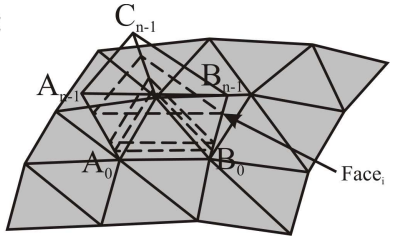


Figure 3.1 : Calcul des coordonnées des faces texturées d'une boîte.

2.2. Ordre d'affichage des tranches.

Dans ce paragraphe nous allons voir que ***OpenGL***, à cause de sa manière dont la technique du z-buffer gère la transparence des polygones, impose que l'affichage de faces transparentes se fasse par ordre de profondeur. Quand ***OpenGL*** affiche un nouveau polygone, il traite tous les pixels de ce polygone comme ceci [MEY98] :

Si la profondeur du point est supérieure à celle stocké dans le z-buffer on ne l'affiche pas, sinon on applique la formule suivante :

$$Couleurecran = Alphapoint \times Couleurpoint + (1 - Alphapoint) \times Couleurecran.$$

Plaçons nous dans le cas où la valeur z du point que l'on veut afficher est plus grande que celle existante dans le z-buffer. Ce point est abandonné par l'algorithme. Si le point du z-buffer appartenait à un polygone totalement opaque ce choix est bon. Par contre, si il appartenait à un polygone semi opaque, le nouvel point n'aurait pas d'être abandonné. Il faut faire une composition des couleurs, mais pour cela il faut se rappeler à la contribution de tous les polygones. Pour résoudre ce problème il faut afficher les faces de l'arrière vers l'avant, ainsi la loi de composition de couleurs vue au dessus donne le bon résultat [MEY98].

Pour chaque boîte on affiche donc les N faces par ordre de profondeur. Il existe deux ordres d'affichage possible, de la base vers le sommet (cas A de la figure 3.2) si l'oeil se trouve au dessus de la surface sous-jacente et du sommet vers la base (cas B) si l'oeil se trouve sous la surface (bien sûr ce deuxième cas n'est utile que si la surface est transparente). Le signe du produit scalaire entre le vecteur V allant de l'oeil au centre de gravité de la boîte et le vecteur normal N à la surface (voir figure 3.2) nous indique dans quel ordre il faut afficher les faces [MEY98].

3. Rendu de toute la surface texturée.

Nous disposons donc d'une peau volumique, découpée en un ensemble de boîtes correspondant aux facettes recouvrant la surface. Nous allons voir dans cette partie deux algorithmes qui sont envisageables pour effectuer le rendu d'une telle scène : Le rendu par boîte et le rendu par tranches identiques.

3.1. Rendu par boîte.

Une première méthode de rendu consiste à prendre chaque face de la surface et à afficher la boîte se trouvant au dessus. L'inconvénient de cette méthode est d'obliger le moteur graphique à changer de contexte texture pour chaque polygone tracé.

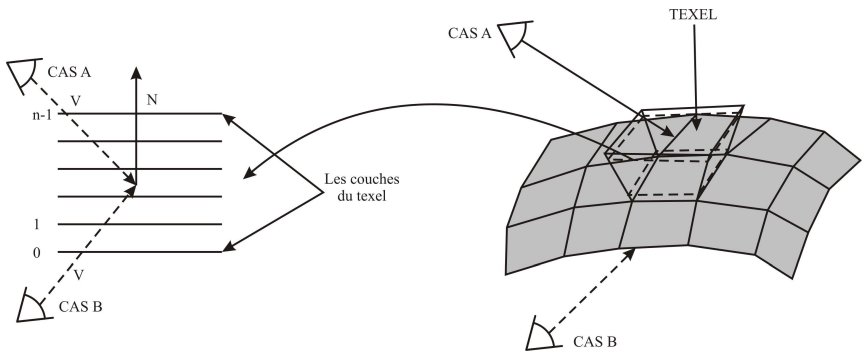


Figure 3.2 : Ordre d'affichage des faces : Cas A de 0 à n-1. Cas B de n-1 à 0.

L'algorithme utilisé est le suivant :

```
Pour i = toutes les boîtes faire
   Pour n = toutes les tranches de la boîte faire
       Charger la texture n
       Afficher la tranche n de la boîte i
   Fin pour
Fin pour
```

Si l'ensemble des textures représentant le volume ne tient pas dans la mémoire interne réservée aux textures le système doit constamment recharger des textures de la mémoire principale vers cette mémoire texture (la mémoire principale sert de zone de swap). Ceci est très coûteux et les performances s'écroulent. Ce cas se présentera si on implémente cet algorithme sur une "petite machine" où la mémoire texture est limitée. Ce cas où la mémoire texture n'est pas suffisante est pénalisant pour les performances mais même si toutes les textures tiennent en mémoire texture le changement de contexte reste une opération qui a un coût qu'il faut limiter au maximum [MEY98].

Comme le chargement d'une texture (couche) dans la mémoire associée correspond à un changement de contexte (l'utilisation de la fonction **glBindTexture** pour attribuer un identificateur à la texture et **glTexImage2D** pour charger la texture), nous aurions un changement de contexte $I \times N$ fois, où I est le nombre des boîtes sur la surface et N est le nombre de couches dans une boîte.

3.2. Rendu par tranche identique.

Même si l'ensemble des textures tient dans la mémoire texture, le fait de changer de contexte texture prend du temps. Ce qui a conduit à présenter un deuxième algorithme qui réduit au maximum les changements de contexte.

Cette deuxième approche consiste à afficher toutes les faces en utilisant la même texture dans une même étape. Comme le volume de référence se reproduit sur toute la surface on affiche la tranche *i* de toutes les boîtes puis on passe à la tranche suivante [MEY98]. L'algorithme utilisé est le suivant :

```
Pour n = toutes les tranches du texel faire
   Charger la texture n
   Pour i = toutes les boîtes faire
      Afficher la tranche n du texel i
   Fin pour
Fin pour
```

Cette deuxième approche est plus rapide que la première. En effet, avec la première méthode on va changer de contexte de texture N_b fois plus que dans la deuxième où N_b est le nombre de boîte (c'est-à-dire le nombre de polygones de la surface) ce qui fait beaucoup.

3.3. Problème restant dans l'ordre d'affichage des boîtes.

Nous avons vu au paragraphe 3.1 qu'il fallait afficher les faces transparentes par ordre de profondeur pour obtenir un résultat correct. La méthode de rendu qu'on a exposé jusqu'à présent ne respecte qu'un ordre partiel d'affichage des faces. L'ordre "du fond vers l'avant" n'est respecté que pour les faces d'une même boîte mais pas pour les boîtes elles-mêmes. Une boîte se trouvant derrière ne sera pas forcement affichée avant celle se trouvant devant. Si la boîte de derrière est tracée après celle de devant, elle n'apparaîtra pas sur l'image à cause de son *z* trop grand. Si la boîte de devant est totalement opaque le résultat est juste mais si elle n'est pas tout à fait opaque on devrait voir un bout de la boîte de derrière. Ce cas se présente lorsqu'il existe dans le volume de référence considéré dans son ensemble des zones translucides (i.e. des zones non complètement opaque et non complètement transparente) [MEY98].

Pour résoudre ce problème il suffit d'établir un ordre d'affichage des boîtes en fonction de leurs profondeurs (algorithme du peintre) mais dans ce cas il faut effectuer un prétraitement qui classe les boîtes par ordre de profondeur. Ceci prend beaucoup de temps machine, nous avons donc choisi d'utiliser un volume de référence qui n'est pas translucide, ce qui est le cas de la plus part des objets classiques.

4. Optimisations.

Dans cette partie on apporte d'abord une amélioration visuelle en modifiant la représentation vue au paragraphe 3.1. Lorsque l'angle entre l'axe de vision et le plan de la couche est très aigu on voit que la couche épaisse de texture est une superposition de couches, nous présenterons donc une solution à ce problème en utilisant trois directions de couches dans le volume de référence.

On exploite ici l'idée qu'il n'est pas nécessaire d'afficher toutes les tranches de tous les texels. C'est pourquoi on va exposer dans le paragraphe 4.2 deux critères de qualité complémentaires basés sur deux causes de l'erreur visuelle, qui permettront de savoir dans quelles conditions l'affichage d'un texel est satisfaisant. Nous utiliserons ces critères pour minimiser le nombre de tranches à tracer pour chaque texel affiché.

4.1. Trois directions de couches.

4.1.1. Principe.

Lorsque l'angle de vue du texel est très aigu on voit clairement que la surface est une superposition de faces (voir figure 3.4 à gauche cas A). L'intuition confortée par le critère de qualité que nous verrons au paragraphe suivant (4.2) nous indique qu'il faudrait plus de tranche pour corriger cette erreur.

Pour résoudre ce problème, on a choisi d'introduire deux autres directions de tranches alternatives (voir figure 3.3), à utiliser lorsque la première donne une qualité trop dégénérée. On choisit la direction la plus adaptée à la position du point de vue (voir un exemple de basculement dans le cas A de la figure 3.4 à droite) [MEY98].

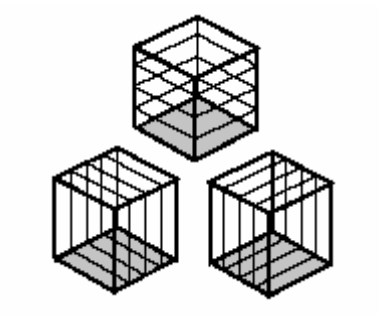


Figure 3.3 : Un texel avec ses 3 directions de faces.

Pour savoir quelle direction est la meilleur on pourrait effectuer trois produits scalaires entre l'axe de vue V et les trois directions du texel *N*, *T*, *B* (Normale, Tangente, Binormale), c'est-à-dire $\vec{V} \cdot \vec{N}$, $\vec{V} \cdot \vec{T}$ et $\vec{V} \cdot \vec{B}$ et choisir la direction dont le produit scalaire est le plus grand en valeur absolue. Mais le critère de qualité que nous le verrons au paragraphe 4.2 définit les choses de façon plus fine puisqu'il tient compte du nombre de tranches qui n'est pas forcement identique dans les trois directions. Ce critère nous indique le nombre minimal de tranches qu'il faut tracer pour une erreur donnée. On calcule ce nombre minimum pour les trois directions et on garde la direction qui comporte le moins de tranches nécessaires. On peut ainsi avoir une qualité constante pour un coût minimum.

Le signe du produit scalaire nous indique dans quel ordre afficher les faces puisqu'elles doivent être tracées de l'arrière vers l'avant (voir paragraphe 2.2).

4.1.2. Modification de l'algorithme de rendu.

Le fait d'ajouter deux directions de tranches entraîne une modification de l'algorithme de rendu exposé au paragraphe 3. On précalcule les polygones pour chaque texel comme pour l'algorithme précédent mais ceci se fait suivant les trois directions des tranches. Ceci ne pose pas de difficulté majeure. Il faut simplement calculer les coordonnées textures *U*, *V* de chaque point des polygones. Ces coordonnées textures se calculent en fonction des coordonnées textures de la surface sous-jacente [MEY98].

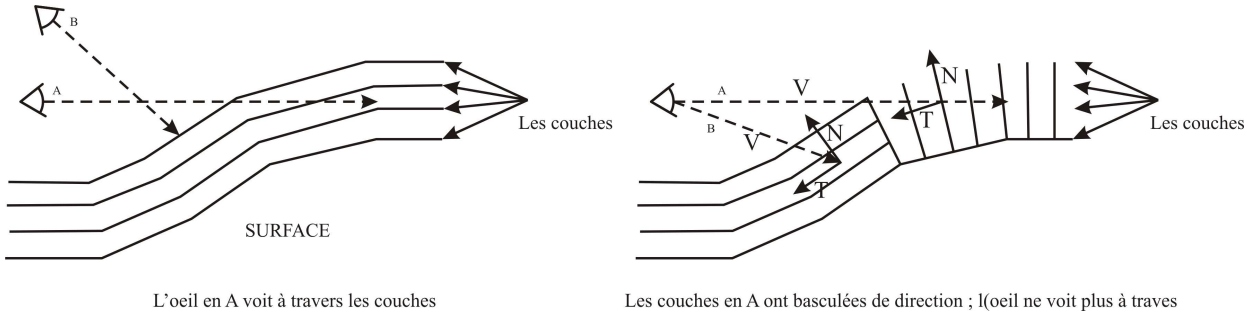


Figure 3.4 : Exemple en 2 dimensions du basculement de direction des tranches.

4.2. Résolution variable.

Ce paragraphe est consacré à la réduction du nombre de couches affichées tout en gardant une qualité visuelle constante. On présente donc une solution multi résolution des tranches du volume de référence, ainsi que deux critères de qualité qui permettront de choisir quelle résolution est la plus adaptée pour chaque texel.

4.2.1. Critère de qualité faisant intervenir l'angle de vue.

Lorsque le texel est vu parfaitement dans l'axe, l'effet de relief est moins important. Donc on pourrait n'afficher qu'une seule face texturée qui serait le résultat précalculé de "l'empilement" (sorte de z-buffer) de toutes les faces texturées [MEY98]. De même, quand le point de vue s'écarte à peine de cet axe quelques tranches suffisent à rendre compte de la parallaxe. L'idée est donc d'afficher un nombre de couches variable qui dépend de l'angle de vue, et plus précisément de la distance *d* (voir figure 3.5) qui correspond à la profondeur visible à l'intérieur de l'objet (qu'on ne devrait idéalement pas voir, mais qui apparaît parce qu'on peut voir un peu entre les tranches). On ne veut pas que l'intérieur soit visible trop profondément, car cela peut conduire à des images dégradées. Pour cela on souhaite que la distance *d* soit majorée par un maximum d_0.

Cette distance *d* dépend directement de l'angle de vue *a* et de la hauteur *h* qui est l'espace entre deux couches. Ce qui nous intéresse est de trouver le nombre *n* minimal.

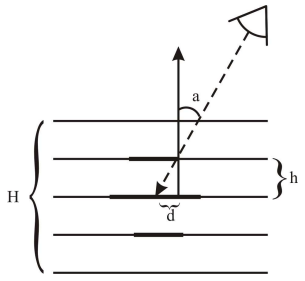


Figure 3.5 : Le premier critère.

On a :

$$\tan(a) = \frac{d}{h}$$

$h = \frac{H}{n}$ où : *H* est la hauteur totale du texel et *n* est le nombre de couches du texel.

D'où :

$$d = \frac{H \times \tan(a)}{n}$$

Notre critère de qualité *c1* est $d<d_0$, ce qui donne : $\frac{H \times \tan(a)}{n} < d_0$

On en déduit $\frac{H \times \tan(a)}{d_0} < n$

Ce qui donne le nombre minimum de tranches qu'il faut utiliser tout en gardant une qualité apparente constante.

On obtient *tan(a)* avec le produit scalaire entre le vecteur de vue V et la normale N :

$$\tan(a) = \frac{\sin(a)}{V.N} = \frac{\sqrt{1-\cos^2(a)}}{V.N} = \frac{\sqrt{1-(V.N)^2}}{V.N}$$

Pour ne pas avoir une erreur *d* supérieure à d_0, on a donc un nombre minimum de couches à respecter. Bien sûr on prendra ce nombre le plus proche possible du minimum de manière à gagner en nombre de faces.

Ce critère nous permet de contrôler ce que l'on voit de l'intérieur d'un objet. Lorsque l'objet est vue dans l'axe le minimum de couches sera très petit, alors que quand l'objet est vu selon un angle loin de l'axe ce minimum augmentera jusqu'à ce qu'une autre direction de couche soit plus efficace et donc on passera alors à l'autre direction [MEY98].

La plus part du temps on gagnera en nombre de faces affichées, mais il peut arriver que le nombre de couche que contient notre texel soit inférieur au nombre minimum de couches que l'on doit afficher pour respecter le critère. Dans ce cas on affiche toutes les faces existantes mais le critère de qualité n'est pas respecté.

4.2.2. Critère de qualité faisant intervenir la distance.

Avec ce critère *c2*, on aimerait que l'écartement apparent (c'est-à-dire l'écartement apparaissant à l'écran en nombre de pixels) entre deux couches d'un texel soit borné. C'est-à-dire que l'on veut que la distance h projetée à l'écran soit bornée par d1 (en pixels). La distance h projetée à l'écran est l (voir figure 3.6).

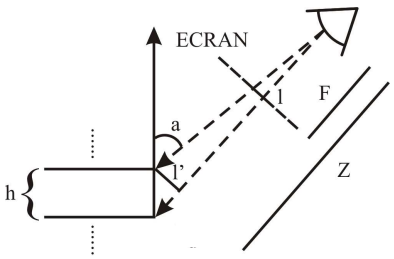


Figure 3.6 : Le deuxième critère.

On a $l = \frac{F \times l'}{z}$ à cause de la projection perspective et $l' = h \times \sin(a)$. On obtient : $l = \frac{F \times H \times \sin(a)}{n \times z}$.

On veut que $l < d_1$ ce qui donne : $\frac{F \times H \times \sin(a)}{n \times d_1} < n$.

Ce critère tient compte de la perspective, donc un texel se trouvant loin du point de vue aura un nombre minimum de couches très faible. Ce critère tient compte aussi de l'orientation du texel par rapport à l'axe de vue. Le nombre minimum de tranches est donc obtenu en tenant compte de l'éloignement et de l'orientation. C'est avec ce critère que l'on obtient les meilleures performances (en nombre d'images par seconde) mais d'un point de vue strictement qualité visuelle il faut utiliser une combinaison des deux critères.

Ce critère vient compléter celui vu au paragraphe précédent (4.2.1). Le nombre minimal de couches à afficher quand on utilise les deux critères est le maximum des deux valeurs trouvées (c.à.d. max(*c1*,*c2*)). Ce deuxième critère va prendre le dessus sur le premier (c'est-à-dire que min(*c2*)>min(*c1*)) quand l'objet sera proche du point de vue et sera orienté avec un angle éloigné de la normale. Dans ce cas là, le deuxième critère indiquera que l'on a tendance à voir l'écartement entre les couches (donc à trahir notre modèle à base de couches) et que donc il faut en afficher plus pour limiter la perception cette écartement [MEY98].

Le premier critère agira quand l'objet est à une distance moyenne ou lointaine du point de vue. En fonction de son orientation il donnera un nombre minimal de couches à avoir pour limiter la perception de l'intérieur de l'objet.

4.3. Pyramide de couches.

Le fait d'afficher moins de couches est un bon moyen de gagner en performance, mais si on supprime simplement des couches on perd de l'information, des détails fins peuvent disparaître. On présente donc ici une solution à ce problème qui consiste à précalculer des images qui sont la combinaison des deux images à remplacer. La structure de donnée résultante se présente sous la forme d'une pyramide d'images.

On va donc construire une pyramide d'images de la manière suivante : à partir des $N=2^k$ images on en construit $\frac{N}{2} = 2^{k-1}$ en les combinant deux à deux. Et ainsi de suite jusqu'à n'obtenir qu'une dernière image. Un niveau i de la pyramide va donc contenir 2^i images. On a choisit de diviser le nombre d'images par deux entre deux niveaux de la pyramide. C'est un choix qui semble raisonnable compte tenu du fait que la taille de la mémoire texture est limitée et que ces images doivent y tenir pour ne pas faire chuter les performances. Il faut que le nombre de tranches soit toujours une puissance de deux pour que ceci fonctionne. Ainsi avec $N=2^k$ images on précalcule 2^k-1 images, ce qui nous fait 2^{k+1}-1 images au total pour la pyramide [MEY98].

La combinaison d'une tranche 1 avec une tranche 2 donne une image où l'on voit la tranche 1 en avant plan et la tranche 2 en arrière plan (on peut voir la tranche 2 seulement aux endroits où la tranche 1 est transparente).

On voit bien [MEY98] que mettre la tranche 1 devant et la tranche 2 en derrière ne donne pas le même résultat que de mettre la tranche 2 devant. Nous sommes donc obligé de construire deux pyramides de tranches comme celle décrite ci-dessus. Une pyramide correspondant au texel vu de dessus et une pyramide correspondant au texel vu de dessous (voir figure 3.7).

Le critère du paragraphe 4.2 nous indique le nombre de tranches minimum qu'il faut tracer. La puissance de deux immédiatement supérieure à ce nombre nous indique quelle est le niveau de la pyramide à utiliser. Le produit scalaire entre le vecteur de vue et la normale au texel que l'on a déjà calculé pour le critère de qualité nous indique quelle pyramide utiliser.

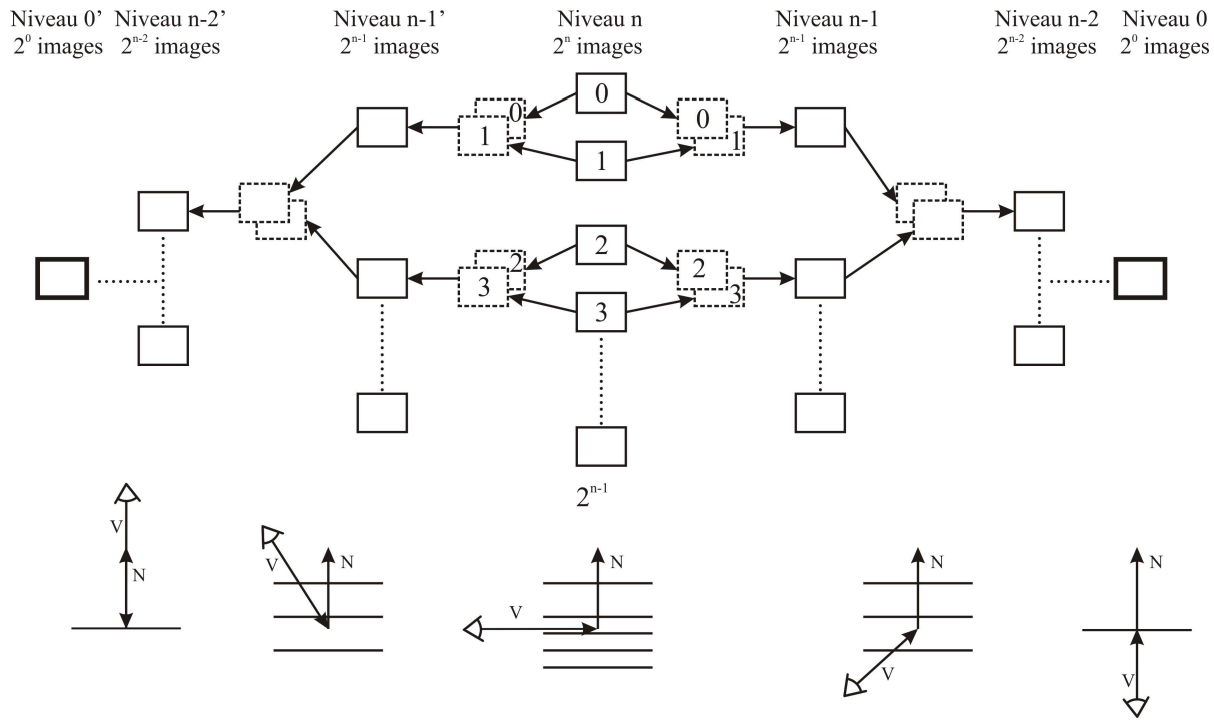


Figure 3.7 : En haut : Pyramide de tranches. En bas : En fonction de la position de l'oeil on utilise le niveau n de la pyramide pour afficher les couches.

4.4. Bonne utilisation du hardware graphique.

L'efficacité du modèle de texture volumique présenté ici est directement dépendante de la manière de construction de la machine sur laquelle il est implanté. La mémoire texture y joue un rôle primordial.

Le goulot d'étranglement de la mémoire texture.

Un texel de taille 64^3 représente 64 images de 16 Ko. Ces 64 images tiennent dans la mémoire texture, ce qui évite la dégradation des performances lors de l'affichage.

Lorsque l'on veut utiliser le critère de qualité il faut précalculer les deux pyramides d'images : On obtient donc 64 + 2 × (32 +16+ ... +1) = 127 images de 16 Ko, ce qui fait 2Mo. Ce qui tient en mémoire texture même sur une carte accélératrice 3D classique d'un PC. Si on choisit d'avoir trois directions de tranches (sans critère de qualité dans un premier temps) on a 3× 64 = 192 images de 16 Ko (=3Mo). Si on considère le critère de qualité et les trois directions de tranches, on a 3 × 64 + 6 × (32 +16 + ... +1) = 570 images de 16 Ko = 9 Mo.

Ces chiffres sont valables pour un volume de référence de taille 64^3 mais ils s'augmentent vite avec l'augmentation de la taille du texel (51 Mo pour un texel de 64×128×128 et 143 Mo pour un texel de 64 × 256 × 256).

5. Les texels sur la surface déformée.

Une fois que l'on a déformé la surface il faut propager la déformation aux texels se trouvant dessus. Le modèle utilisé pour déformer la surface n'influe pas sur la manière de calculer la déformation des texels.

Toutes les boîtes sur la surface se déforment : la base des boîtes est confondue avec le polygone de la surface correspondant, leurs vecteurs verticaux sont confondus avec les normales à la surface et la longueur de ces vecteurs qui correspond à l'épaisseur de la texture est constantes tout au long de l'animation (cette longueur est définie par l'utilisateur). Sur la figure 3.8 de droite on peut voir les boîtes plaquées sur la surface du drapeau (on peut aussi y voir les vecteurs normaux).

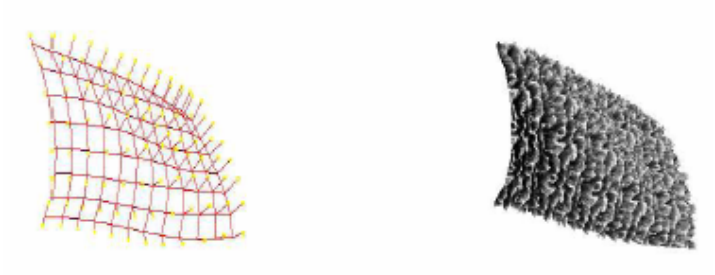


Figure 3.8 : Drapeau. Gauche : Les boîtes sur le drapeau. Droite : Le résultat avec la texture volumique.

6. Conclusion.

Nous avons introduit un nouveau modèle de texture volumique basé sur une représentation en tranches. Ce modèle utilise pleinement les fonctionnalités des cartes graphiques pour arriver à rendre, des scènes comportant des textures volumiques, plusieurs fois par seconde alors que les modèles précédents demandaient un temps de calcul très long, voir plusieurs minutes dans le modèle initial. Dans ce nouveau modèle le motif de texture est représenté par des tranches, chaque tranche étant en fait une image 2D. Ce qui permet d'utiliser le tracé de faces texturées qu'offre ***OpenGL*** pour le rendu qui est très rapide car il est implémenté en hardware.

Le motif de texture se plaque de manière répétitive sur toute la surface et se définit complètement et indépendamment du maillage de celle-ci ; ce qui est un gage de généralité. La qualité visuelle a été améliorée par l'utilisation de trois directions de tranches, et nous avons pu optimiser le nombre de polygones tracés et diminuer ainsi le temps d'affichage d'une scène pour atteindre un niveau d'interactivité très satisfaisant en introduisant un critère de qualité.

Résultas et perspectives

1. Introduction.

Dans cette partie nous venons présenter un ensemble de résultats permettant de valider le modèle utiliser pour le placage d'une texture volumique sur une surface tridimensionnelle. Les résultats sont obtenues en utilisant un Pentium IV 3.0 GHz équipé d'une mémoire 256 Mo DDR et une carte graphique NVidia FX-5200 avec une mémoire de 128 Mo. Nous allons présenter à la fin de ce chapitre un bilan récapitulatif du travail réalisé.

2. Génération de volume de référence.

Puisque nous avons présenté différentes méthodes de génération d'un volume de référence, nous présentons dans ce qui suit chaque méthode implantée à part.

2.1. Volume de référence à partir d'une texture de ***Perlin***.

Sur la figure 4.2 on peut voir un exemple de deux texels construits à partir des deux images 2D de *Perlin* qui se trouvent à la figure 4.1 (Une texture de bois à gauche et une autre de granite à droite). Chaque texel est représenté avec 64 couches, chacune à une résolution de 64×64 pixels. Un volume de référence avec cet résolution (64^3) ne tient à la mémoire texture que 1 Mo (64 couches de 16 Ko pour chacune). Un volume de référence de la même résolution tient dans la mémoire un taille en mémoire plus de 12 Mo. Donc, un volume de référence à base de couches d'images permet un grain appréciable en espace mémoire.

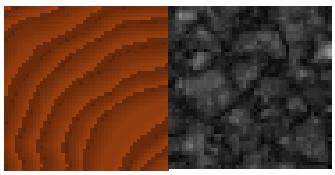


Figure 4.1 : Deux textures de *Perlin*.

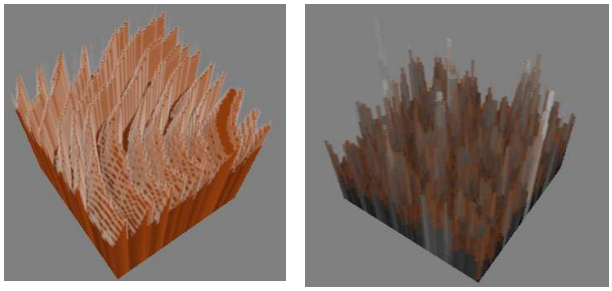


Figure 4.2 : Deux volumes de référence à partir des deux textures de *Perlin*.

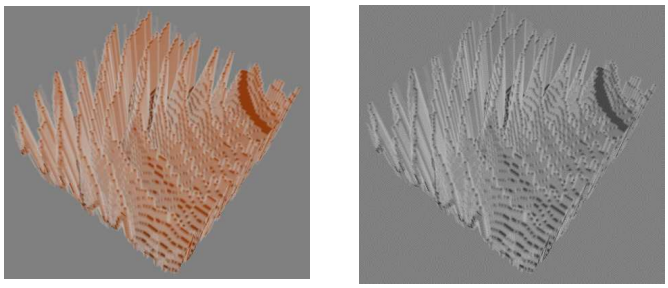


Figure 4.3 : Deux volumes de référence à partir d'une texture de bois.

La figure 4.3 représente le même volume de référence, mais cette fois, il est générer en utilisant la méthode expliquée dans le paragraphe 5.1.2-b du chapitre 2 (on complète chaque colonne en dessous de lui jusqu'au voisin le plus bas); à gauche il est présenté en couleur et présenté en niveaux de gris à droite.

La figure 4.4 représente un volume de référence déformé. La déformation de volume de référence n'influe pas sur le temps de rendu en utilisant cette présentation à base de couches. Par contre, pour un volume de référence de *Neyret*, leur déformation augmente le temps de rendu à cause de l'utilisation d'un algorithme itératif pour suivre un rayon déformé dans le volume de référence. Par exemple, une implantation de cet algorithme donne un temps de rendu de 5 à 20 minutes pour un volume de référence de résolution 128^3, dont la plupart de ce temps étant consacré au calcul d'intersection entre les rayons et les patches de la géométrie et les parois des texels [NEY96].

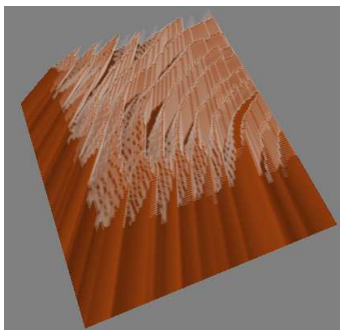


Figure 4.4 : Un volume de référence déformé.

2.2. Volume de référence à partir d'une scène reconstruite

Nous avons utilisé deux méthodes de reconstructions afin de générer un volume de référence à base de couches ; reconstruction en utilisant des images de profondeur et celle décrite par Seitz. Un exemple des images de référence est présenté dans la figure 4.5.

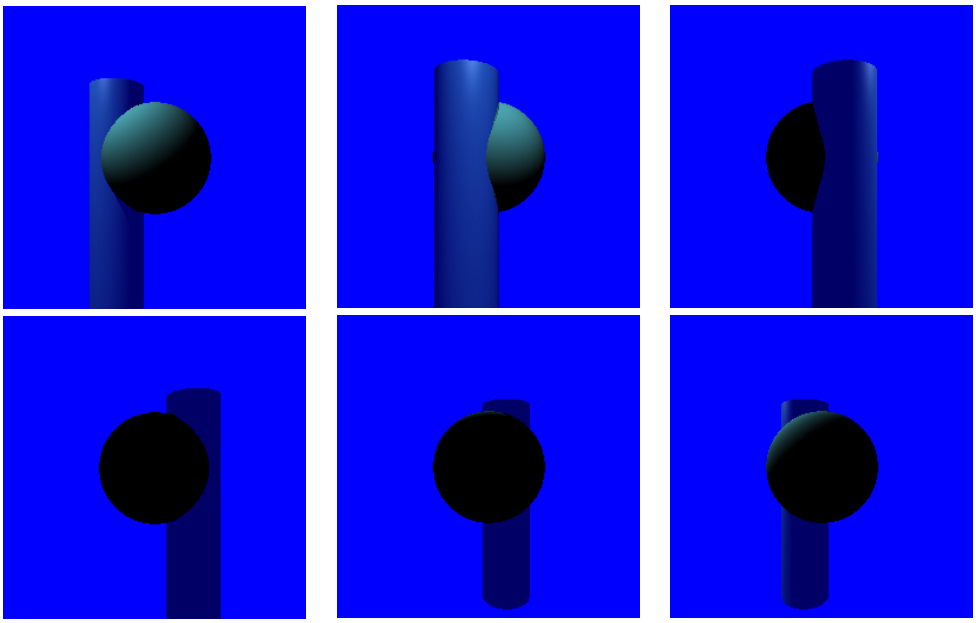


Figure 4.5 : Six vues d'une scène construites en utilisant le lancer de rayons.

La première permet une reconstruction exacte de la scène en utilisant l'information de profondeur accompagnant chaque pixel d'une image source (une coloration exacte des voxels). L'image d'en haut à gauche de la figure 4.6 est régénérée en utilisant six images de profondeur de résolution de 256×256 pixels. Pour ce cas, il y a **8597** voxels coloré d'un total de **262144** voxels (64^3) dans un temps de calcul de **3.52 secondes**. L'image de droite en haut est générée en utilisant les mêmes images et nous avons **7105** voxels colorés en utilisant une grille de voxels de la même résolution (64^3). La coloration prend un temps de **28.36** secondes, ce qui représente à peu pré 8 fois le temps du premier cas. La première méthode de reconstruction prend l'avantage qu'elle permet une coloration exacte par rapport à la deuxième, mais l'information de profondeur n'est pas toujours disponible, surtout pour le cas où nous voulons utiliser des images réelles. Par contre, la deuxième méthode permet d'avoir des résultats acceptables, même s'il y a des fautes de coloration de quelques voxels à cause du problème de visibilité.

Pour cette dernière méthode, il suffit d'avoir les paramètres intrinsèques et extrinsèques de la caméra de vision - des informations qu'il est facile à les avoir pour des images de synthèse. Pour des images réelles acquissent par des caméra numériques, il faut effectuer une opération de calibrage pour avoir ces informations (Annexe B).

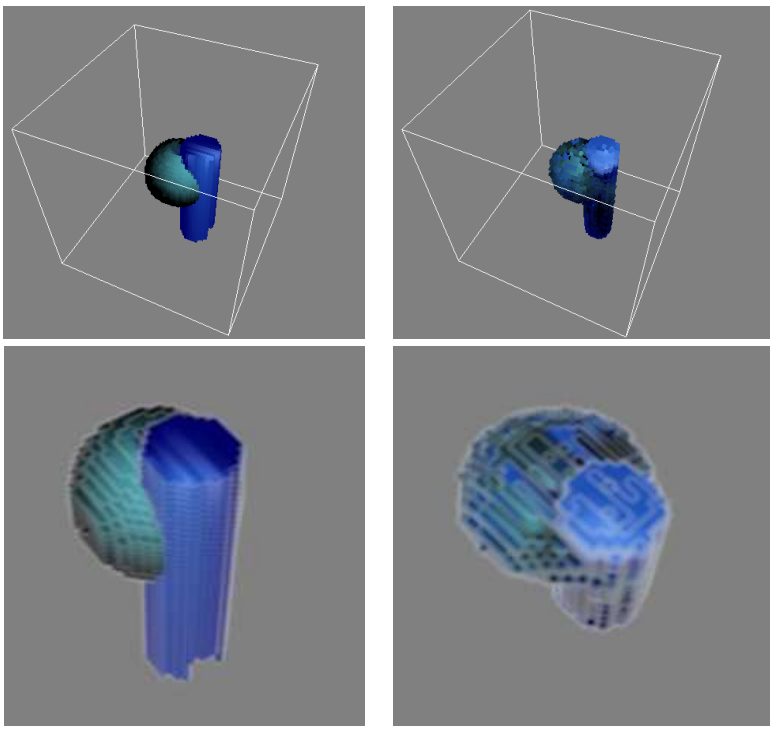


Figure 4.6 : Deux volumes de référence générés en utilisant la méthode de coloration de voxels à partir des images de profondeur (à gauche) et en utilisant la méthode de *Seitz* (à droite).

La figure 4.7 à gauche représente un objet C.S.G construit à partir de 12 images de référence, et le volume de référence correspondant en couches (au milieu avec une résolution de 64^3 et à droite avec une résolution de 128^3).

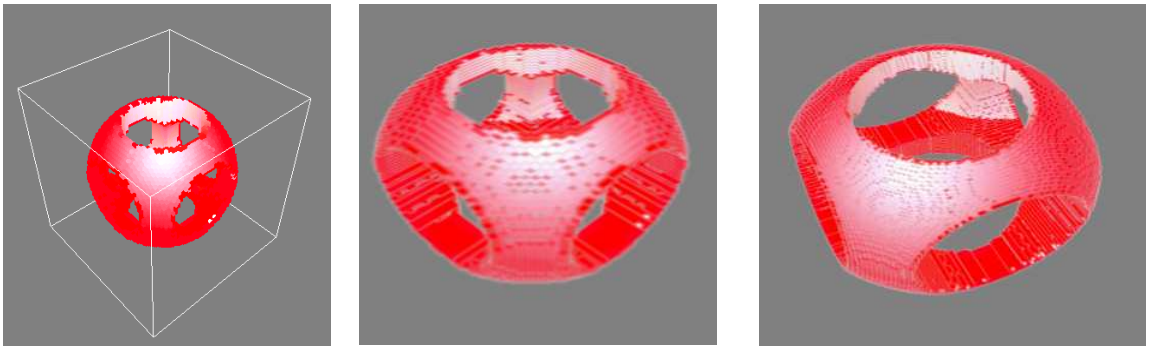


Figure 4.7 : Un volume de référence (Objet CSG) généré en utilisant la coloration de voxels.

3. Habillage d'une surface

Une scène comme celle présentée à droite sur la figure 4.8 est composée de **100 texels** de bois (de taille 64×64×64) posés sur la surface présentée à gauche s'affiche à **15.88 F.P.S (images/seconde)**, dont le temps de rendu est **62.99 millisecondes**.

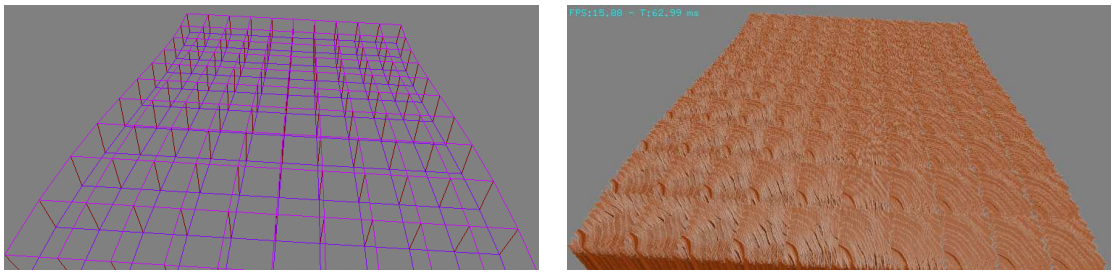


Figure 4.8 : Une surface composée de 10×10 patches et l'habillage de cette surface par une texture volumique de bois.

La figure 4.9 de gauche présente le problème posé si nous utilisant seulement des couches superposées dans la direction des normales aux facettes. En effet, si l'angle de vision, entre le vecteur de vision $\vec{V}$ et la normale à la couche, est très grand (c'est-à-dire que la valeur absolue du produit scalaire entre ces deux vecteurs tend vers zéro) on vois à travers ces couches. Par contre, la figure 4.9 de bas démontre que l'utilisation des couches dans les trois directions permet de résoudre ce problème.

L'utilisation de trois directions de couches étend trois fois l'espace mémoire utilisé pour stocker le volume de référence avec ses trois directions de couches, et par conséquent l'espace occupé dans la mémoire texture s'augmente. Un volume de référence de résolution de 64×64×64 occupe **3 Mo** au lieu de **1 Mo**, et un autre de résolution de 128×128×128 occupe **24 Mo** au lieu de **8 Mo**.

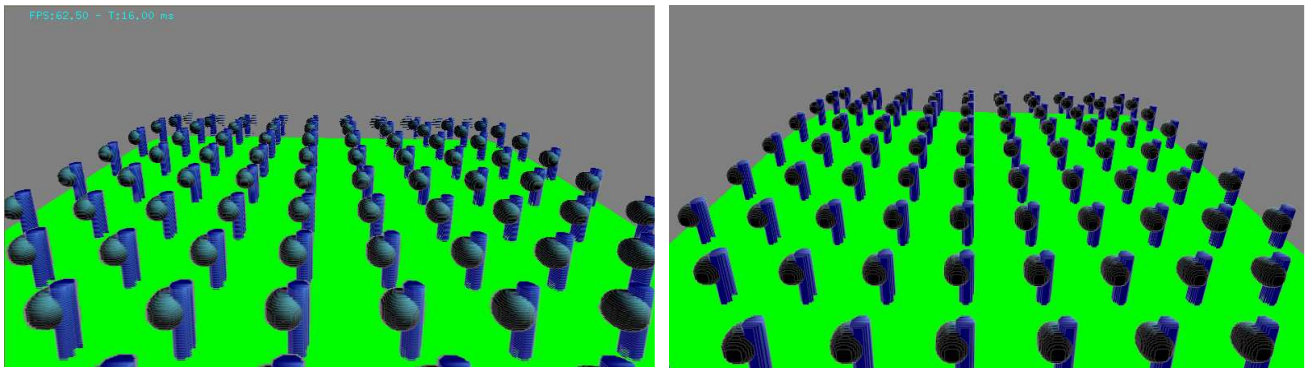


Figure 4.9 : Problème avec une seule direction des couches (à gauche) et la résolution du problème en utilisant trois directions de couches (à droite).

La figure 4.10 présente un ensemble de scènes construites en utilisant différents types de volumes de références. Toutes ces scènes comportes 100 texels de 64 couches (64×64 pixels) pour chacune. Nous remarquons que le temps de rendu de ces scènes est moins d'une seconde (**21.28**, **47** et **31.25 millisecondes** pour les figure de haut à gauche, à droite et en bas respectivement). Pour ces scènes, la fréquence d'affichages est **46.99**, **21.28** et **32.25 F.P.S**, ce qui nous ramène à dire que notre rendu est effectué en temps réel.

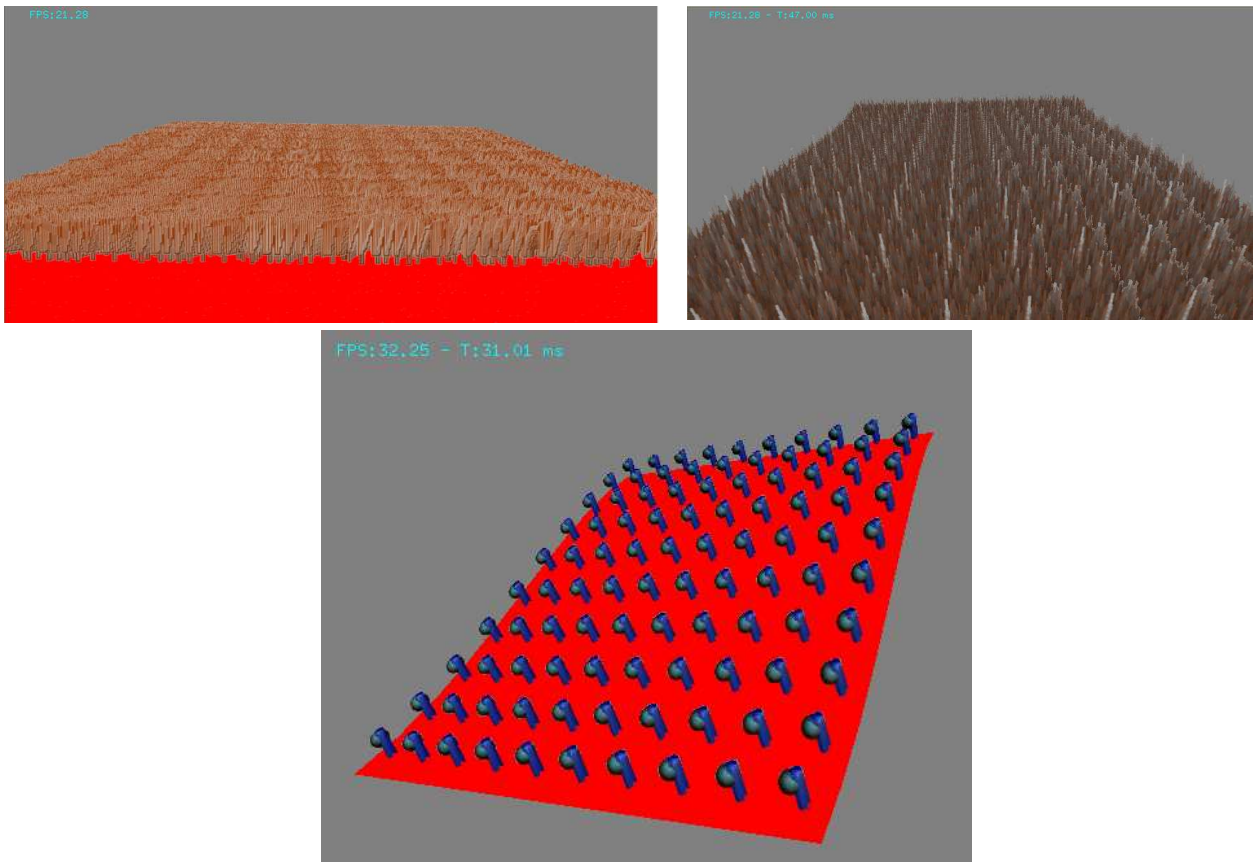


Figure 4.10 : Plusieurs scènes rendues en utilisant différents modèles de volumes de référence.

La figure 4.11 illustre le processus suivi durant l'opération de génération des textures volumiques à bases de couches d'images :

- La première étape consiste à générer le volume de référence ; pour le cas de l'élément de référence construit à partir d'une scène reconstruites en utilisant des images de référence, nous générons d'abord l'ensemble des voxels colorés en utilisant ces images (figure 4.11 en haut à gauche), et nous passant par la suite à la représentation à base de couches (figure 4.11 en haut à droite).
- La deuxième étape correspond à la génération de la surface à habiller ; c'est une ensemble de boîtes construites à partir des patches quadriques (les patches peuvent être aussi des triangles ce qui nous ramène à afficher chaque couche comme deux triangle contigus).

Ces deux étapes sont indépendantes, donc nous pouvons construire la surface avant de générer l'élément de référence.

- La dernière étape correspond au rendu de notre texture volumique ; un rendu qui s'effectue en affichant les couches dans les boîtes de la surface prenant encharge l'ordre de profondeur des couches.

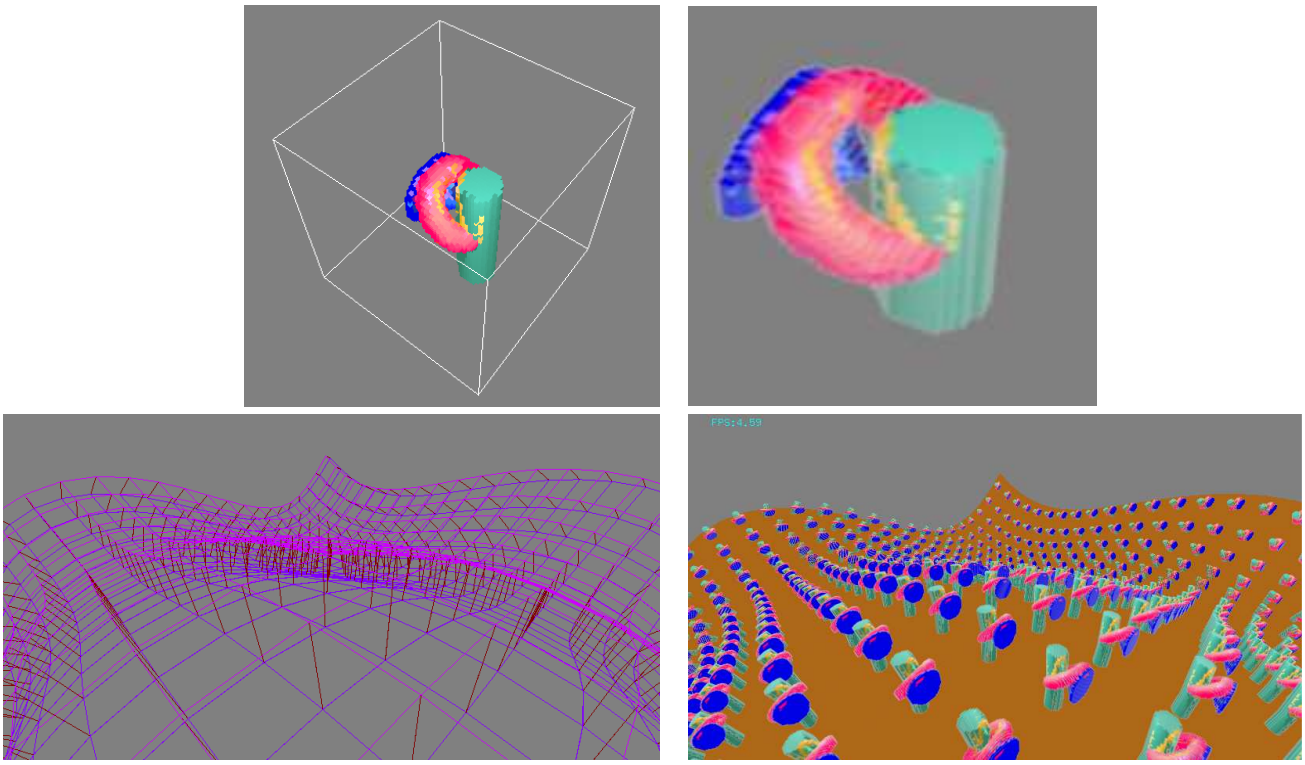


Figure 4.11 : Les étapes de processus de rendu à base de couches d'images.

Pour la surface de la figure 4.11 de bas à droite (comportant 400 boîtes), nous observons que la fréquence d'affichage des images est **4.99 F.P.S**, correspond à **200 millisecondes**, est petite. La fréquence d'affichage est liée au nombre de patches de la surface, la position de l'observateur par rapport à la surface (très pré, très ou un peu loin) et l'information contenue dans chaque volume de référence (c'est-à-dire l'occupation des primitives contenues dans le volume de référence, ou d'une façon très claire, le nombre de pixels opaques et transparents dans les couches constituant notre échantillon).

La figure 4.12 présente un autre ensemble de scènes rendues en utilisant les textures volumiques à base de couches d'images. L'image de gauche est obtenue en plaquant une texture d'herbe sur la surface. Celle de droite est obtenue en utilisant une autre texture d'herbe plaquée sur une surface de 400 patches. L'image en bas représente une colline habillée comportant 400 arbres sur une surface d'herbe. Le temps de rendu de cette scène est **203** millisecondes avec une fréquence de **4.93 F.P.S**.

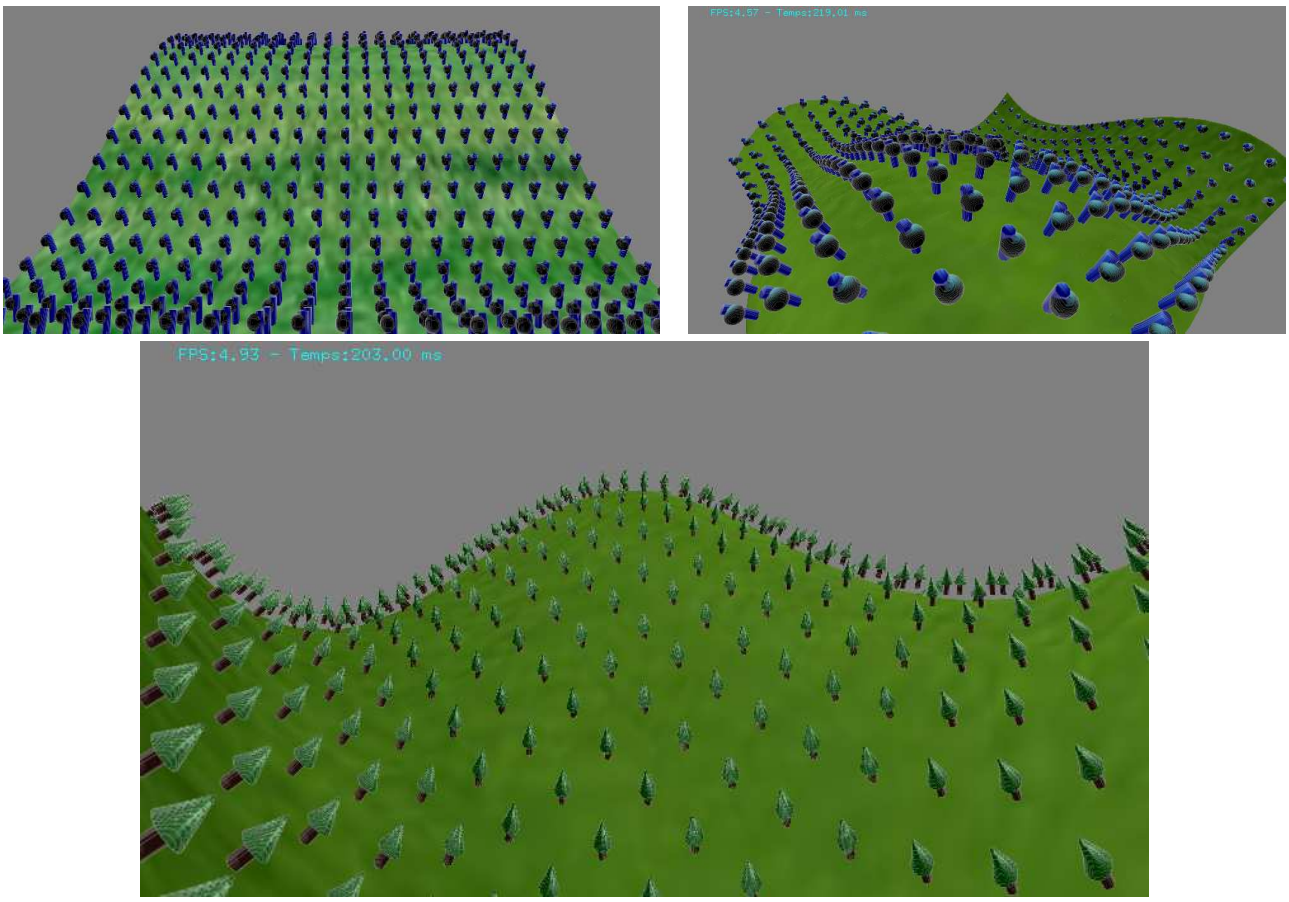
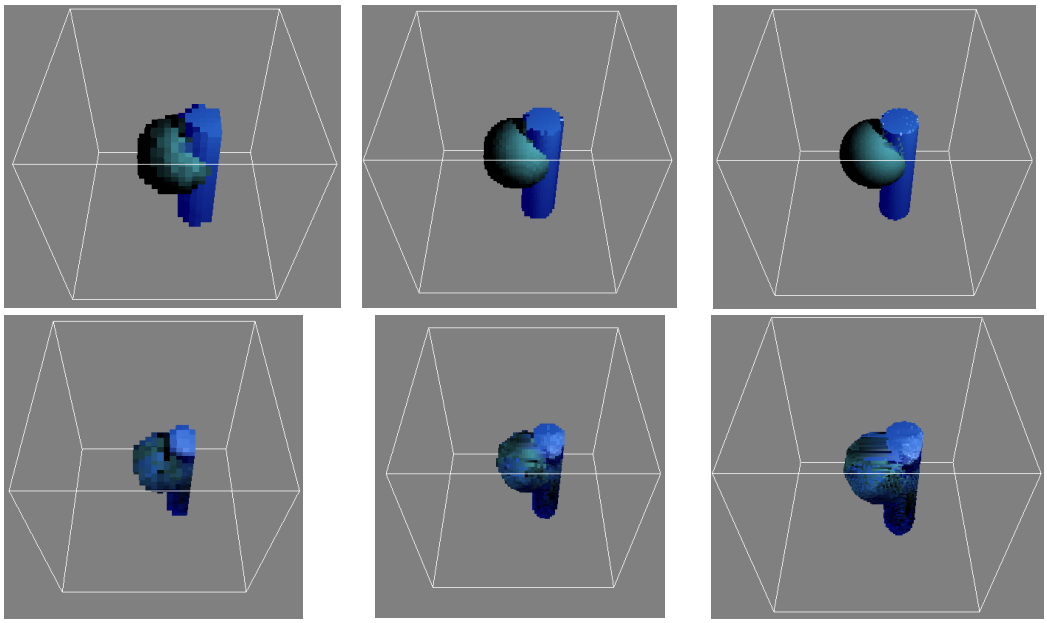


Figure 4.12 : Trois scènes créées en utilisant la technique de rendu à base de couches d'images.

4. Bilan récapitulatif

4.1. La reconstruction des objets



Résolution de 32^3 Résolution de 64^3 Résolution de 128^3

Figure 4.13 : Une reconstruction à partir des images de profondeur (en haut) et une autre en utilisant l'algorithme de *Seitz* (en bas).

L'utilisation des images de profondeur permit une reconstruction plus précise et dans des temps plus petit que dans le cas où nous utilisons l'algorithme de *Seitz*. La table ci-dessous rassemble des comparaisons en terme de nombre de voxels colorés et le temps nécessaire pour reconstruire les objets présents dans la figure 4.13.

	Coloration en utilisant des images de profondeur		**Coloration en utilisant la méthode de *Seitz***	
Résolution de la grille 3D	**Nb. Voxels colorés**	**Temps écoulé(s)**	**Nb. Voxels colorés**	**Temps écoulé(s)**
32^3	1152	1.01	687	12.55
64^3	7376	1.28	7105	27.94
128^3	47977	1.34	56350	80.89

En effet, une reconstruction à partir des images de profondeur est très avantageuse par rapport à celle de *Seitz* en se qui concerne la côté précision de l'opération de reconstruction. Malheureusement, l'information du profondeur n'est pas toujours disponible se qui rend la méthode de *Seitz* la plus avantageuse de ce côté.

4.2. Optimisations apportées au rendu des couches d'images

Le temps de rendu d'une scène composée de **1600 patches** est de **422 millisecondes**, avec une fréquence d'affichage de **2.37 F.P.S**. Le volume de référence utilisé a une résolution de 64^3 et occupe une taille de 3 Mo où nous utilisons des couches images dans les trois directions. Malgré que le temps de rendu apparaît acceptable, des optimisations apportées au algorithme de rendu permet d'avoir un gain appréciable en temps de calcul avec une augmentation dans l'espace mémoire nécessaire pour charger le volume de référence.

Le volume de référence dans sa nouvelle représentation (pyramide de couches) occupe un espace de **14.25 Mo** dont les niveaux du pyramide comporte successivement 4, 8, 16, 32, 64, 62, 16, 8 et 4 couches pour chacune des trois directions. La table ci-dessous rassemble les temps de rendu de la même scène décrite précédemment (1600 boîtes), mais cette fois en utilisant le premier critère c1 (qui fait intervenir l'angle de vue) du paragraphe 4.2.1 au chapitre précédent.

d0	**Nbr. total des couches affichées**	**Fréquence d'affichage (F.P.S)**	**Temps écoulé (ms)**	**Nbr. minimal des couches affichée par boîte**	**Nbr. maximal des couches affichée par boîte**
0.5	7728	21.28	47	4	8
0.25	10384	10.64	94	4	16
0.125	15696	8.00	125	4	32
0.0625	26320	4.59	218	4	64

A partir de cette table, nous remarquons que l'augmentation de la taille de la surface visible d0 permet de diminuer le nombre de couches à affichées et diminuer par conséquent le temps nécessaire pour l'affichage de la scène.

En faisant intervenir le deuxième critère c2 du paragraphe 4.2.2 au chapitre précédent (la distance de la caméra par rapport à chaque boîte), nous pouvons diminuer, aussi, le temps de calcul. La table ci-dessous présente les temps de calcul en faisant varier d1 le nombre de pixels représentant l'écartement visible entre les couches. L'augmentation de d1 permet de diminuer les temps de calcul, c'est-à-dire que l'augmentation en écartement entre les couches permet de diminuer le nombre de couches affichées, et diminuer par conséquent le temps de calcul.

d1 (pixels)	Nbr. total des couches affichées	Fréquence d'affichage (F.P.S)	Temps écoulé (ms)	Nbr. minimal des couches affichée par boîte	Nbr. maximal des couches affichée par boîte
1	14952	9.17	109	4	16
2	7728	12.82	78	4	4
3	6400	12.82	78	4	4

La table ci-dessous, présente les temps de calcul en faisant fixer d1 à 3, pour une surface de 400 boîtes, et en faisant varier la distance entre la caméra et la surface à habiller. Pour des distances proches le nombre de couches pour chacune des boîtes s'augmente et par conséquent le temps de calcul s'augmente. Le cas de la dernière ligne est expliqué par le fait que les couches ne sont pas tous visibles sur la surface d'affichage et par conséquent le temps d'affichage se diminue.

Distance entre caméra et centre de la scène	Nbr. total des couches affichées	Fréquence d'affichage (F.P.S)	Temps écoulé (ms)	Nbr. minimal des couches affichée par boîte	Nbr. maximal des couches affichée par boîte
36.36	2132	66.67	15	4	8
25.36	3436	62.47	16.01	4	16
13.36	5264	32.25	31.01	4	32
3.36	9048	62.53	15.99	8	64

L'utilisation des deux critères c1 et c2 ensemble nous obtenons les meilleurs résultats en se qui concerne le rapport qualité temps (pour une surface de 1600 boîtes dans le cas des statistiques présentes dans la table ci-dessous), par le fait que le deuxième critère fait intervenir la notion de distance par rapport aux boîtes pour la spécification du nombre de couches à afficher. A une distance proche il faut afficher un grand nombre de couches où intervient le premier critéire.

d0	d1	Nbr. total des couches affichées	Fréquence d'affichage (F.P.S)	Temps écoulé (ms)	Nbr. minimal des couches affichée par boîte	Nbr. maximal des couches affichée par boîte
0.5	1	14776	12.66	79	4	16
0.5	2	7736	15.87	63	4	8
0.5	3	7740	16.13	62	4	8
0.25	1	15124	12.82	78	4	16
0.25	2	15124	15.87	63	4	16
0.25	3	15124	16.13	62	4	16
0.125	1	30196	7.14	140	4	32
0.125	2	30196	7.09	141	4	32
0.125	3	30196	6.37	157	4	32

5. Perspectives du travail réalisé

La méthode de rendu présentée dans ce travail est très efficace pour l'affichage des textures volumiques introduites par *F. Neyret*. L'efficacité réside dans le fait que temps de rendu de scènes complexes comportant plusieurs patches est raisonnable et n'excède pas 1 seconde pour les scènes générées. Généralement, pour une dizaine de patches la fréquence d'affichage varie, jusqu'à présent, entre 20 et 63 images par seconde, qui représente un taux d'affichage en temps réel. De plus, malgré l'utilisation des images de synthèse pour la coloration des voxels, l'efficacité de notre modèle correspond à la possibilité d'avoir des textures volumiques réalistes en utilisant la nouvelle méthode de génération d'un volume de référence en utilisant des images réelles.

Malgré tout ça, il reste quelques points qui ne sont pas atteintes et qui restent comme des perspectives pour ce travail :

- La génération des volumes de référence en utilisant la méthode de coloration de voxels sur des images réelles, après avoir calibrer la caméra en calculant pour chaque image leurs paramètres intrinsèques et extrinsèques.
- L'utilisation de quelques techniques de rendu et de modélisation (comme celle de coloration de voxels de *Seitz*) pour avoir une bibliothèques large de volumes de référence.
- La modélisation de différents types de surfaces utilisant des générateurs de maillages volumiques, en utilisant les maillages surfaciques générés et sauvegardés.

6. Conclusion.

Ce chapitre a été consacré à la présentation de quelques résultats permettant d'évaluer les performances de la nouvelle technique de modélisation d'une texture volumique à base de couches d'images et le processus de rendu de cette nouvelle représentation. Nous constatons que les temps de calcul en faisant intervenir les différentes optimisations sont acceptables, sachant que les scènes sont complexes (comportant des déformations avec un grand nombre de patches).

Néanmoins, il reste quelques améliorations à introduire dans notre système pour avoir des résultats plus réalistes en utilisant des volumes de références réels et en introduisant un mécanisme d'éclairage de la scène générée.

Conclusion générale

De nos jours, le domaine de synthèse d'images réalistes est très riche de techniques traitant la complexité géométrique et temporelle. Les techniques de rendu à base d'images présentent une famille de ces dernières, où l'image représente un outil très fort pour la création des images réalistes dans des temps acceptables.

Le rendu des scènes complexes répétitives en utilisant des textures volumiques a généré de l'intérêt. Le modèle d'origine dédié à la modélisation de fourrure s'est transformé peu à peu en une représentation avantageuse pour générer des scènes complexes répétitives dans des temps généralement acceptables. Cependant, le besoin de générer, dans des temps réel, ce type de scènes pour quelques domaines tel que la réalité virtuelle, a conduit de chercher une nouvelle représentation pour les textures volumiques.

En faisant lier, les textures volumiques aux techniques de rendu à base d'images, *Meyer* a introduit cette nouvelle représentation. En effet, au lieu de considérer un volume de référence comme un ensemble de voxels constituant un arbre octal et d'utiliser le lancer de rayons pour rendre cette structure, nous avons utilisé une représentation à base de couches d'images. Un volume de référence est, donc, un ensemble de polygones texturés et superposés, ce qui permet l'utilisation du matériel graphique pour les affichés et réaliser ainsi un gain appréciable en temps de calcul.

Partant de ces principes, nous avons conçue un système de modélisation et de rendu de textures volumiques à base de couches d'images, tout en cherchant l'efficacité de processus de rendu surtout en ce qui concerne le temps de rendu.

Notre travail a été divisé en deux parties principales :

- La première consacrée à une étude et une classification des méthodes de rendu à base d'images, où on présente celle du rendu à base de couches d'images.
- La deuxième concerne la présentation de la technique de modélisation et de rendu d'une texture volumique à base de couches d'images. Dans l'étape de modélisation nous avons introduit une technique de génération d'un volume de référence en se basant sur la méthode de coloration de voxels qui est dédiée principalement à la reconstruction des scènes tridimensionnelles à partir des images.

Notre travail a été achevé par un chapitre présentant quelques résultats obtenus avec notre système de modélisation et de rendu, accompagnés d'un bilan récapitulatif prouvant l'utilité et la puissance de cette technique par la génération des résultats en temps réel.

A travers ce travail, nous pouvant dire que nous avons atteint l'objectif principal de ce projet ; l'efficacité de rendu en terme de temps de calcul et l'espace mémoire utilisé. En comparant cette technique à celle utilisant le rendu classique par le lancer de rayons pour la génération des textures volumiques, nous pouvant dire que nous avons obtenir un gain appréciable soit en terme d'espace mémoire utilisé soit en terme de temps de calcul.

Au point de vue améliorations à apporter sur notre travail, nous pouvant citer les suivantes :

- Nous pouvons encore introduire d'autres techniques de génération des couches du volume de référence pour garantir la diversité des modèles de textures (avoir une base très riche de modèles d'objets volumiques pour les utiliser comme des volumes de référence).
- Une voie très importante serait l'utilisation des images réelles pour la génération des volumes de référence plus réalistes, en utilisant la technique de coloration de voxels. Cette étape sera réalisable si on arrive à implanter une technique de calibration de la caméra d'acquisition, pour déterminer les paramètres intrinsèques et extrinsèques de la caméra.
- L'étude de la possibilité d'introduire des paramètres d'éclairage à ce nouveau modèle pour remédier le problème de l'éclairage statique de la scène.

Malgré que l'éclairage est statique (calculé durant l'étape de construction de volume de référence), la représentation à bases de couches d'images, est considérée comme prometteuse du fait qu'elle permet de générer des images en temps réel.

Annexe A :
Traitement de la complexité en synthèse d'images

1. Introduction.

La construction d'images de synthèse nécessite de grandes puissances de calcul surtout si l'on souhaite réaliser des applications interactives en temps réel. Malgré la montée en puissance des PC, la demande est toujours plus grande que les possibilités offertes. La recherche conduit donc à concevoir des méthodes intelligentes de construction de scènes dynamiques où l'on joue sur les algorithmes de visibilité ainsi que sur les méthodes de modélisation des objets (fractals, L-systèmes, ... etc.).

Pour remédier aux problèmes du rendu polygonal, plusieurs techniques et approches ont été présentées. Ces dernières sont classées en deux grandes familles [DEB04] : Les approches classiques et les approches émergentes.

2. Les approches classiques.

2.1. Accélération par le hardware graphique.

Jusqu'à nos jours une des premières solutions adoptées, afin d'éviter les problèmes de la représentation géométrique, est l'utilisation de matériel graphique, tel que les stations graphiques[1], en bénéficiant de leurs performance de calcul. La plupart des stations graphiques utilisent un lissage de *Gouraud* et le tampon de profondeur pour le rendu. Ces stations sont très chères (coûteuses) ce qui a limité leur usage personnel. De plus, la spécificité de certaines fonctionnalités, hypothéquant la pérennité et la portabilité des méthodes qui les utilisent[2].

2.2. Simplification polygonale.

Les techniques de simplification polygonale offrent une solution pour la totalité des modèles complexes comportant un très grand nombre de polygones. Ces méthodes peuvent notamment être utilisées quand l'objet est petit sur l'écran, s'il est éloigné, s'il se déplace, s'il n'est pas dans la direction principale du regard de l'utilisateur, tout en cherchant à réduire le coût de rendu sans une perte significative dans le contenu visuel de la scène [LUE01]. La majorité d'algorithmes de simplification nécessite que les maillages des objets soient bien formés et vérifier la propriété de 2D-mainfold ; toute arête soit l'incidence de deux faces (voire les figures A.1 et A.2).

[1] Tel que les SGI de Silicon Graphics ou Alto de Xerox.
[2] La rapidité de l'évolution du matériel peut rendre trivial un problème difficile quelques mois plus tôt, ou au contraire une fonctionnalité puissante mais peu utilisée par l'industrie peut disparaître

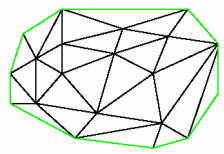


Figure A.1 : Une 2d-mainfold avec une frontière.

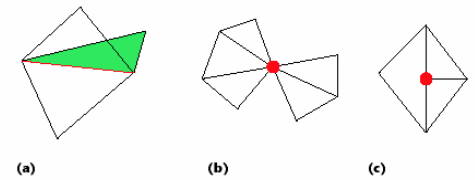


Figure A.2 : Exemples des mailles de non manifold.

Les algorithmes de simplification sont classés selon les différents critères adaptés. On a choisi de présenter une classification selon les mécanismes de déplacement ou « *enlèvement* » de polygone de base, où il y a trois grandes classes de simplification [KRU97, LUE01] :

- **La décimation :** les techniques de décimation enlèvent itérativement des sommets ou des faces de la maille, et re-triangulent le trou résultant après chaque pas. Le procédé de décimation s'arrête lorsqu'il ne peut plus retirer de géométrie et lorsque la représentation satisfait un degré d'approximation spécifié par l'utilisateur [LUE01, VIN98, SCH03]. Ces algorithmes sont relativement simples à coder et peuvent être très rapides.
- **L'échantillonnage :** cette méthode effectue un échantillonnage de la géométrie de l'objet, soit en choisissant arbitrairement un certain nombre de sommets à conserver, soit en englobant le modèle dans une grille tridimensionnelle et en échantillonnant chaque boîte de la grille. Ces approches sont les plus complexes et les plus difficiles à coder. Ces algorithmes travaillent d'habitude mieux sur des formes organiques douces sans coins pointus.
- **La subdivision adaptative :** elle commence avec un modèle de base très simple et le subdivise récursivement en ajoutant des détails à certains endroits du modèle à chaque étape. L'algorithme s'arrête lorsque le modèle approxime le modèle original à un degré spécifier par l'utilisateur [LUE01]. Cette méthode est assez peu utilisée car il n'est pas facile de construire la simplification initiale dans le cas général.

De nos jours, les algorithmes de décimation sont les plus utilisés dans le domaine de simplification, en dépit des inconvénients suivants :

- Le temps de calcul est prohibitif.
- Il n'existe pas d'algorithme général qui peut simplifier tous les objets.
- Le résultat de décimation n'est pas parfait (l'apparition des trous par exemple).

3. Les approches émergentes.

3.1. Détermination de visibilité.

On peut [LEB00] classer de différents façons les algorithmes de visibilité. Certains déterminent des surfaces visibles ou potentiellement visibles d'un point de vue précis ou ponctuel (*pinhole camera*), et par conséquent elles sont difficilement applicables pour déterminer de façon spécifique la visibilité entre éléments paires, comme c'est le cas dans un algorithme de radiosité.

Les différentes techniques de visibilité se divisent selon les catégories suivantes [LEB00] :

- **La visibilité approximative :** Plusieurs méthodes de visibilité se basent sur l'échantillonnage ; tel que la visibilité volumique et la carte d'ombre de *Williams*.
- **La visibilité exacte :** D'autres techniques retournent des résultats exacts mais à une précision fixée à l'avance par la taille de l'image désirée. Ces techniques comprennent entre autres : le tampon de profondeur, le balayage des lignes, l'algorithme du peintre par arbre BSP et les

algorithmes par subdivision d'aire. Malheureusement, leur complexité algorithmique devient rapidement un souci majeur pour des scènes très complexes.

- **La visibilité conservatrice :** un groupe d'algorithmes dits conservateurs tentent d'accélérer les différentes requêtes de visibilité. Ces algorithmes retournent un ensemble de surfaces visibles d'un point de vue ou d'une région. Pour être efficaces, il est essentiel que « toutes » les surfaces visibles soient contenues dans l'ensemble final. Parmi les techniques utilisées on peut citer [LEB00] : les portails, la pyramide de vue et celle utilisant l'occlusion provoquée par les surfaces rapprochées afin d'éliminer les surfaces cachées.

3.2. Niveaux de détails (***LOD***).

Après avoir éliminé tous les objets qui ne sont pas dans le champ de vision, il est intéressant d'utiliser une géométrie plus simple pour les objets ayant une moindre importance visuelle. On fait donc appel aux niveaux de détails (appelés *LOD* de l'anglais ***Level Of Detail***).

Cette technique essaye d'accélérer le rendu et d'augmenter l'interactivité en simplifiant la scène polygonale générée. Pour cela, on représente les objets éloignés avec un petit *LOD* et les objets voisins avec un plus haut *LOD* (Figure A.3). Le problème de l'utilisation des niveaux de détails se décompose en deux parties : La création et la gestion des *LOD*.

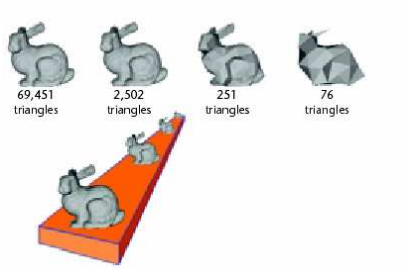


Figure A.3 : Niveaux de détails dégradés en fonction de la distance.

La création des *LOD* est basée sur la simplification polygonale. Il y a deux grandes classes de méthodes de création des *LOD* [KRU96] : La simplification orientée géométrie et la simplification orientée scène.

La première méthode comporte deux techniques : la simplification polygonale déjà vue et la simplification structurelle qui change la structure de représentation des objets. Par exemple, il est possible de remplacer un objet polygonal par une boite englobante texturée à l'aide d'une image produite à partir d'une version détaillée de l'objet (les imposteurs) [SCH03]. D'autres méthodes proposent de remplacer les objets par un polygone texturé à l'aide d'une image de l'objet original calculée avec un haut niveau de détails [KRU97].

La plupart des algorithmes orientés scène cherche à simplifier des régions de la scène plutôt que les objets eux-mêmes. Ainsi, ils décomposent récursivement la scène en zones 3D (en utilisant un octree ou une grille 3D), pour obtenir une description hiérarchique. Un parcours de la hiérarchie est effectué pour chaque image afin de déterminer le niveau de détails à utiliser. En plus, elles ne sont vraiment efficaces que sur des scènes très profondes, sans occlusion due par exemple à des murs [KRU97].

En 1993, *Funkhouser* et *Séquin* ont signalé que la gestion des différents *LOD* doit être faite de manière prédictive[1], plutôt que réactive[2]. Ces auteurs définissent un certain nombre de facteurs qui permettent de choisir un *LOD* parmi d'autres [KRU97] : La distance, l'incidence (angle de vision), la vitesse et la tâche réalisée (adapter la présentation à la tâche en cours) [KRU97].

Si deux images successives utilisent deux représentations différentes d'un objet, il est possible d'échanger immédiatement les représentations entre les deux images. L'action de transition entre les *LOD* ne peut être faite de manière automatique et nécessite une intervention humaine. Toutefois, certaines méthodes proposent d'utiliser un taux d'erreur ou la distance par rapport à l'original en faisant une correspondance avec la taille dans l'espace image. En effet, les temps de calcul restent prohibitifs pour une utilisation interactive.

3.3. Les algorithmes spécialisés.

3.3.1. Les cartes d'horizon.

Les cartes d'horizon encodent, comment de petites perturbations géométriques sur une surface portent des ombres sur la surface elle-même. Le but est de produire des cartes d'horizon consistant aux données acquises.

Pour construire la carte d'horizon d'un objet, *Rushmeier* [RUS01] a proposé une méthode qui consiste à acquérir huit images, une pour chacune des directions azimutales dans la carte d'horizon. Les huit images sont obtenues à partir des directions azimutales standard, à un angle de zénith d'environ 45°. Avec la carte des normales, les ombres propres sont identifiés dans chaque image en cherchant les pixels où le produit scalaire de la normale de la surface et la direction de la lumière est négatif. Les pixels restants sont dans les ombres portées.

Cette méthode exige un nombre restreint d'images, et n'exige aucune reconstruction de la surface balayée. Elle prend en charge les bosses relativement pointus, toutefois elle produit des ombres plausibles qui sont conformes aux ombres observées dans les images d'entrée. Cependant, cette technique reste coûteuse à utiliser car d'une part le rendu en plusieurs passes prend du temps, et d'autre part la carte de visibilité (d'horizon) occupe beaucoup d'espace mémoire [RUS01, POR04].

3.3.2. Approches phénoménologiques.

Une approche phénoménologique vise à reproduire directement la forme de l'objet et les phénomènes émergeants (plis, ondes, instabilités, formes d'équilibre) pour certaines familles.

a- Les Plantes.

La modélisation et l'animation des plantes représentent un secteur intéressant et stimulant pour l'animation par ordinateur. Les plantes semblent qu'elles exposent la complexité arbitraire en possédant une structure contrainte. Ils grandissent d'un seul point de source, développant une structure d'embranchement en quelque temps tandis que les branches s'allongent. Les plantes ont été modelées en employant les L-systèmes (Figure A.4).

[1] Basée sur la complexité de la scène.
[2] Basée sur le temps de calcul de l'image précédente.

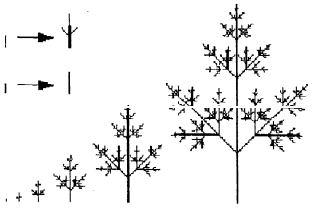


Figure A.4 : Arbre désigné par un L-système.

Figure A.5 : Exemple de déchirure.

S. Lefebvre a été intéressé essentiellement aux déchirures (écorce, pain) qui résultent d'un phénomène de croissance [LEF01]. Il s'agit d'un habillage visuel couplant la géométrie et la texture, c'est pourquoi une approche phénoménologique est tout à fait appropriée (Figure A.5).

b- Liquide.

Parmi les techniques de modélisation et d'animation des liquides on peut citer : L'animation interactive des vagues d'un océan et l'animation de la lave.

L'animation et l'affichage interactifs d'un océan illimité se fondent sur un modèle procédural de vagues. Ce modèle exprime les déplacements de points surfaciques de la vague active. Le modèle utilisé se base sur celui de bosse de *Gerstner* simulant des trochoïdes. Les trains de vagues sont produits d'une manière dont ils approchent un spectre de vague connu. Ce modèle est consacré aux eaux profondes, et ne couvre pas ainsi la réfraction des vagues proche du rivage ni la rupture des vagues [HIN01].

Concernant le rendu, *Hinsinger* et al. [HIN01] appliquent une carte d'environnement sur la surface de l'océan, reflétant ainsi le soleil et le ciel. Le rendu et l'affichage sont ainsi exécutés à l'aide de matériel graphique.

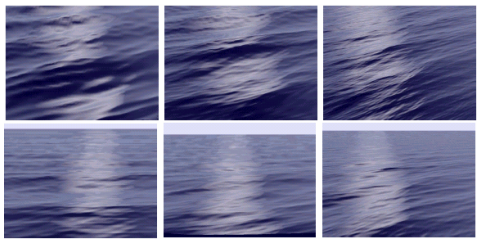


Figure A.6 : Différentes résolutions de mailles. A gauche : 50x50. Milieu 100x100; A Droite : 500x500;

Un des avantages de cette approche est sa flexibilité : l'utilisation des trochoïdes permet de modeler un éventail de surfaces d'océan, de mers calmes aux orageuses. En effet, l'ajout d'effets supplémentaires tel que des vagues de bateau est facile. De plus, le rapport qualité/coût est réglable, ainsi des images de plus haute qualité peuvent être calculées en utilisant le même modèle.

L'animation de la lave [PIE99] coulant en pentes d'un volcan apporte plusieurs défis : la modélisation des particularités mécaniques de la lave[1] et le calcul, dans un temps raisonnable, des interactions à l'intérieur du flux, entre le flux et une base de données de terrain complexe; et finalement, rendant l'aspect visuel du flux.

[1] Comment elle se développe dans le temps selon la température.

Des flux de lave naturels présentent une large diversité de morphologies et de structures. Parmi d'autres facteurs, la topographie de terrain et le débit du cratère affectent la diffusion de la lave.

Le terrain est défini par un modèle de carte d'élévation digital (MED), c'est-à-dire une grille 2D où les altitudes sont stockées, pour une détection plus facile de collision entre une particule[1] et le terrain (par interpolation bilinéaire) lors de la projection de la particule sur le plan horizontal. En pratique, la détection de collision est seulement exécutée pour les particules qui sont à la surface du flux.

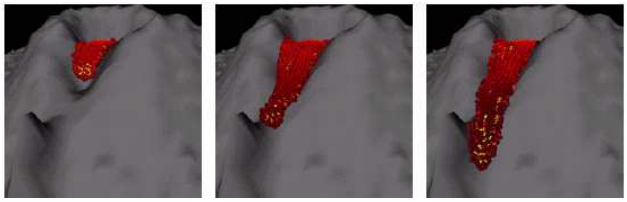


Figure A.7 : Un exemple de flux de lave. Environ 3000 particules sont étendues du volcan dans la dernière image.

Les avantages principaux du modèle de particules utilisé consistent en ce que l'utilisateur peut définir le comportement macroscopique du matériel par l'équation d'état, et la possibilité de contrôler la densité massive restée de la lave.

c- Phénomènes gazeux.

Le gaz n'a aucune définition géométrique ; sa modélisation, son rendu et son animation sont souvent en corrélation. Leurs mouvements sont généralement mentionnés comme un calcul dynamique de fluides (CFD : *computational fluid dynamics*) [PAR02, NEY97]. Le gaz est d'habitude traité comme *compressible*, ce qui signifie que la densité est variable dans l'espace et le changement de calcul de la densité fait partie du coût de calcul [PAR02].

Il y a trois approches de la modélisation du gaz [PAR02] : les méthodes basées sur grille (formulations *Eulérienne*), les méthodes basées sur les particules (formulations *Lagrangienne*) et les méthodes hybrides.

La modélisation de nuages est une tâche très difficile en raison de leur complexité, leur amorphe, leur structure. Dans ce cas il est facile de juger le réalisme d'un modèle de nuage.

Kajiya a produit le premier modèle volumique du nuage dans l'infographie [PAR02]. *R. Voss*, *Gardner* et *Kluyskens* ont employées des surfaces semi transparentes pour produire les images de nuages convaincantes. Bien que des techniques basées surface puissent produire des images réalistes de nuages vus d'une distance, ces modèles ne permettent pas à l'utilisateur de voyager à travers et d'inspecter leur intérieur.

En 1997, *Neyret* a produit quelques résultats préliminaires d'un modèle de nuage basé sur des caractéristiques physiques générales, telles que des processus de convection et de bullage [NEY97]. Ce modèle semble prometteur pour simuler les nuages convectifs[2]. Cependant, il a utilisé des surfaces (de grandes particules) pour modeler la structure de nuage. A l'année 2000, *Neyret* [Ney00] a proposé un

[1] Représentée par une sphère de rayon *ri*.
[2] Ils sont très denses, d'albedo très proche de un, la zone de transition entre intérieur et extérieur est très fine, la surface est composée de forme sphérique.

modèle de rendu analytico-phénoménologique, permettant de construire très rapidement des images de haute qualité visuelle (effets et résolution).

D'autres auteurs, tel que *Nishita* et al. *Stam* et *Ebert* [PAR02], ont employé l'idée de fonctions implicites rendues par volume pour la modélisation du nuage volumique.

Malgré la complexité des processus physiques qui forment des nuages, la plupart de leurs aspects visuels importants ont été efficacement modelés par les chercheurs. Cependant, il reste des défis en termes de contrôle de mouvement de nuage ainsi que le rendu.

d- Feu.

Le feu est particulièrement un processus intensif et difficile à modéliser. Il possède toutes les complexités de fumée, des nuages, en plus de la complexité de changement des processus actifs internes produisant la lumière, le mouvement et la création des attributs d'affichage qui varient rapidement [PAR02].

L'approche de système de particule, utilisée pour modéliser le feu, emploie une hiérarchie de particules à deux niveaux. Le premier niveau de particules est placé au point d'impact pour simuler la détonation initiale. Le deuxième consiste en anneaux concentriques de particules, chronométrés pour progresser du point central vers l'extérieur, formant le mur de feu et des explosions (Figure A.8) [PAR02]. Chaque anneau de la hiérarchie de deuxième niveau se compose d'un nombre de systèmes de particules individuels qui sont placés en chevauchement sur l'anneau afin de former un anneau continu. Les systèmes de particule individuels sont modelés pour ressembler aux explosions.

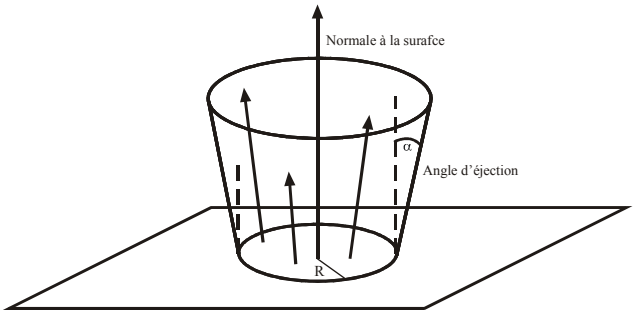


Figure A.8 : Exemple de génération de feu.

3.4. Les représentations alternatives.

3.4.1. Codage direct du comportement lumineux.

a- Les Systèmes de particules

Reeves [Pérc98, PAR02] a introduit la modélisation par particules pour représenter les objets aux formes mouvantes : flammes, fumées, feux d'artifices, torrents, cascades, nuages, champs ou végétation sous le vent.

Figure A.9 : Un exemple de paysage généré par des systèmes de particules.

L'idée initiale est de représenter des géométries complexes ou difficilement représentables avec de très grands nombres de particules[1], munie d'attributs tels qu'une position, une couleur, une transparence, une vitesse, une taille, une forme, une durée de vie. Ces particules évoluent dans l'espace en subissant différentes influences (gravité, lois de croissance, etc.) et forment ainsi des trajectoires. Il n'y a pas d'interaction entre les particules dans le modèle initial.

Reeves et *Blau* présentent en 1985 des applications de systèmes de particules pour la modélisation et le rendu d'éléments naturels : arbres, prairies, …etc. L'une des nouveautés est que les particules peuvent interagir entre elles (e.g. collisions) ; donc la forme et la topologie des objets ainsi engendrées émergent du comportement d'ensemble des particules, et ne sont pas connus à priori [PAR02].

K.Sims a utilisé ces systèmes de particules pour réaliser des animations très réussites (de cascades par exemple). Par ailleurs une notion de particule orientée a été introduite pour modéliser des objets déformables ou réaliser des animations [PAR02].

Outre la spécificité de son champ d'application, une des limites de cette technique est son incapacité de modéliser une structure pour une espèce d'arbres donnée. En outre, la spécificité de son algorithme de rendu, qui dessine les particules comme des traits de crayon, diffère totalement de la représentation classique et rend assez difficile son intégration à une scène existante.

b- Rendu à base de points.

M. Levoy et *T. Whitted* ont introduit, en 1985, une nouvelle représentation hybride située entre les représentations à base d'images et les représentations polygonales [MEY01]; dont l'idée est d'utiliser le point comme des particules pour rendre un objet solide. Ils présentent les problèmes fondamentaux comme la reconstruction de la surface, la visibilité et le rendu de surfaces semi transparentes. Mais ce n'est qu'en 1998 que *J.P. Grossman* et *Dally* reprirent cette idée et formalisèrent pour la première fois l'utilisation de points comme primitive de rendu.

[1] Primitives géométriques classiques simples (point, polygone, sphère, …etc.)

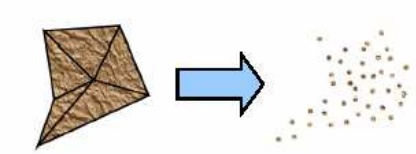


Figure A.10 : Maillage triangulaire (avec texture) vers un nuage de point (échantillonnage).

Les objets sont représentés par un ensemble d'échantillons ponctuels appartenant à leur surface. En fait, ces échantillons sont communément appelés surfels pour « élément de surface » (« surface element » en anglais). Ce terme a été introduit mathématiquement par *Herman* en 1992, mais *Pfister* et al. [PFI00] proposent une nouvelle définition plus adaptée. D'après *Pfister*, un surfel est un n-uplet de dimension 0 avec des attributs de forme et de matière qui approxime localement la surface d'un objet. De plus, les surfels ne contiennent aucune connectivité explicite avec leurs voisins, ce qui en fait une représentation très souple à manipuler. Le coût d'affichage d'un objet est proportionnel à sa taille à l'écran et non à sa complexité intrinsèque. Ce qui permet de conserver un taux de rafraîchissement de l'image constant, même avec beaucoup d'objets affichés.

Stamminger [STA01] montre aussi que cette technique est bien adaptée au rendu de phénomènes naturels variés comme une montagne, le mouvement de l'eau ou le rendu d'arbres.

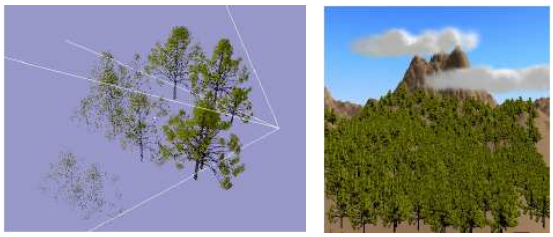


Figure A.11 : À gauche : des arbres représentés par des points. Plus on s'éloigne de la caméra moins il y a de points affichés. À droite : un exemple de paysage rendu avec la technique des points.

Il existe deux manières d'aborder la reconstruction de la surface. Une première approche est de travailler dans l'espace image [PAU01, BAA01]. Avec ce type d'algorithme, il est possible de prendre en compte des surfaces semi transparentes (utilisation d'un A-buffer), par contre seule une implémentation logicielle est actuellement possible. Pour bénéficier d'une accélération par le matériel graphique il est nécessaire d'exprimer cette reconstruction dans l'espace objet [RUS00, KAL01].

Cette technique est jeune et comporte certains défauts : texturation limitée, objets transparents et semi- transparents difficilement représentables, le calcul par moyenne des normales lorsqu'on simplifie le modèle est contestable, …etc. Néanmoins l'idée semble très prometteuse, pour prouver la quantité de travaux en cours sur ce sujet.

c- Shaders.

L'idée de *shader*[1] est de représenter un groupe de primitives par sa forme, associée à une formule analytique décrivant son comportement photométrique et son opacité globale (Figure A.12), c'est-à-dire par un modèle d'illumination (*shader*). Ce *shader* est obtenu en intégrant un modèle d'illumination simple (e.g. celui de *Phong*) sur l'ensemble des primitives du groupe en tenant compte

[1] Modèle d'illumination analytique et hiérarchique destiné à représenter la nature microscopique d'un objet ou les phénomènes sous–pixel lors de rendu.

de la visibilité (Ceci se traduit par une restriction du domaine d'intégration), et en calculant leur opacité moyenne [MEY01].

Cette idée est présente dans beaucoup de modèles d'illumination de surface ou par exemple le modèle de *Goldman* ou celui de *Kajiya* pour la représentation de fourrure. *Gardner* a proposé, en 1985, un *pseudo shader* pour synthétiser la complexité des surfaces naturelles [PER98]. Il utilise une sommation d'un certain nombre de sinusoïdes d'amplitudes et de fréquences variées. Cette méthode a permis de visualiser de manière efficace des arbres, des montagnes et plus particulièrement différents types de nuages.

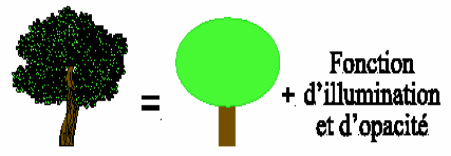


Figure A.12 : Un objet vu de loin peut être représenté par sa forme et son modèle de *shader*.

Figure A.13 : Un chien rendu par le *shader* de *Goldman*.

Alexandre Meyer a introduit, en 1998, un *pseudo shader 3D* pour rendre des forêts de pins, dans la mesure où la connaissance à priori sur la distribution des aiguilles est très forte. Il dérive une hiérarchie de trois *shaders* intégrant l'illumination (y compris les ombres et la transparence résiduelle) à l'échelle d'une aiguille, d'un cône d'aiguilles, et de toute une touffe. L'implémentation actuelle (peu optimisée), tourne environ 8 fois plus vite que *Rayshade* (un logiciel de rendu par lancer de rayons), pour calculer l'image d'une forêt de sapins sans artefacts visuels.

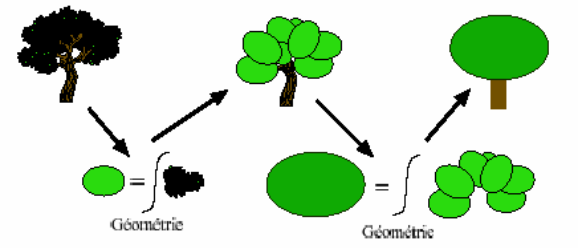


Figure A.14 : Principe d'une hiérarchie de *shaders* analytique dans le cas d'un arbre.

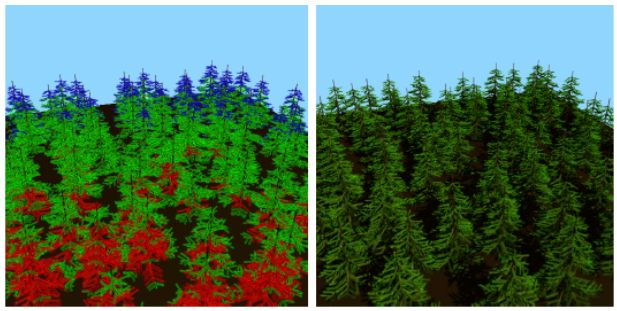


Figure A.15 : À gauche : Les trois niveaux de détails (en rouge, en vert et en bleu). À droite : 80 sapins.

d- Bump mapping.

Le *Bump Mapping*, appelé également *Embossing*, est une technique qui renforce le réalisme des images, des textures et des objets dans un environnement 3D, en donnant l'illusion de profondeur sur des surfaces apparemment lisses, grâce à des effets de relief.

Dans le monde réel, l'impression de profondeur est donnée par la quantité de lumière réfléchie par la surface de l'objet et perçue par l'œil : la quantité de lumière réfractée par l'objet et la direction qu'elle prend sont déterminées par le type de surface, lisse ou rugueuse, plate ou saillante, sur laquelle la lumière se reflète. Plus la direction de la lumière est perpendiculaire à l'objet, plus cette lumière va s'y refléter. Comparativement, moins la direction de la lumière est perpendiculaire à la surface, moins la lumière est reflétée.

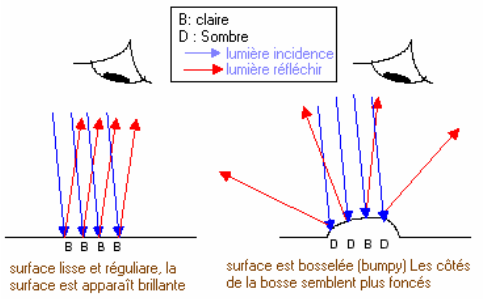


Figure A.16 : Réflexion de la lumière sur des surfaces plates et bosselées.

Perlin, en 1985, propose de généraliser la notion de perturbation de la normale (Bump Mapping) aux textures 3D en définissant une fonction appelée « *Dnoise* » analogue à la fonction de bruit qui, au lieu de retourner un scalaire, retourne un vecteur pseudo-aléatoire. Ce vecteur sert à perturber la normale en (x, y, z) par simple addition au vecteur normal, N, en (x, y, z) : N' = N + Dnoise(x, y, z).

3.4.2. Le codage texturelle.

a- Rendu à base d'images.

Les techniques de rendu basé sur des images ont été introduites pour essayer d'éviter l'étape de modélisation, ainsi que pour tenter d'accélérer et d'améliorer le rendu en utilisant l'information contenue dans des images capturées du monde réel [GUI00].

La technique de rendu à base d'image la plus fréquemment utilisée en synthèse d'images est le plaquage de textures où la complexité apparente d'un objet est représentée par l'application d'une image sur sa surface. La forme grossière de l'objet est représentée par sa description géométrique, les détails étant décrits par les informations dérivées de l'image de la texture qui lui est appliquée. Plus récemment, des variantes plus intelligentes du plaquage de textures ont été développées, en particulier le plaquage de textures projectives permettant à un ensemble de primitives 3D de partager une même texture définie pour un point de vue particulier. Dans une certaine mesure, ce principe peut être étendu à l'utilisation d'une unique image panoramique pour représenter l'environnement. C'est le cas dans la technologie **Quicktime-VR** développée par *Apple* [GUI00].

Des représentations alternatives de la scène peuvent être utilisées, en ajoutant aux images des informations supplémentaires telles que la profondeur ou le flot optique (décrivant le déplacement apparent des pixels entre deux images). L'utilisation de techniques de re-projection d'images permet alors d'obtenir de nouvelles vues en tenant compte des effets de parallaxe (dus à la projection perspective) et des occlusions entre les différents objets de la scène. La difficulté majeure des techniques précédentes concerne l'estimation des données additionnelles (profondeur, flot optique).

Lors de la re-projection des images, il se peut que des régions de la scène qui n'étaient pas visibles, le deviennent. Pour pallier ce problème, une méthode consiste à combiner les informations colorimétriques et géométriques de plusieurs images dans une seule image à plusieurs niveaux de profondeurs (appelée LDI). Une autre alternative consiste à utiliser une image à plusieurs centres de projection [GUI00].

Une alternative consiste à considérer un ensemble d'images comme une base de données de rayons traversant la scène. Les rayons non présents dans la base de données étant interpolés. L'avantage de ces méthodes est qu'elles ne nécessitent qu'une interprétation minimale des données, mais supposent avoir accès à un grand nombre d'images [GUI00].

Ces nouvelles techniques de rendu offrent l'avantage que la complexité des modèles est sans limite, et le temps de rendu est indépendant de la complexité (il dépend plutôt de la taille de l'image à produire). Bien qu'en général moins précises à cause de la nature discrète des images utilisées, ces techniques offrent aussi l'avantage d'être assez rapide [Bais98].

b- Les nuages de panneaux d'affichage (billborad clouds)

Un *nuage de panneaux d'affichage* est un ensemble de polygones (ou panneaux) texturés, partiellement transparents, avec des tailles, orientations et résolutions de texture indépendantes (Figure A.17). Typiquement, les *billboards* sont employés pour représenter des objets tels que des nuages ou des arbres, très difficiles à modéliser explicitement [DEC02, POR04].

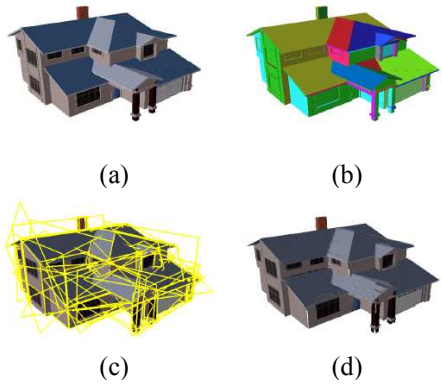


Figure A.17 : Exemple d'un nuage de panneaux : (a) modèle original (1,960 polygones) (b) rendu en utilisant une couleur par panneau (c) Vue de 52 panneaux texturés (d) nuage de panneaux rendu.

Le principe de la méthode de *Décoret* et al. [DEC02, POR04] est de remplacer un objet par la combinaison de plusieurs *billboards* représentant cet objet sous différents angles. Leur méthode permet de répartir automatiquement un certain nombre (fixé par l'utilisateur) de plans recouvrant l'objet de façon optimale. Cet ensemble de *billboards* est simplement affiché lors du rendu, les plans se superposent et se combinent à l'aide de la composante de transparence des textures.

Beaucoup de techniques ont employé la notion de polygones multiples, partiellement transparents pour accélérer l'affichage de modèles complexes. En effet, ces techniques peuvent être vues comme des cas spéciaux des nuages de panneaux (les panneaux classiques, les couches d'imposteurs où tous les panneaux sont parallèles, les caches d'image où un panneau est associé à chaque région d'une partition spatiale, …etc.). [DEC02]

L'avantage principal de nuages de panneaux [DEC02, POR04] consiste en ce qu'aucune information topologique (comme la connectivité de polygone) n'est exigée. De plus la complexité visuelle de nuage de panneaux résulte principalement de la texture, et aucune description complexe de frontière n'est nécessaire.

Les nuages de panneaux sont fortement efficaces dans la simplification de modèles complexes avec des textures multiples vers quelques dizaines de polygones texturés. La représentation de nuage de panneaux permet alors un rendu très rapide, ou un test de collision efficace. La qualité visuelle des modèles simplifiés est très haute et contrôlée par l'utilisateur. Le problème principal de cette méthode est que l'éclairement ne peut varier, du fait de l'emploi de placage de texture dépendant du point de vue.

c- Textures procédurales

Les textures procédurales ont été proposée par *Ken Perlin* en 1985, dont l'idée est que la plupart des textures naturelles se caractérisent avant tout par une certaine irrégularité et une grande quantité de détails distribuée plus ou moins aléatoirement. De cet effet, *Perlin* a introduit la notion de *bruit volumique* (*solide noise*) afin de générer des textures 3D d'apparence naturelle comme le marbre, nuage, bois.

Perlin et *peachey*, ont utilisé le bruit[1] de la manière suivante : la texture est décrite dans un premier temps par un modèle mathématique simplifié, appelé fonction de base ou modèle de base. Ensuite cette fonction de base est perturbée par un bruit ou une autre procédure stochastique pour lui donner un aspect plus irrégulier et plus naturel. De plus, il a introduit un autre type de fonction stochastique, nommée turbulence qui sert à perturber le modèle de base ; elle est définie par une somme de bruit sur différents niveaux de fréquences (octaves) et d'amplitudes [PER98].

Le temps de calcul nécessaire pour générer une texture procédurale est grand, ce qui ne permet pas d'avoir un rendu de scènes animées en temps réel. A cet effet, *Antoine Miné* [ANT01] a introduit, en 2001, « les textures procédurales en temps réel », pour cela il a utilisé la bibliothèque de rendu **OpenGL** de **Silicon Graphics**. L'inconvénient majeur de cette méthode est le choix des paramètres de texture ; ces derniers sont choisis par expérience pour obtenir un type de texture particulier.

En 1996, *Worley* a introduit les textures cellulaires, permettent de décrire des minéraux en cristaux, des peaux à écaille et de simuler des surfaces ayant un aspect externe organique. Le principe de base consiste à générer des échantillons d'éléments géométriques sur une surface tout en simulant le développement de cellules biologiques. Les textures cellulaires sont obtenues par l'interaction de différents éléments appelés cellules 3D liées à une surface et forme donc une texture 3D géométrique, orientée et colorée [KEN97].

Ce type de texture, peut être combiné avec d'autres systèmes tel que le système de particules ou les systèmes à réaction-diffusion, afin de former un seul modèle.

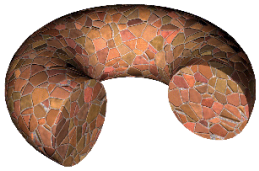


Figure A.18 : Exemple de texture cellulaire.

Figure A.19 : Exemple de texture par Réaction- Diffusion.

[1] Une fonction qui pour tout point de R^3, retourne une valeur pseudo aléatoire comprise entre –1 et 1.

Le modèle de texture par réaction - diffusion décrit une méthode biologique pour la génération de texture tout en gardant l'aspect géométrique des surfaces. Réaction – Diffusion (RD) est un processus dans lequel deux produits chimiques ou plus se diffusent sur la surface d'un objet et réagissent entre eux pour former un échantillon stable [PER98].

Ce modèle est proposé pour la simulation de processus d'interaction locale non linéaire générant des échantillons biologiques. Ces derniers compensent les effets non uniformes de la paramétrisation des surfaces, ils sont dirigés par deux aspects concurrents [PER98] :

- Diffusion d'éléments à travers les tissus (la surface).
- Réaction de ces éléments à l'excitation ou à la négligence des autres éléments existant dans le même tissu. Le résultat de la réaction peut être une production ou une destruction de ces éléments.

3.4.3. Codage volumique.

Les textures volumiques ont été introduites dans les années quatre-vingt par *Kajiya et Kay* [NEY96]. En premier lieu ce modèle était dédié à la construction des fourrures puis il s'est développé au fur et à mesure jusqu'à devenir un modèle plus général incluant la notion de multi-échelles.

a- La texture volumique de *Kajiya* et *Kay*.

Introduite en 1989, cette texture a généré de l'intérêt, du fait qu'elle capture la complexité des surfaces en réduisant l'aliassage d'une façon importante. *Kajiya et Kay* ont introduit essentiellement cette texture pour générer un modèle de fourrure [NEY96].

Le principe consiste à fabriquer un échantillon de texture volumique; en l'occurrence un échantillon de fourrure, lequel est stocké en un seul exemplaire dans un volume de référence, dont les copies déformées, appelées « texels », seront plaquées sur une surface composée de patchs bilinéaires, constituant une peau épaisse continue à la surface d'un objet.

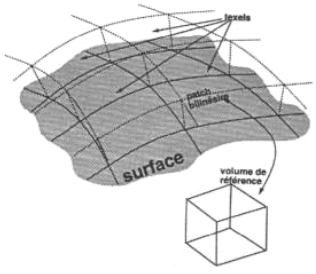


Figure A.20 : Principe de placage d'une texture volumique.

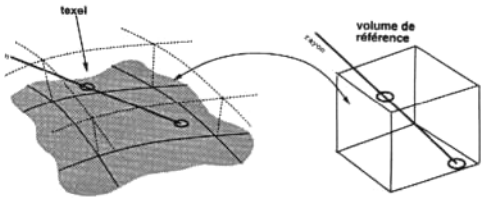


Figure A.21 : Rendu de la texture.

Le volume de référence comporte un ensemble d'éléments de volumes appelés « voxels ». Le contenu de chaque voxel est une probabilité d'occultation isotrope, plus un comportement photométrique local, qui consiste en un repère local et une fonction de réflectance analytique, permettant au texel de refléter la lumière comme s'il contenait vraiment un morceau de surface.

Au rendu, un rayon intersecte la surface sous-jacente et la surface parallèle qui correspond au plafond des texels. On se ramène alors dans l'espace de la texture où les texels s'identifient au volume de référence.

Un échantillonnage stochastique du rayon est réalisé afin d'alléger les calculs; ensuite, l'illumination locale est effectuée à l'aide du modèle de réflectance qui simule la présence d'un cylindre. Un rayon est envoyé vers la source de lumière afin de tester l'ombrage du voxel courant.

Ce modèle est le premier dans le domaine à avoir traiter les scènes complexes répétitives et a été proposé pour éviter les problèmes qu'auraient posés les modèles géométriques, il présente aussi plusieurs limites :

- ❖ Ce modèle reste un modèle dédié à la fourrure.
- ❖ Le modèle ne tient pas en compte de la physique du matériau (opacité, illumination) ce qui nuit au réalisme du résultat.
- ❖ Le rendu est particulièrement lent.

b- Texture volumique de *Neyret*.

Afin de réaliser une représentation efficace des géométries complexes et partant des textures volumiques de *Kajiya et Kay*, *Neyret* a proposé en 1995 une autre approche hiérarchique, pour le rendu de la texture volumique [NEY96].

Les principales caractéristiques dont ce modèle se base sont :

- ❖ Utilisation de l'octree pour le codage volumique de l'échantillon de texture.
- ❖ Une représentation multi-échelle dans l'esprit du *mip-map*.
- ❖ Des informations géométriques (encodé par une probabilité d'occultation) et photométriques (encodé par une approximation de la distribution des normales).

En ce qui concerne la modélisation de la réflectance, Neyret [NEY98] a choisi la BRDF[1] qui caractérise l'anisotropie. Pour représenter cette dernière, il a considéré que la BRDF est issue de la distribution des normales locales (NDF : Fonction de Densité des Normales). La réflectance est donc obtenue en intégrant un modèle simple, typiquement celui de *Phong*, sur la distribution des normales.

Neyret a choisi l'ellipsoïde comme primitive de compromis pour encoder la réflectance puisqu'il permet de simuler des formes communes comme la sphère, le cylindre et l'élément de plan. De plus, l'opérateur d'addition est défini pour les ellipsoïdes, ce qui est une caractéristique très importante pour permettre la représentation multi-échelle. Il utilise donc les ellipsoïdes comme support des normales.

Le rendu de la texture volumique se fait en deux passes :

- ➢ Rendu global qui se fait au niveau de chaque texel en utilisant un algorithme classique de lancer de cône en parcourant l'espace scène (ou géométrique), l'espace texel puis l'espace objet en utilisant l'interpolation tri-linéaire ; où le rayon subit une légère déformation qui va simuler la déformation du texel pour épouser la totalité de la boîte.

[1] Bidirectional Reflectance Distribution Function.

- Rendu local qui se fait au niveau de chaque voxel en intégrant le modèle d'illumination locale de Phong.

  Cette méthode présente quelques limites, à savoir :

  - Le coût inhérent à l'animation du texel de référence.
  - Le stockage du volume de référence est géant et exponentiel suivant le nivaux de profondeur utiliser lors de codage de l'octree.

4. Conclusion.

Nous avons présenté dans cette partie une vue générale sur les méthodes traitant la complexité que se soit géométrique ou temporelle en synthèse d'images. Nous avons commencé par les approches classiques, puis nous avons passé à présenter les techniques émergentes. Les recherches de ces dix dernières années sont tous orientées vers ces techniques émergentes et surtout les représentations à base de points et à base des images.

Annexe B :
Vision 3D et calibrage de caméra.

1. Introduction.

La vision 3D peut être expliquée par la stéréovision. Ce type de vision est une reproduction du système visuel humain qui consiste à voir une même scène sous deux incidences différentes. La différence entre ces incidences permet d'établir la profondeur de la scène, et donc d'en avoir une représentation tridimensionnelle.

Différentes étapes sont nécessaires pour réaliser une reconstruction de surface par ce type de système. Généralement, L'étape de calibrage des appareils d'acquisition, qui nous intéresse, consiste à déterminer certains paramètres mathématiques et physiques des appareils photos utilisés, permettant entre autre de passer du référentiel de la caméra au référentiel de la scène.

Avant d'aborder au problème de calibrage, il est nécessaire de présenter le modèle de projection utilisé pour modéliser la projection d'une caméra réelle ; c'est le modèle de projection perspective.

2. La projection sténopé.

Le modèle sténopé modélise une projection perspective avec un **centre de projection** *C* et un **plan image** Π. Le centre de projection est souvent aussi appelé **centre optique** et le plan image, **rétine**. La projection d'un point ***Q*** de l'espace est définie comme étant l'intersection *q* du **rayon de projection** <***CQ***> avec le plan image. Cette projection peut être représentée par une transformation projective P de $\boldsymbol{IP}^3$ vers $\boldsymbol{IP}^2$ [STU97] :

$$q3 \sim P_{3\times4}Q4 ,$$

Où les indices indiquent les dimensions des entités. La matrice P est appelée **matrice de projection**.

On suppose dans la suite que la matrice de projection est toujours de plein rang, c'est-à-dire de rang 3. Le centre de projection est représenté (en coordonnées homogènes) par n'importe quel vecteur *C* du noyau de P. Ceci correspond bien au fait que la projection du centre de projection n'est pas définie : $PC = 0_3$ (nous rappelons que les vecteurs nuls ne sont pas admis comme vecteurs de coordonnées homogènes).

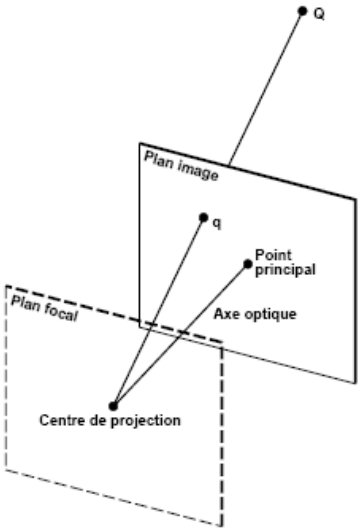


Figure B.1 : *Le modèle sténopé.*

On définit maintenant quelques autres notations associées au modèle sténopé : l'**axe optique** est la droite passant par le centre de projection et perpendiculaire au plan image. Son intersection avec le plan image est le **point principal**. Le **plan focal** est constitué des points de $\mathbf{P}^3$ qui se projettent sur des points à l'infini du plan image. Il s'agit donc du plan passant par le centre de projection qui est parallèle au plan image [STU97].

On distingue maintenant les deux principaux modèles de caméra : une **caméra perspective** est une caméra effectuant une projection sténopé, dont le centre de projection est un point qui n'est pas à l'infini. Une telle projection est aussi appelée projection perspective parfaite. Une **caméra affine** est une caméra dont la matrice de projection est une transformation affine, c'est-à-dire de la forme suivante [STU97] :

$$P \sim \begin{pmatrix} \overline{P}_{2\times 3} & \overline{\mathbf{p}}_2 \\ \mathbf{0}_3^T & 1 \end{pmatrix} .$$

Une condition nécessaire et suffisante pour qu'une caméra soit affine est que son plan focal soit à l'infini. Ceci implique que le centre de projection est à l'infini. *Toutes* les droites perpendiculaires au plan image passent par le centre de projection et donc les concepts d'axe optique et de point principal n'ont plus de sens pour la caméra affine.

Dans le prochain paragraphe, nous décrirons les paramètres intrinsèques et extrinsèques de la caméra perspective de manière classique.

3. Paramètres intrinsèques et extrinsèques de la caméra perspective.

La matrice de projection P ne révèle pas directement les grandeurs physiques qui caractérisent le processus de prise d'images avec une caméra réelle. On va donc, dans la suite, décrire une façon de décomposer P en des paramètres reflétant des grandeurs physiques de la caméra (et du système d'acquisition). Cette caractérisation de la caméra ne capte pas les effets non modélisables par le modèle sténopé idéal, comme par exemple les distorsions optiques. La plupart des chercheurs en vision

par ordinateur se contentent pourtant du degré d'approximation de la réalité fourni par ce modèle (ce qui est différent dans le domaine de la photogrammétrie où la précision maximale des mesures est l'un des buts essentiels) [STU97].

D'abord, nous introduisons quelques repères de coordonnées par rapport auxquels nous allons définir les transformations composant **P** [STU97, MOR93] :

- Un repère euclidien global R_s = (O, $\vec{U}$, $\vec{V}$, $\vec{W}$) par rapport auquel les points tridimensionnels sont donnés. Nous l'appelons le *repère monde*.

- Un repère 3D local R'_s = (C, $\vec{U}$', $\vec{V}$', $\vec{W}$') attaché à la caméra, le *repère caméra*. C'est un repère euclidien d'origine C, le centre de projection, et dont les vecteurs unitaires sont déterminés par :

  - $\vec{W}$' est orthogonal au plan image,
  - $\vec{U}$' est paralléle à l'axe horizontal du plan image, $\vec{u}$,
  - $\vec{V}$' est tel que ($\vec{U}$', $\vec{V}$', $\vec{W}$') soit orthonormé direct.

- Un repère 2D R_i = (o, $\vec{u}$, $\vec{v}$) du plan image, le *repère image*. On note $\alpha = (\vec{u},\hat{\vec{v}})$ l'angle orienté entre les deux vecteurs de base.

- Un deuxième repère 2D R'_i = (c, $\vec{u}$', $\vec{v}$') du plan image, le *repère pixels*, associé à la grille des pixels. C'est un repère orthonormé d'origine c. Les vecteurs unitaires du repère sont :

  - $\vec{u}$'= $\vec{U}$'
  - $\vec{v}$'= $\vec{V}$'

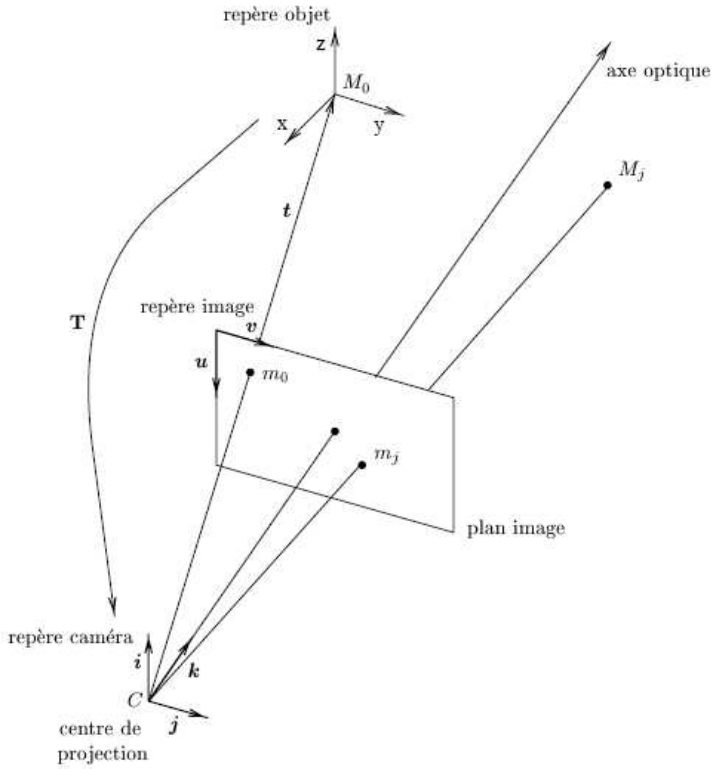


Figure B.2 : La modélisation géométrique d'une caméra.

Le repère caméra est souvent choisi de la façon suivante : le centre de projection est pris comme origine, l'axe optique coïncide avec l'axe des *Z* et le plan image est le plan *Z* = *f*, où f est la distance focale de la caméra. Le repère image a son origine au point principal et ses axes sont parallèles aux axes des *X* et *Y* du repère caméra. Quant au repère pixels, nous choisissons de mettre son origine au coin supérieur gauche de la matrice des pixels ; un des deux axes du repère est parallèle à un axe du repère image. Le deuxième axe est généralement perpendiculaire au premier, mais l'omission de cette contrainte permet de modéliser des caméras avec des pixels non rectangulaires, en forme de parallélogramme. Ces repères sont présentés dans la figure 2.2.

La projection perspective **P** peut maintenant être décomposée en une suite de transformations entre les repères définis ci-dessus [STU97] :

$$\mathsf{P} \sim \mathsf{A}\ \mathsf{P}^{P}\ \mathsf{T}\ ,$$

Où nous avons :

- **T** est la transformation rigide entre le repère monde et le repère caméra. Elle est de la forme :

$$\mathsf{T}_{4\times4} = \begin{pmatrix} \mathsf{R}_{3\times3} & \Leftrightarrow\mathbf{R}\mathbf{t}_3 \\ \mathbf{0}_3^{\mathsf{T}} & 1 \end{pmatrix}\ ,$$

Où **R** est une matrice de rotation (une matrice orthogonale). **R** indique l'orientation de la caméra par rapport au monde et **t** sa position.

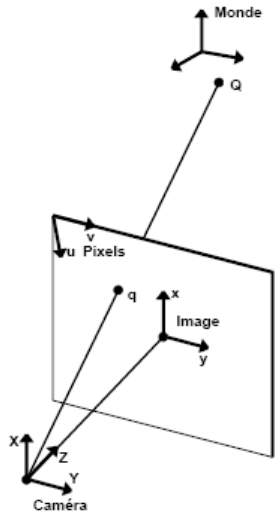


Figure B.3 : Illustration d'une projection perspective.

- $\mathbf{P}^{P}$ est la projection perspective proprement dite. Un point Q ~ $(X, Y, Z, 1)^{T}$ dans le repère caméra se projette sur un point q ~ $(x, y, 1)^{T}$ avec :

$$x = f\frac{X}{Z} \qquad y = f\frac{Y}{Z}\ .$$

En utilisant des entités homogènes, cette projection peut être représentée par la matrice :

$$\mathsf{P}^{P}_{3\times4} \sim \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}\ .$$

- **A** est la transformation (affine) reliant les repères image et pixels. Sa forme générale est :

$$\mathsf{A} = \begin{pmatrix} k_u & -k_u \cot\Theta & u_0 \\ 0 & \frac{k_v}{\sin\Theta} & v_0 \\ 0 & 0 & 1 \end{pmatrix} .$$

Les paramètres utilisés sont les suivants [STU97, CHR98] : k_u et k_v sont les dimensions des pixels (la longueur inverse de leurs « côtés ») ; Θ l'angle entre les axes (les côtés) des pixels ; $(u_0, v_0)^T$, les coordonnées du point principal dans le repère pixels.

Puisque **R** et **t** définissent le positionnement de la caméra dans son environnement, ils sont souvent appelés **paramètres extrinsèques** de la caméra. Les autres grandeurs (k_u, k_v, f, Θ, u_0 et v_0) décrivent des propriétés physiques spécifiques de la caméra et de la carte d'acquisition et sont donc appelées **paramètres intrinsèques**.

Il est facile de voir que les paramètres k_u, k_v et *f* ne comptent que pour deux paramètres indépendants : le produit de **A** et $\mathbf{P}^P$ est :

$$\mathsf{A}\ \mathsf{P}^P = \begin{pmatrix} k_u f & -k_u f \cot\Theta & u_0 & 0 \\ 0 & \frac{k_v f}{\sin\Theta} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} \mathsf{K}_{3\times3} & \mathbf{0}_3 \end{pmatrix}$$

Où k_u et k_v apparaissent toujours avec le facteur *f*. Nous restreignons donc l'ensemble des paramètres intrinsèques aux suivants : $\alpha_u = k_u f$, $\alpha_v = k_v f$, Θ, u_0, v_0. Les paramètres α_u et α_v mesurent la distance focale, avec comme unité les dimensions des pixels.
Nous appelons **K** la **matrice des paramètres intrinsèques** [STU97, MOR93] :

$$\mathsf{K} = \begin{pmatrix} \alpha_u & -\alpha_u \cot\Theta & u_0 \\ 0 & \frac{\alpha_v}{\sin\Theta} & v_0 \\ 0 & 0 & 1 \end{pmatrix} .$$

Le rapport $\tau = \dfrac{\alpha_u \sin\Theta}{\alpha_v}$ est appelé **rapport d'échelle** de la caméra. Avec des pixels rectangulaires (Θ = 90°), qui c'est le cas que nous le utilisons pour la reconstruction, le rapport d'échelle équivaut le rapport des dimensions d'un pixel.

4. Calibrage des appareils d'acquisition.

Quel que soit le type de système stéréoscopique développé, le calibrage des appareils photos est nécessaire. La méthode consiste en général à photographier un objet de contrôle à géométrie variable selon les besoins des utilisateurs.

Le but du calibrage est d'obtenir les paramètres mathématiques et physiques du système d'acquisition (Les paramètres intrinsèques extrinsèques). Les paramètres intrinsèques permettent de trouver l'expression mathématique de la fonction de projection, qui à un point dans une image, lui associe son correspondant dans un repère tridimensionnel. Les paramètres extrinsèques décrivent la position et l'orientation de l'appareil photo lors de l'acquisition.

4.1. Principe expérimental.

Le calibrage est généralement fait avec une mire comportant des points de contrôles, qui peuvent être des points, des intersections de droites ou encore des formes faciles à extraire à partir d'images numériques.

Les positions des points de contrôles sont connues à priori. Si c'est une grille, les dimensions de chaque carré sont connues. Si c'est un nuage de points, la taille de chaque point ainsi que la distance les séparant sont connues.

La géométrie de la mire dépend complètement des besoins des utilisateurs. Très généralement une mire à damier est utilisée, car on peut extraire suffisamment de paramètres dans l'image par une simple détection de coins.

Plusieurs types d'acquisitions peuvent être effectuées afin de calibrer les appareils photos. Voici quelques exemples qui permettent de mettre en avant différentes contraintes appliquées à la mire :

- La mire est posée perpendiculairement sur un rail et face à la caméra, puis elle est déplacée le long du rail par des intervalles constants. Ainsi, pour chaque position de la mire, sa position en profondeur dans l'espace par rapport à la caméra est connue.
- La mire est placée sur un mur, et dans différentes orientations. Elle est donc toujours dans un même plan.
- La mire est placée à l'emplacement de l'objet à reconstruire, mais posée dans différentes directions et orientations. Aucune contrainte ne lui est donc appliquée.

L'acquisition des images stéréoscopiques est évidemment la partie la plus cruciale pour une future reconstruction tridimensionnelle d'une scène.

L'acquisition en elle même n'est pas compliquée, mais elle doit faire face à différents problèmes qui peuvent rendre la phase d'appariement difficile voire impossible. Par exemple l'éclairage est primordial. Si une des caméras stéréoscopiques ne voit pas la scène selon le même éclairage, il existera des différences d'intensités des niveaux de gris dans les images stéréoscopiques. La mise en correspondance, étape suivante avant la reconstruction, reposant principalement sur les différences de niveaux de gris, un mauvais éclairage de la scène donnera lieu à des erreurs d'appariement.

Un autre problème fréquent est l'occlusion d'objets ou de personnes dans les images. En effet certains objets peuvent être présents dans une image et non dans l'autre. Ainsi lors de la mise en correspondance il y aura perte d'information et donc problème pour la future reconstruction.

Figure B.4 : Occlusion de partie d'une scène. Des branches disparaissent et apparaissent.

Autre cas, qui lui est un peu particulier. Si la scène à acquérir est un visage humain, la peau étant faiblement texturée, la mise en correspondance se basera sur des zones homogènes de niveaux de gris : il ne sera donc pas possible de trouver le bon correspondant d'un pixel d'une image dans l'autre image stéréoscopique puisque tous les points se ressembleront. Une parade consiste à rehausser artificiellement la texture de la peau par projection d'images comprenant des formes aléatoire sur le visage (utilisation d'un projecteur de diapositives standard).

En résumé, pour que les images acquises puissent être utilisées pour la reconstruction 3D, il faut donc parfois faire preuve de créativité et de sens pratique, et avoir une bonne connaissance de la scène à acquérir.

4.2. Détermination des paramètres extrinsèques en utilisant quatre points plats.

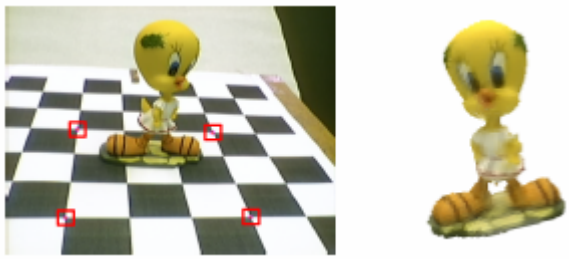


Figure B.5 : Une image d'entrée choisie avec quatre points de référence qui doivent être choisies manuellement par l'utilisateur, et la reconstruction de l'objet en utilisant la coloration de voxel.

On veut calculer la matrice des paramètres extrinsèques M_{ext}, l'une des parties cruciales de notre calibrage de caméra, à partir de quatre points plats sachant la correspondance connue entre des coordonnées du monde réelles et des coordonnées d'une image. Puisque les points sont plats, on les choisit pour être sur le plan Z = 0 dans le monde réel. On supposera que les coordonnées de l'image sont non déformées, donc nos équations deviennent [VAN04] :

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \sim \begin{pmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \\ 0 \\ 1 \end{pmatrix}$$

On a la susdite équation pour tous les quatre points de référence i : $i \in \{1, 2, 3, 4\}$. Puisque le troisième élément de $\overrightarrow{X}$ est 0 pour tous les points, on peut omettre cet élément et la troisième colonne de M_{ext} puisque leur produit sera toujours 0 après la multiplication matricielle [VAN04] :

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \sim \begin{pmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \\ 1 \end{pmatrix}$$

Le facteur homogène 1 de $\overrightarrow{X}$ deviendra le quatrième élément après ce vecteur a été multiplié par M_{ext} et deviendra toujours 1 puisque M_{ext} est une transformation rigide. Dans la matrice M_{int}, on voit que ce 1 n'est pas en réalité employé n'importe où (la quatrième colonne seulement contient des zéros), donc on peut, sans risque, omettre ce terme aussi [VAN04] :

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \sim \begin{pmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \\ 1 \end{pmatrix}$$

Par conséquent, cette matrice est devient maintenant réversible. Maintenant, la question qui se pose est comment c'est possible d'inverser une projection perspective, comme l'information de profondeur est perdue dans telles projections. Il y a une réponse simple à cette question : la projection n'est pas réversible du tout ; on utilise seulement le principe des coordonnées homogènes. Le facteur homogène est un facteur de graduation, il permet de représenter une ligne de tous les points 3D qui se projettent sur notre point 2D. Si on change ce facteur de 1, on obtient d'autres points sur cette ligne.

$$\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \\ 1 \end{pmatrix} = \mathbf{H} \begin{pmatrix} X_i \\ Y_i \\ 1 \end{pmatrix}$$

Nous avons maintenant une correspondance entre deux plans (un plan du monde réel et le plan image). Un placage entre deux plans est connu comme une homographie plan à plan et il peut être décrit par une matrice H (3x3). Le calcul d'une homographie est un problème résolu. On a quatre points avec deux correspondances de coordonnée chacun ; cela nous donne 8 équations. Après la conversion de ces équations à la forme matricielle et le détachement des éléments de H dans un vecteur séparé *h* nous aurions [VAN04] :

$$\begin{pmatrix} X_1 & Y_1 & 1 & 0 & 0 & 0 & -X_1x_1 & -Y_1x_1 & -x_1 \\ 0 & 0 & 0 & X_1 & Y_1 & 1 & -X_1y_1 & -Y_1y_1 & -y_1 \\ X_2 & Y_2 & 1 & 0 & 0 & 0 & -X_2x_2 & -Y_2x_2 & -x_2 \\ 0 & 0 & 0 & X_2 & Y_2 & 1 & -X_2y_2 & -Y_2y_2 & -y_2 \\ X_3 & Y_3 & 1 & 0 & 0 & 0 & -X_3x_3 & -Y_3x_3 & -x_3 \\ 0 & 0 & 0 & X_3 & Y_3 & 1 & -X_3y_3 & -Y_3y_3 & -y_3 \\ X_4 & Y_4 & 1 & 0 & 0 & 0 & -X_4x_4 & -Y_4x_4 & -x_4 \\ 0 & 0 & 0 & X_4 & Y_4 & 1 & -X_4y_4 & -Y_4y_4 & -y_4 \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{pmatrix} = 0$$

Le vecteur *h* peut être calculé à partir de cette équation en utilisant l'algèbre linéaire, en réduisant au minimum l'erreur dans de re-projection de nos points. Puisque nous avons 4 points, la solution non triviale est uniquement définie. Si on a plus de points disponibles, on peut les ajouter comme des rangées supplémentaires à la matrice A. Nous pouvons reconstruire M_{ext} à partir de H

comme si les colonnes 1 et 2 définissent les vecteurs de base pour les axes des X et des Y. Le vecteur de base de Z peut être calculé à partir des deux premiers en calculant leur produit vectoriel.

La partie de translation de M_{ext} est la 3ème colonne de H. Notre matrice est achevée en donnant à la 4ème rangé leur valeurs par défaut pour une transformation rigide : 0, 0, 0 et 1.

M_{ext} n'est pas nécessairement une transformation rigide maintenant, comme c'est possible d'avoir une longueur différente de 1 pour les vecteurs de base des axes de X et Y. Cela arrive parce que les points de référence ne sont pas généralement exacts. *Bouget* résout ce problème en conversant la partie rotative de M_{ext} aux coordonnées de *Rodrigues*. Il exécute alors une dégradation du gradient pour réduire au minimum l'erreur de re-projection dans les points de référence sur les 3 coordonnées et les 3 éléments du vecteur de translation de *Rodrigues*. Les coordonnées de *Rodrigues* peuvent seulement décrire des rotations sans changement d'échelle et assurer que la version finale de matrice M_{ext} soit en effet une transformation rigide [VAN04].

5. Conclusion.

Cette partie présente les principes de la vision 3D par ordinateur. Nous avons, donc, introduire le modèle de projection sténopé et les déférents paramètres caractérisant une caméra de projection perspective. Ces paramètres, lorsqu'ils sont déterminés, ils permettent d'effectuer un calibrage des appareils d'acquisition. Cette dernière étape permet d'effectuer l'opération de la reconstruction tridimensionnelle.

Annexe C :
Le placage de texture d'*OpenGL*

1. Introduction.

Dans cette partie, nous essayerons de présenter la partie essentielle du placage de texture en utilisant *OpenGL*. Cela inclut le chargement de la texture à la mémoire vidéo et le placage de la texture sur la surface destinée (la géométrie).

2. Chargement de textures.

Si on dispose de quelques matières comme des données d'image au format RGB dans un buffer et on veut l'appliquer à notre géométrie avec *OpenGL*, la première des choses que nous devons faire avant que *OpenGL* puisse utiliser ces données de texture est leur chargement à la mémoire vidéo. Une fois qu'une texture est chargée à la mémoire vidéo, elle peut être employée durant le temps d'exécution de notre application. Mais, Avant qu'une texture soit chargée à la mémoire vidéo, il y a quelques étapes d'initialisation qui doivent avoir lieu. Ci-dessous nous allons décrire l'ordre d'exécution de certains appels pour charger notre texture. Ces appels doivent être effectuer une fois l'application s'exécute [MIL00] :

- **GlBindTexture.**

  La première chose qui doit avoir lieu dans le processus de chargement de la texture est un appel à **glBindTexture**. **glBindTexture** spécifie pour *OpenGL* l'identificateur "id" de la texture que nous travaillerons avec. Un identificateur d'une texture "id" est juste un numéro que nous emploierons pour avoir accès à nos textures. Voici un exemple de cet appel :

  ```
  glBindTexture (GL_TEXTURE_2D, 13);
  ```

  Cet appel rend active la texture qui est associée à l'identificateur 13 la texture. N'importe quels appels qui ont un rapport avec le placage de texture *OpenGL* utilisent cette texture. Il est important d'utiliser ce numéro de nouveau plus tard pour appliquer la texture à la géométrie.

- **GlPixelStorei.**

  L'appel de **glPixelStorei** spécifie pour *OpenGL* la façon d'alignement des données qui vont être chargées. On présente ci-dessous un appel à **glPixelSrotei** :

  ```
  glPixelStorei (GL_UNPACK_ALIGNMENT, 1);
  ```

Cet appel spécifie pour *OpenGL* que les données d'un pixel qui vont être passées sont alignées dans un ordre d'octets, cela signifie que les données d'un pixel ont un octet pour chaque composante ; un pour le rouge, le vert et le bleu.

- **GlTexParameteri.**

  glTexParameteri initialise les divers paramètres pour la texture courante. Chacune de ces lignes est importante, alors il faut assurer l'ajout de chacune dans le code de notre application :
  glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

- **GlTexEnvf.**

  L'appel de **glTexEnvf** permet d'initialiser des variables d'environnement pour la texture actuelle. Cet appel spécifie pour *OpenGL* comment la texture agira quand il est rendu dans une scène. Voici un exemple de cet appel :
  glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

  Celui-ci initialise la texture active à GL_MODULATE. L'attribut GL_MODULATE permet d'appliquer des effets comme l'éclairage et la coloration à notre texture. Si nous ne voulons pas éclairer et colorer notre texture, et nous voudrons afficher la texture inchangée quand la coloration est appliquée il faut remplacer GL_MODULATE avec GL_DECAL.

- **glTexImage2D.**

  L'appel de **glTexImage2D** est le but visé. Cet appel chargera la texture à la mémoire vidéo où elle sera prêt pour être employer dans notre programme. On va expliquer les paramètres d'appel de cette fonction un par un puisque c'est important pour ce que nous somme entrain de faire.

  - *target* : la cible de cet appel (le type de notre texture), il sera toujours GL_TEXTURE_2D pour notre cas.
  - *level* : le niveau de détail.
  - *internalformat* : Avec ce paramètre on peut spécifier pour *OpenGL* le nombre de composants de chaque pixel de la texture qui doit être chargée. Parmi les valeurs affectées à ce constant on peut citer : GL_RGB qui est égal à 3 (rouge, vert et bleu) et GL_RGBA qui est égal à 4 (rouge, vert, bleu et alpha spécifiant la transparence).
  - *width & height* : la largeur et hauteur de l'image à utiliser comme texture. Ceux-ci doivent être une puissance de deux (2, 4, 8, 16, 32, 64, 128, 256, 512, …etc).
  - *border* : la frontière de l'image, doit être 0 ou 1. Nous employons toujours 0 dans notre code puisque nous n'utilisons pas de frontières d'image.

- *format* : le format des données de pixel qui seront chargées (GL_RGB ou GL_RGBA).
- *type* : le type des données qui seront chargées. Nous utilisons la valeur GL_UNSIGNED_BYTE.
- *pixels* : Pointeur vers les données d'image. Ce sont les données de l'image qui seront chargées à la mémoire vidéo. Après notre appel à **glTexImage2D** nous pouvons libérer cette espace mémoire puisque la texture est déjà chargée dans la mémoire vidéo.

  Voici un exemple d'appel à **glTexImage2D** :
  GlTexImage2D (GL_TEXTURE_2D, 0, GL_RGB, imageWidth, imageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, imageData);

Après avoir effectuer tout ces appels de fonction *OpenGL* la texture sera chargée et prêt à être appliquée à notre géométrie. La partie suivante discutera l'opération d'application de la texture à la géométrie.

3. Utilisation de la texture.

Le processus à suivre pour appliquer une texture à la géométrie dépend du type de données traitées. En raison de ce fait, nous allons présenter dans cette partie quelques indicateurs sur la placage de texture, la méthode de placage de la texture sur un quadrant et le système de coordonnées texture [MIL00].

Indicateurs :

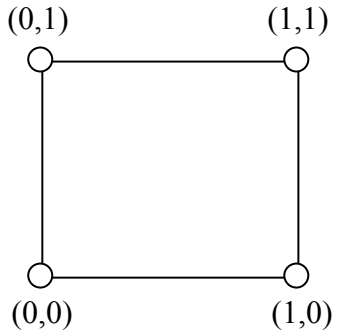
1) Il faut assurer l'activation du placage de texture. Pour cela, on fait appel à **glEnable**(GL_TEXTURE_2D).
2) Il faut assurer que nous avons spécifie la texture à utiliser avant que de faire appel à **glBegin**/**glEnd**. Nous ne pouvons pas lier une texture à utiliser au milieu d'une paire **glBegin**/**glEnd**.
3) Il faut assurer la spécification d'une coordonnée de texture avant de spécifier le sommet d'une face. Si nous avons 3 sommets, le modèle utilisé pour le placage d'une texture sur le triangle sera le suivant : TexCoord; VertexCoord; TexCoord; VertexCoord; TexCoord; VertexCoord;
4) Assurons-nous que nous avons stocké nos identificateurs des textures dans des variables. **glGenTextures** est la façon la plus facile d'obtenir un "id" une texture libre.

Un quadrant texturé :

Nous présentons ci-dessous un exemple de la manière de texturation d'un quadrant. Ce code suppose qu'on a activé la texturation et qu'il y a eu une texture chargée avec l'id de 13.

```
glBindTexture (GL_TEXTURE_2D, 13);
glBegin (GL_QUADS);
glTexCoord2f (0.0, 0.0);
glVertex3f (0.0, 0.0, 0.0);
glTexCoord2f (1.0, 0.0);
glVertex3f (10.0, 0.0, 0.0);
glTexCoord2f (1.0, 1.0);
glVertex3f (10.0, 10.0, 0.0);
glTexCoord2f (0.0, 1.0);
glVertex3f (0.0, 10.0, 0.0);
glEnd ();
```

Le système de coordonnée texture :



L'image ci-dessus représente le système de coordonnée de texture *OpenGL*. Dans le code au-dessus des appels à **glTexCoord2f** sont très important, car l'ordre affecte le résultat final du placage de texture. Quand on fait un appel à **glTexCoord2f** (x, y), *OpenGL* place la coordonnée de la texture de cette emplacement sur l'image. Si on veut texturer un triangle, il y aura trois textures coordonnées sur l'image. Une fois qu'un **glEnd** est atteint, le triangle formé par les coordonnées de la texture est alors plaqué sur le triangle composé des sommets de la facette [MIL00].

4. Conclusion.

Cette partie récapitule le processus de placage de texture en utilisant OpenGL. Ce processus passe par deux étapes principales qui sont le chargement de la texture dans la mémoire vidéo et par la suite l'affichage de chaque texture comme étant un polygone transparent texturé.

Bibliographie

[ANT01] Antoine Miné, « Texture procédurales en temps réel avec OpenGL », Rapport de stage, laboratoire iMAGIS,16 juillet 2001.

[BAA01] J. van Baar, M. Zwicker, H. Pfister et M. Gross. « Surface splatting In Computer Graphics », (SIGGRAPH ’01 Proceedings), Août 2001.

[CHE95] Shenchang Eric Chen, « QuickTimeVR – An Image-Based Approach to Virtual Environment Navigation », Rapport d’activité, Apple Computer Inc, 1995.

[CHR98] Stéphane Christy, « Localisation et modélisation tridimensionnelles par approximations successives du modèle perspectif de caméra », Thèse de doctorat, L’Institut National Polytechnique de Grenoble, 17 septembre 1998.

[DEB04] Gilles Debunne, « Rendu à base d’images », DEA IVR, 2004.

[DEC02] Xavier Décoret, Frédo Durand, François X. Sillion et Julie Dorsey, « Billboard Clouds », Rapport d’activité INRIA, juin 2002.

[GOR96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski et Michael F. Cohen, « The Lumigraph », Rapport d’activité, *SIGGRAPH*, 1996.

[GUI00] Erwan Guillou, « Simulation d'environnements complexes non lambertiens à partir d'images : Application à la réalité augmentée », Thèse de doctorat, Université de Rennes 1, Décembre 2000.

[HIN01] Damien Hinsinger, Fabrice Neyret et Marie-Paule Cani, « Interactive Animation of Ocean Waves », Rapport d’activité, iMAGIS, 2001.

[HQU00] Huamin Qu, « Sample-based Rendering », Rapport d’activité, Department of Computer Science, State University of New York at Stony Brook, 2000.

[JAG03] Martin Jagersand et al., « Recent Methods for Image-based Modeling and Rendering », Tutorial 1 de réalité virtuelle, IEEE, Mars 2003.

[KAL01] Aravind Kalaih et Amitabh Varshney, « Differential point rendering », Proceedings of 12th Eurographics Workshop on Rendering, pages 139–150, Juin 2001.

[KEN97] Kenneth E. Hoff III, « Reviewer of a Cellular Texture Basis Function », Computer Graphics Research : Procedural Texturing, 11 Avril 1997.

[KRU96] Mike Krus, Françoise Guisnel, Patrick Bourdot et Guillaume Thibault, « Niveaux de Détails et Simplification Polygonaux : un tour d'horizon.», AFIG'96, Quatrièmes Journées de l'Association Française d'Informatique Graphique, 1996.

[KRU97] Mike Krus, « Maillages Polygonaux et Niveaux de Détails étude bibliographique », Rapport Technique, Electricité de France, Direction des Etudes et Recherches, Mai 1997.

[LAC94] Philippe Lacroute et Marc Levoy, « Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation », Rapport d'activité, SIGGRAPH *'94* - Orlando - Florida, Juillet 1994.

[LEB00] Luc Leblanc, « Construction et utilisation des bloqueurs pour l’accélération des requêtes de visibilité », Mémoire de Maître ès sciences, Université de Montréal, Août 2000.

[LEF01] Sylvain Lefebvre, « la simulation visuellement réaliste de surfaces à base de déchirures », stage de DEA, 2001.

[LEO97] Leonard McMillan, « An image-based approach to three-dimensional computer graphics », Thèse de doctorat, Université de North Caroline – Chapel Hill, 1997.

[LIS98] Dani Lischinski Ari, « Image-Based Rendering for Non-Diffuse Synthetic Scenes », Rapport d'activité, The Hebrew University, 1998.

[LUE01] David P. Luebke, « A Developer's Survey of Polygonal Simplification Algorithms », Université de Virginia, May/Juin 2001.

[MEY98] Alexandre Meyer, « Textures volumiques interactives », Rapport de D.E.A., Responsable : F. Neyret, 18 juin 1998.

[MEY01] Alexandre Meyer, « Représentations d'arbres réalistes et efficaces pour la synthèse d'images de paysages », Thèse de Phd, Université Joseph Fourier, 10 décembre 2001.

[MIL00] Nate Miller, « OpenGL Texture Mapping: An Introduction », Cours sur OpenGL, 29 Février 2000, http://www.gamedev.net/reference/programming/features/ogltm/

[MIN01] Antoine Miné, « Textures procédurales en temps réel avec OpenGL », Rapport de stage de fin de 1ère année de Magistère, Laboratoire iMAGIS, 16 Juillet 2001.

[MOR93] Luce Morin, « Quelques contributions des invariants projectifs à la vision par ordinateur », Thèse de doctorat préparée au sein du laboratoire LIFIA-IRIMAG, L'Institut National Polytechnique de Grenoble, 12 Janvier 1993.

[NEY96] Fabrice Neyret, « Textures volumiques pour la synthèse d'images », Thèse de doctorat, Université Paris XI Orsay, Juin 1996.

[NEY97] Fabrice Neyret, « Qualitative Simulation of Convective Cloud Formation and Evolution », Rapport d'activité, Septembre 1997.

[NEY98] Fabrice Neyret, « Modeling, Animating and Rendering Complex Scenes Using Volumetric Textures », IEEE Transaction On Visualisation and Computer graphics, vol.4, NO 1, Janvier/Mars 1998.

[Ney00] Fabrice Neyret, « A phenomenological shader for the rendering of cumulus clouds », Rapport technique, INRIA, May 2000.

[OLI00] Manuel M. Oliveira, « Image-Based Modeling and Rendering Techniques: A Survey », Rapport d'activité, Institut d'informatique, Porto Alegre, RS, Brasil, 2000.

[PAR02] Rick Parent, « Computer Animation Algorithms and Techniques », Edition MORGAN KAUFMANN PUBLISHERS, 2002.

[PAU01] M. Pauly and M. Gross « Spectral processing of point-sampled geometry », SIGGRAPH Computer Graphics Proceedings, Août 2001.

[PER98] B. Péroche et al. « Informatique graphique : Méthodes et Modèles », 2ème Edition revue et augmentée, Edition Hermès, 1998.

[PFI00] H. Pfister, M. Zwicker, J. van Baar, et M. Gross, « Surfels : Surface elements as rendering primitives », SIGGRAPH 2000, Computer Graphics Proceedings, 2000.

[PIE99] Dan Stora Pierre, Olivier Agliati Marie-Paule Cani, Fabrice Neyret et Jean-Dominique Gascuel, « Animating Lava Flows », Rapport d'activité, IMAGIS, Juin 1999.

[POR04] Damien PORQUET, « Rendu en temps réel de scène complexe », Thèse de doctorat, 29 Novembre 2004. www.msi.unilim.fr_porquet\memoire\memoire-html.html

[RUS00] S. Rusinkiewicz et M. Levoy, « QSplat : A multiresolution point rendering system for large meshes », SIGGRAPH 2000, Computer Graphics Proceedings, 2000.

[RUS01] Holly Rushmeier, Laurent Balmelli et Fausto Bernardini, « Horizon Map Capture », Rapport d'activité, IBM T. J. Watson Research Center y, EUROGRAPHICS, 2001.

[SCH03] Tino Schwarze, « Kommunikationsmechanismen für paralleles, adaptives Level-of-Detail in VR-Simulationen », Thèse de doctorat, Université technique de Chemnitz, 5 Mars 2003.

[SEI97] Steven Maxwell Seitz, « Image-based transformation of viewpoint and scene appearance », Thèse de doctorat, Université de Wisconsin – Madison, 1997.

[SHA97] Jonathan Shade, Steven Gortler, Li-wei Hey et Richard Szeliski, « Layered Depth Images », Rapport d'activité, SIGGRAPH, 1997.

[STA01] M. Stamminger et G. Drettakis « Interactive sampling and rendering for complex and procedural geometry », Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering 01), Eurographics, 2001.

[STU97] Peter Franz Sturm, « Vision 3D non calibrée : contributions à la reconstruction projective et étude des mouvements critiques pour l'auto-calibrage », Thèse de doctorat, L'Institut National Polytechnique de Grenoble, 17 Décembre 1997.

[VAN04] Koen Erik Adriaan van de Sande, « Absolute localization using a pseudo-randomly colored pattern », Rapport d'activité, Université Van Amsterdam, 6 Juillet 2004.

[VIN98] Vincent FURMINIEUX et Matthieu AUDOIN « LOD et Progressive Meshes », document PowerPoint, 1998.

Résumé :

Les textures volumiques offrent un rendu réaliste des scènes complexes répétitives, en utilisant le lancer de rayons, dans des temps acceptables (de 5 à 20 minutes). Cependant, pour des applications exigeant le temps réel, l'obtention d'un gain supplémentaire parait difficilement envisageable en conservant cette technique sous sa forme initiale (représentation du volume de référence par un arbre octale).

Parmi les approches de rendu à base d'image, traitant les problèmes de la complexité géométrique et temporelle, se présente celle de rendu à base de couches d'images. Cette technique permet un rendu réaliste de scènes et d'objets à un moindre coût en calcul qu'avec une méthode traditionnelle. Ce gain en temps de calcul est dû à l'exploitation des capacités des cartes graphiques traitant, efficacement, les polygones. De ce fait, chaque couche d'image doit être représentée par un polygone texturé.

Pour le rendu, il suffit de projeter et de composer successivement les couches sur le plan image (Z-Buffer) et obtenir, ainsi, l'image finale. Chaque couche projetée est combinée avec le résultat précédent en tenant compte de la densité de chaque pixel.

La projection d'une couche est plus rapide que les calculs de projection pour chaque voxel le long d'un rayon, ce qui permet un gain de temps appréciable.

Mots clés :

Synthèse d'images, rendu à base d'images, couches d'images, niveaux de détails, textures volumiques, réalisme, visibilité.

ملـــــخص :

تسمح تقنية المظهر الخارجي الحجمي بالحصول على صور حقيقية لمناظر مركبة تتوفر على عناصر متكررة بإرسال أشعة انطلاقا من نقطة نظر معينة، وهذا في أوقات جد مقبولة (من 5 إلى 20 دقيقة).

غير أنه بالنسبة للتطبيقات التي تستوجب الحصول على نتائج فورية في وقت حقيقي، فإنه يبدو من الصعب التطلع للحصول على ربح إضافي للوقت اللازم لإنشاء الصورة النهائية إذا ما أبقينا هذه التقنية بالوجه الذي هي عليه حاليا (تمثيل الحجم المرجعي كشجرة ثمانية).

من بين العديد من تقنيات الإنشاء التي تعتمد على الصور من أجل معالجة مشاكل التعقيدات الهندسية وكذا تلك المتعلقة بالوقت، نجد تقنية الإنشاء التي تستخدم الصور على شاكلة طبقات. هذه الأخيرة تسمح بإنشاء صور حقيقية لمناظر أو أجسام معينة في أقل الأوقات وهذا مقارنة بالتقنيات التقليدية. إن هذا الربح في الوقت راجع تحديدا إلى استغلال القدرات التي تتمتع بها محولات العرض الحالية، كونها تعالج وتعرض المضلعات بجدارة عالية. لأجل هذا نستطيع أن نعتبر كل طبقة من الصور كمضلع مكسو.

من أجل عرض النتيجة النهائية، يكفي إسقاط ومزج الطبقات بشكل متتابع للحصول على الصورة النهائية. يتم تركيب كل طبقة معروضة مع النتيجة السابقة، آخذين بعين الاعتبار كثافة (أو شفافية) كل نقطة من الصورة.

إن عرض طبقة ما يكون أسرع بكثير من حساب مسقط كل عنصر حجمي (ضمن الشجرة الثمانية) بالنسبة لكل شعاع، مما يسمح لنا بربح معتبر في الوقت اللازم لعرض النتيجة.

الكلمـــات المفاتيح :

إنشاء الصور، العرض باستخدام الصور، طبقات الصور، مستويات التفصيل، المظهر الخارجي الحجمي، الواقعية، الرؤية.