50376
1996
71

USTL

LIFL Laboratoire d'Informatique Fondamentale de Lille

N° d'ordre : 1686

THESE

Nouveau Régime

Présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

Pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

Alain PREUX

Antialiassage en synthèse d'images, état de l'art et proposition de méthodes temps réel pour les bords de polygones

Thèse soutenue le 19 Janvier 1996, devant la Commission d'Examen :

Président	:	Vincent CORDONNIER	Professeur	Université de LILLE I
Rapporteurs	:	Didier ARQUÈS	Professeur	Université de Marne la Vallée
	:	Djamchid GHAZANFARPOUR	Professeur	E.N.S.I.L. (Limoges)
Examinateurs	:	Christophe CHAILLOU	MdC	Université de LILLE I
	:	Michel MÉRIAUX	Professeur	Université de Poitiers
	:	Bernard PÉROCHE	Professeur	E.M.S.E. (Saint-Étienne)

SCD LILLE 1
D 030 305646 2

ga 2-107031

Table des matières

UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE
BIBLIOTHÈQUE UNIVERSITAIRE

Chapitre 1

Introduction

L'infographie est un domaine en pleine expansion, ses champs d'actions sont aussi divers que variés et leur nombre ne cesse de s'accroître. La puissance de calcul des machines, elle aussi sans cesse grandissante, permet de développer les techniques de synthèse d'images dans deux directions : l'augmentation de la complexité des scènes et l'amélioration de la qualité des images. Ces deux thèmes ont en fait un but commun qui est l'augmentation du réalisme. Le terme très à la mode de « *Réalité Virtuelle* » illustre parfaitement cette quête du Graal que se sont fixés certains chercheurs : la qualité de l'image calculée doit être suffisante pour faire oublier le support de restitution. Nous sommes encore très loin de ce résultat, mais l'amélioration de la qualité des images reste un enjeu important. Les champs d'application de la synthèse d'images peuvent se résumer en deux catégories :

- Les applications réclamant des images très réalistes qui cherchent à modéliser des éclairements complexes (ombres portées, phénomènes de réflexion et de réfraction lumineuse,...). Ce type d'application utilisent en général des algorithmes de lancer de rayons et/ou de radiosité, qui produisent des images d'un grand réalisme mais qui sont encore très couteuses en temps de calcul.
- Les applications interactives pour lesquelles la vitesse d'affichage (et donc de calcul) est plus importante que le réalisme des images. Ce sont alors des techniques de rendu projectif, associées à des modèles d'éclairement plus simples, qui sont utilisées. De nombreux constructeurs proposent des machines spécialisées permettant un affichage en temps réel de ce type de scènes. C'est dans ce cadre que se situe la base de notre travail.

Quelles que soient les techniques utilisées, à cause du caractère numérique et discret des ordinateurs, la synthèse d'images souffre de défauts que nous appelons *aliassage*. Ces phénomènes, qui sont essentiellement dûs au processus d'échantillonnage, sont bien connus en traitement de signal. Après quelques rappels théoriques dans ce domaine, nous cherchons à mettre en évidence les causes de ces phénomènes qui sont de deux ordres : le repliement de spectre et le crénelage introduit par la reconstruction du signal. Une bonne compréhension des mécanismes mis en jeu nous permet de mieux analyser ce qui se passe en synthèse d'images, en particulier en différenciant plusieurs types de défauts trop souvent confondus dans le même terme.

Les manifestations de l'aliassage en synthèse d'images sont nombreuses et variées. Nous faisons une étude complète de ces phénomènes en expliquant leur origine et en donnant les

diverses solutions. Nous parcourons donc les différents champs de la synthèse d'images, des méthodes de rendu à l'animation, en passant par les modèles d'éclairement. En effet, bien que l'aliassage en informatique graphique soit souvent résumé aux *moirés* et aux *marches d'escalier*, beaucoup d'autres *artefacts* sont générés à différentes étapes. Nous nous intéressons à ces différentes formes en prenant en compte les spécificités du contexte. Les origines pouvant être différentes, chaque cas est étudié séparément des autres.

Le problème des marches d'escalier sur les bords d'objets peut être résolu soit en sur-échantillonnant, soit en utilisant des techniques d'échantillonnage surfacique. Cette deuxième solution semble intéressante car elle ne contraint pas le système à faire chuter ses performances. De nombreuses méthodes de ce type ont été publiées : des techniques de composition d'images aux méthodes à masques en passant par les algorithmes à voisins. Nous en proposons une classification comparative.

Dans le cadre des méthodes projectives pour la radiosité, on observe certains phénomènes qui sont également appelés aliassage. Après avoir analysé ces défauts, nous avons eu l'idée d'utiliser des techniques développées dans un autre contexte. Ces méthodes se sont révélées très bien adaptées et efficaces sur cette forme particulière d'aliassage.

L'augmentation de la puissance des machines bon marché, incite les infographistes à créer des scènes de plus en plus complexes et avec un niveau de détail grandissant. Il est fréquent, et le phénomène ne va faire que s'accroitre, de trouver des scènes contenant beaucoup de polygones de taille inférieure au pixel. Ces facettes créent dans l'image de très hautes fréquences et sont la cause de graves défauts d'aliassage quand des algorithmes classiques sont utilisés.

Cet argument est à mettre en opposition avec le développement de la technologie des écrans (simultanément à l'évolution des processeurs) qui nous propose des moniteurs de résolution de plus en plus fine, ce qui a bien sûr pour effet l'atténuation naturelle de l'aliassage.

Il faut cependant bien comprendre que cette augmentation du nombre de pixels multiplie d'autant le coût de calcul d'une image. Il est clair que la puissance de calcul n'est pas encore suffisante pour augmenter à la fois la complexité des scènes et la résolution des images, à moins d'un compromis.

Les utilisateurs (les constructeurs?) devront donc choisir entre des images de très bonne qualité représentant des scènes simples et des images de moins bonne qualité représentant des scènes plus complexes.

Cette alternative est bien évidemment déjà présente avec les machines proposant un sur-échantillonnage. Nous pensons que dans ce cadre, des algorithmes améliorant sensiblement la qualité de l'image, en ne pénalisant pas trop le temps de calcul, sont une alternative intéressante. C'est dans cette direction que nous avons mené nos travaux, notre idée de départ étant de combiner les avantages des algorithmes d'échantillonnage surfacique avec la simplicité du Z-Buffer. Pour cela, nous choisissons de remplacer la multiplication des calculs du sur-échantillonnage par la duplication de la mémoire d'image. Cette proposition semble être bien adaptée aux évolutions technologiques actuelles, nous pensons en particulier aux nouvelles mémoires *3DRam*.

Plusieurs solutions ont été envisagées, certaines se sont révélées inefficaces, d'autres donnent des résultats très satisfaisants. Dans tous les cas, c'est un compromis entre la qualité et la complexité (essentiellement en mémoire) qui est proposé et non une méthode exacte.

Première partie

Notions de base

Chapitre 2

La synthèse d'images

2.1 Introduction

La synthèse d'images 3D se décompose en deux étapes distinctes qui sont la modélisation et le rendu. La modélisation consiste à représenter une scène à l'aide de primitives géométriques complexes. La création d'une scène se fait en représentant les différents objets qui la constituent et en définissant leur position par rapport à un repère que l'on nomme le repère de la scène. On attribue ensuite à chaque objet des caractéristiques propres telles que la couleur, la texture ou encore des coefficients représentant leur aspect (brillance, rugosité,...).

Une scène doit également comprendre des sources lumineuses qui serviront au calcul de l'éclairement. Ces sources sont également positionnées dans l'espace des objets, leur influence sera déterminée par leurs propriétés (intensité, couleur, direction).

La conception de scènes (ou modélisation) en infographie fait partie de la conception assistée par ordinateur (C.A.O.), qui utilise le plus souvent une représentation *fil de fer*. Cette technique d'affichage est très peu réaliste mais possède l'avantage considérable de permettre une visualisation rapide et interactive.

Pour l'étape de rendu, il est nécessaire de désigner un point comme étant la position de l'observateur (ou d'une caméra) dans la scène afin de calculer un *point de vue*. Le calcul des couleurs, des réflexions de lumières et l'élimination des parties cachées se fait par rapport à ce point de vue.

Il n'est pas possible dans le cadre de ce travail de présenter dans le détail les différentes étapes de la synthèse d'image, il est évident que nous considérons le lecteur comme un spécialiste du domaine. Cependant il nous a paru important de survoler l'ensemble du processus de création d'une image de synthèse de manière à pouvoir définir précisément où se situe notre travail. De plus, nous détaillons un peu plus finement certaines notions qui nous seront nécessaires tout au long de ce manuscrit.

De nombreux algorithmes sont cités sans être référencés, ils ont été largement décrits dans des ouvrages généraux, on les trouve également dans tout bon cours universitaire de second cycle.

2.2 La modélisation

Modéliser une scène consiste à la représenter sous forme d'une description numérique. Cette description doit comporter des informations géométriques et photométriques. Celles-ci

peuvent provenir de capteurs dans le cas de reproduction de scènes réelles, de calculs dans le cas de simulations, ou de l'imagination du concepteur qui élabore des scènes *qui n'existent pas.*

La plupart des modeleurs proposent des primitives de modélisation de haut niveau telles que des objets volumiques, des surfaces paramétriques ou implicites, etc... En général, il n'est pas possible de représenter exactement un objet, on utilise plutôt une approximation (un modèle) plus ou moins précise selon l'usage, à l'aide de ces primitives de modélisation.

Dans tous les cas, si ces scènes sont destinées à être visualisées par un algorithme de rendu, elles doivent être converties en primitives *affichables.* Ces primitives d'affichage sont différentes selon le type d'algorithme de rendu (*cf.* ci-dessous), mais elles sont toujours plus simples que les primitives de modélisation. Une nouvelle étape d'approximation est donc nécessaire (par exemple la facettisation).

Il est également nécessaire de définir les sources lumineuses présentes dans la scène, en spécifiant leur emplacement, leur intensité et leur direction. Enfin, les objets décrits géométriquement doivent posséder des caractéristiques colorimétriques et/ou de textures afin de calculer leur couleur en chaque point de l'écran au moment du rendu.

2.3 Le rendu

On entend par rendu le procédé qui consiste à afficher une scène de manière plus ou moins réaliste. Après avoir effectué un changement de repère (du repère de la scène vers celui de l'observateur), il reste deux traitements distincts à réaliser :

- déterminer les objets visibles en chaque pixel de l'écran. Cette étape se décompose elle-même en deux questions simples auxquelles il faut répondre pour chaque pixel :
  - *quels objets se projettent ?*
  - *lequel est visible ?*
- calculer la couleur de ces objets en fonction de leurs caractéristiques propres et de l'environnement.

Pour effectuer le premier traitement, que l'on appelle la conversion des objets en pixels et l'élimination des parties cachées, il existe deux grandes catégories d'algorithmes :

- le tracé de rayons (*ray casting*),
- la projection perspective.

Pour le calcul de la couleur, plusieurs modèles d'illumination peuvent être utilisés, nous en présentons quelques uns.

Préliminaires

Avant de présenter la suite, nous tenons à mettre au clair le vocabulaire utilisé qui, une fois de plus, est ambigu à cause de mauvaises traductions des termes initialement définis en anglais. Il existe dans cette langue deux termes distincts qui sont *ray casting* et *ray tracing.* Le premier représente le fait de calculer l'intersection entre une droite (le rayon) et un objet. Il est principalement utilisé pour effectuer l'élimination des parties cachées. Le second terme

englobe ce que l'on a l'habitude d'appeler le parcours de la lumière, et correspond en fait à un algorithme de calcul d'éclairement tenant compte des rélexions multiples. Tout irait bien si le *ray casting* était traduit par *lancer de rayons* qui est une bonne traduction littérale et *ray tracing* par *suivi de rayons*. Malheureusement, le *ray casting* est le plus souvent traduit par *tracé de rayons*, alors que le *ray tracing* prend le plus souvent la forme de *lancer de rayons*.

Par ailleurs, on a l'habitude de parler du *lancer de rayons* comme technique de rendu qui comprend à la fois l'algorithme de *ray tracing* pour les calculs d'éclairement et l'utilisation des rayons primaires pour l'élimination des parties cachées.

Afin d'éviter toute confusion, nous utiliserons dans la suite de ce mémoire l'expression *lancer de rayons* pour parler de l'algorithme général, et nous employerons la terminologie anglo-saxonne lorsque nous voudrons différencier les deux concepts.

2.3.1 Le *ray casting*

Le *ray casting* est certainement l'algorithme le plus naturel pour déterminer la visibilité des objets en chaque pixel. Des *"rayons"*, passant par le centre de chaque pixel, sont lancés depuis un point correspondant à l'œil de l'observateur. Pour chacun des objets de la scène, on calcule son intersection avec les rayons : l'objet dont l'intersection est la plus proche est l'objet visible. La perspective est ainsi naturellement prise en compte par ce modèle.

Pour le calcul de la couleur en chaque pixel, on peut *suivre* le rayon à travers ses réflexions multiples dans la scène en suivant le parcours inverse de la lumière. C'est ce qu'a proposé Whitted dans [Whitted80] et que l'on appelle le *ray tracing*, nous le présentons plus loin (*cf.* §2.4.2).

Cet algorithme présente, contrairement au rendu projectif que nous décrivons ci-après, l'avantage de pouvoir afficher directement n'importe quelle primitive géométrique, si complexe soit-elle, pourvu qu'on sache calculer son intersection avec une droite.

Par contre, c'est un algorithme *orienté pixel*, c'est-à-dire qu'il traite tous les objets pour chaque pixel et non pas le contraire. L'inconvénient de cette méthode est que l'éclairement doit être recalculé entièrement pour chaque pixel sans pouvoir réutiliser les valeurs calculées dans les pixels voisins. En particulier aucune interpolation n'est possible avec le *ray casting*. Cette caractéristique n'est un inconvénient qu'en terme de temps de calcul, elle permet en fait d'effectuer en chaque pixel un calcul plus exact qu'avec une méthode utilisant des interpolations.

2.3.2 Le rendu projectif

Le rendu projectif est l'alternative au *ray casting*. Après avoir initialement défini les objets dans un repère global, on effectue un changement de repère pour passer dans celui de l'observateur. Les objets 3D sont ensuite convertis en pixels par projection perspective sur l'écran cependant que leur coordonnée Z (la profondeur) est calculée (le plus souvent par interpolation). L'élimination des parties cachées est effectuée soit par Z-buffer, soit par un tri préalable des primitives d'affichage dans l'espace objet, comme par exemple dans l'algorithme dit : *du peintre*. Il existe d'autres algorithmes que nous ne décrivons pas ici.

La facettisation

Contrairement au *ray casting* où la visibilité d'un objet dans un pixel est déterminée par l'intersection de la primitive avec une droite, il faut, en rendu projectif, *projeter* les objets sur

l'écran. C'est un algorithme qui cette fois est *orienté objet* : pour chaque objet on détermine tous les pixels qui sont concernés. Cela suppose donc que l'on sait déterminer le contour de chaque primitive. En fait, cette opération est très difficile pour les objets géométriques complexes. Dans la pratique, toutes les primitives de modélisation sont *facettisées* (*i.e.* découpées en facettes planes). Les polygones (le plus souvent à trois ou quatre côtés) sont actuellement les seules primitives d'affichage utilisées en rendu projectif.

Cette étape a pour conséquence la déformation des contours. Les objets réels étant approchés par la juxtaposition de facettes planes, les contours se retrouvent *segmentés*. On peut observer ce phénomène sur la sphère facettisée présentée à la *figure 2.5*.

La facettisation permet d'une part de ne stocker les informations géométriques qu'aux sommets des polygones, et d'autre part d'effectuer le rendu par interpolation. Cette méthode est très rapide, et sa simplicité a permis de l'implémenter matériellement dans de nombreuses stations de travail spécialisées pour le graphique 3D. C'est d'ailleurs la seule méthode de rendu qui soit à l'heure actuelle câblée.

Le Z-Buffer

Avec le rendu projectif de scènes polygonales, l'élimination des parties cachées est le plus souvent réalisée par l'algorithme du Z-buffer. Le principe est très simple : pour chaque pixel où l'objet courant est présent, on compare la profondeur de cet objet avec celle déjà stockée. Si elle est plus petite alors la profondeur et la couleur de l'objet courant remplacent les anciennes valeurs.

Le Z-buffer, d'une grande simplicité algorithmique, est très gourmand en mémoire puisqu'il nécessite de stocker en plus de la mémoire d'image (mémoire de trame), une information de profondeur en chaque pixel. Cependant, sa simplicité l'a rendu hégémonique en rendu projectif et beaucoup de constructeurs proposent des machines graphiques avec Z-buffer câblé. Nous verrons par la suite qu'il est mal adapté à l'antialiassage.

Affichage par suivi de contours

Les algorithmes d'affichage par suivi de contours (également appelés remplissage par contour/segment) sont basés sur un algorithme de tracé de segment incrémental de type Bresenham ou DDA [1] afin de déterminer les contours du polygone, le remplissage s'effectue ensuite en reliant horizontalement des pixels de bord opposés (*cf.* figure 2.1 (a)).

1. DDA sont les initiales de l'expression anglo-saxonne : Digital Differencial Analyzer

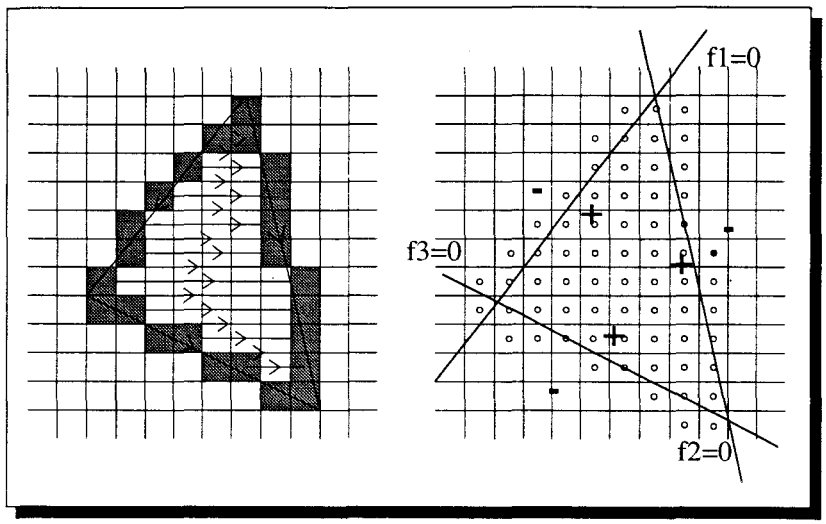


FIG. 2.1 - *Remplissage de polygone par (a)contour/segment (b)test d'inclusion*

Remplissage par test d'inclusion

- Méthode de la boîte englobante
  Une autre méthode d'affichage et de remplissage de polygones consiste à tester pour chacun des pixels, si il se trouve à l'intérieur ou à l'extérieur de la primitive. Pour cela, on utilise les équations des droites délimitant le polygone. En normalisant les signes des coefficients de ces équations, on obtient des relations qui retournent un résultat positif lorsque le point se trouve à l'intérieur du polygone et négatif pour au moins l'une des trois droites quand le point est à l'extérieur (*cf. figure 2.1 (b)*). Cette technique, courante dans les implémentations matérielles de l'algorithme de *pixelisation* (conversion des objets en pixels, en anglais *rasterisation*), présente en effet l'avantage de ne pas avoir à traiter certains problèmes classiques liés à la transformation perspective. En effet, dans les algorithmes de remplissage par contour-segment, la transformation des objets en pixels s'effectue (*cf.* ci-dessus) en traçant les segments-contour à partir des coordonnées des sommets projetés. De nombreuses erreurs de calcul introduites lors de cette étape sont à l'origine de défauts d'affichage (recouvrements, trous entre deux facettes, etc...). En représentant les objets par leurs équations de contour, il est possible de transformer directement les coefficients des équations pour obtenir les équations du polygone projeté, ce qui d'une part diminue les erreurs d'arrondi, et d'autre part évite de traiter comme des cas particuliers les polygones dont la projection d'un ou plusieurs sommets se retrouve en dehors des intervalles permis. Cette solution est donc idéale pour les implémentations matérielles, d'autant qu'il est possible de paralléliser le processus comme l'ont proposé les concepteurs du système Pixel-Plane ([Fuchs et al.85]). Afin de ne pas effectuer les tests d'inclusion sur l'ensemble des pixels de l'écran, il faut utiliser des boîtes englobantes.

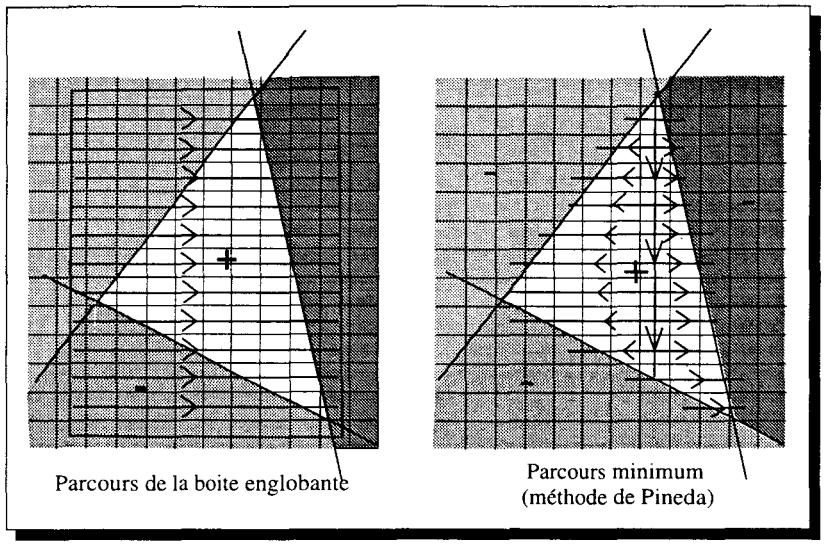


FIG. 2.2 - *Remplissage par test d'inclusion (a) sur la boite englogante (b) par la méthode de Pineda*

- Méthode de Pineda
  [Pineda88] a décrit une méthode utilisant quelques tests supplémentaires, mais permettant de ne tester que les pixels recouverts par le polygone traité. Cette technique s'apparente aux algorithmes de remplissage à germe utilisant la connexité des pixels à l'intérieur des polygones. À partir d'un point de départ intérieur au polygone qui sert de *point de retour* (par exemple le sommet le plus haut), l'algorithme parcourt les points situés à droite jusqu'à la rencontre avec un segment-contour, revient au point de retour, recommence à gauche le même parcours, puis réitère le processus avec la ligne directement inférieure. Le signe des fonctions des arêtes permettent de stopper les progressions latérales et de retourner au point de retour de la ligne courante. La *figure 2.2 (b)* montre un exemple de parcours de triangle avec cet algorithme.

2.4 Les modèles d'illumination

Un autre aspect de la synthèse d'images est lié aux conditions d'éclairement de la scène que l'on cherche à modéliser. En effet, dans la réalité, les objets ont une couleur propre que l'on pourrait mesurer en lumière blanche, mais lorsque nous les regardons, nous percevons des couleurs très différentes selon les conditions d'éclairage. Ces variations, qui peuvent être très importantes, dépendent des sources lumineuses, de leur nombre, de leur emplacement, de leur intensité, de leurs caractéristiques spectrales et de la position de l'observateur par rapport à elles. Les méthodes que nous présentons cherchent à modéliser les échanges de flux lumineux dans la scène, afin d'en déduire la couleur des objets visibles.

Il existe deux catégories de modèles d'illumination : les modèles locaux, empiriques mais offrant un bon compromis entre le réalisme et les temps de calcul, et les modèles globaux qui donnent d'excellents résultats mais qui sont très gourmands en calcul.

2.4.1 Les modèles locaux

Pour rendre compte des réflexions spéculaires et diffuses, les modèles d'éclairement local doivent tenir compte des caractéristiques des objets, et dans le cas des surfaces réfléchissantes, de la position des sources lumineuses et de l'observateur. Plusieurs modèles empiriques ont été proposés pour tenir compte d'une composante *ambiante*, des réflexions *diffuses* (dépendantes de l'angle d'incidence des rayons lumineux sur la surface, au point considéré) et des réflexions *spéculaires* (dépendantes de la position de l'observateur par rapport aux rayons réfléchis). Tous ces modèles sont de la forme:

$$I_\lambda(x) = I_{a_\lambda}(x) + I_{d_\lambda}(x) + I_{s_\lambda}(x)$$

où $I_{a_\lambda}(x)$, $I_{d_\lambda}(x)$ et $I_{s_\lambda}(x)$ représentent respectivement les intensités ambiante, diffuse et spéculaire pour la longueur d'onde λ. Leur somme donnant l'intensité totale pour cette même longueur d'onde au point x.

En général, pour des facilités de calcul, et bien que l'on sache que cela n'est pas suffisant, ce calcul d'intensité est réalisé pour trois longueur d'onde, le *rouge*, le *vert* et le *bleu*. Ce sont ces trois intensités qui sont utilisées pour reproduire les couleurs sur un téléviseur ou sur un écran d'ordinateur.

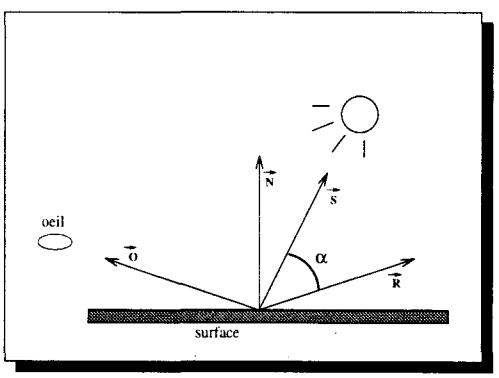


FIG. 2.3 - *Géométrie pour le modèle de réflexion de Phong*

Les intensités ambiantes et diffuses sont en général calculées sous la forme:

$$I_a = ka_\lambda \quad C_\lambda \qquad I_d = \sum_{i=1}^{p} Is_{i_\lambda} \quad kd_\lambda \quad C_\lambda \quad (\vec{S_i}.\vec{N})$$

avec

- ka_λ : le coefficient de réflexion de lumière ambiante de la surface,
- kd_λ : le coefficient de diffusion de la surface,
- Is_{i_λ} : l'intensité de la source i dans la longueur d'onde λ,
- C_λ : la composante pour la longueur d'onde λ de la couleur intrinsèque de l'objet,

et les vecteurs décrits sur la *figure 2.3.* Pour la composante spéculaire, il existe plusieurs modèles, celui de Phong est l'un des plus utilisés :

$$I_{s_\lambda} = \sum_{i=1}^{p} ks \quad C_\lambda \quad (\vec{R}.\vec{S_i})^n) = \sum_{i=1}^{p} ks \quad C_\lambda \quad \cos^n \alpha$$

avec

- k_s : le coefficient de réflexion spéculaire de la surface,
- n : un indice qui contrôle l'*étalement* de la tâche spéculaire.

2.4.2 Les modèles globaux

Dans un modèle local, les réflexions sont calculées localement pour chaque point de la surface. Ce type de modèle ne permet pas la prise en compte des réflexions multiples et nuit au réalisme des images calculées.

Par exemple, la localité des calculs spéculaires empêche de simuler des reflets dans un miroir. De même, la détection des ombres nécessite une prise en compte globale de l'éclairage.

En fait, l'énergie reçue par la surface d'un objet peut provenir directement ou indirectement des sources lumineuses (*via* une ou plusieurs réflexions sur d'autres surfaces). C'est ce que vont s'attacher à simuler les modèles globaux. L'énergie émise par une surface sera toujours la somme de son énergie propre (si c'est une source) et de la part de l'énergie reçue qu'elle réfléchit. Toutefois l'énergie reçue par une surface comprendra l'énergie reçue des autres surfaces réfléchissantes.

Le principe de base de ces modèles est la conservation de l'énergie. Au départ (après la modélisation de la scène), des sources sont définies, qui possèdent une énergie propre à émettre dans la scène. Un modèle global représente une modélisation de la distribution de cette énergie dans la scène.

L'équation de Kajiya (éq. 2.1), introduite en 1986, a permis d'unifier dans une théorie générale deux modèles complémentaires qui avaient été définis séparément: le *ray tracing* et la radiosité.

$$I(x,x') = g(x,x') \left[\epsilon(x,x') + \int_S \rho(x,x',x'') I(x',x'') dx'' \right] \tag{2.1}$$

avec

- $I(x,x')$: l'intensité véhiculée de x vers x',
- $g(x,x')$: la fonction de visibilité entre les points x et x' (0 s'il ne se voient pas, sinon g varie comme l'inverse du carré de la distance entre x et x'),
- $\epsilon(x,x')$: l'émittance propre transférée de x vers x',
- $\rho(x,x',x'')$: la réflectance bi-directionnelle au point x correspondant aux directions x' et x'',

l'intégrale est sur S, l'ensemble des points de toutes les surfaces de la scène.

Le *ray tracing*

Le *ray tracing* est un algorithme de rendu réaliste qui prend en compte les phénomènes lumineux comme la réflexion et la réfraction de manière naturelle (le principe étant le parcours inverse des rayons lumineux). Cet algorithme du lancer de rayons effectue les calculs d'éclairement en même temps que l'élimination des parties cachées. Il existe une littérature conséquente sur cette technique très populaire, citons par exemple [Glassner89], et plus proche de nous [Ris95]. Nous ne détaillons pas cette méthode ici, retenons simplement qu'elle permet un rendu très réaliste, en rendant compte des multiples réflexions spéculaires dans la scène.

La radiosité

À l'opposé du *ray tracing*, la radiosité qui est issue de la théorie des transferts radiatifs de chaleur, considère toutes les réflexions comme étant purement diffuses. Cette restriction rend la méthode indépendante du point de vue. L'éclairement d'une surface résulte alors de la fraction de l'énergie propre des sources qui atteint directement ou indirectement (*i.e.* après de multiples réflexions diffuses) la surface.

Une fois les calculs d'échanges d'énergie entre les objets de la scène terminés, l'affichage en fonction d'un point de vue peut s'effectuer soit par un rendu projectif avec interpolation de Gouraud, soit par un *ray tracing* pour permettre de cumuler les effets de chacune des méthodes, à savoir les réflexions diffuses et les réflexions spéculaires. Cette dernière possibilité est actuellement la technique qui offre, et de loin, le rendu le plus réaliste. Mais à quel prix...

2.5 Les systèmes de représentation des couleurs

Les modèles d'éclairement utilisés en synthèse d'images, qu'ils soient empiriques (exemple : Gouraud, Phong) ou dérivés des modèles physiques (comme les algorithmes de radiosité) n'ont qu'un seul but : modéliser (et surtout calculer) la couleur des objets d'une scène en différents points. Cette couleur dépend des caractéristiques intrinsèques des objets, des conditions d'éclairement et du point de vue pour les modèles complets (prenant en compte les réflexions spéculaires). L'étape de rendu d'une image consiste donc à déterminer pour chaque pixel la couleur de l'objet qui s'y projette.

2.5.1 Le modèle *RVB*

La technologie des moniteurs utilisant le modèle trichromatique *Rouge Vert Bleu*, la plupart des algorithmes de rendu calculent ces trois composantes indépendamment les unes des autres en tenant compte des caractéristiques intrinséques des objets de la scène et des sources lumineuses. Ces caractéristiques sont elles aussi le plus souvent décrites avec le modèle *RVB*. L'utilisation de tout autre modèle oblige, à un moment ou à un autre, de transposer les valeurs calculées dans le modèle *RVB* nécessaire au moniteur.

2.5.2 Le modèle *TIS*

Le modèle Teinte Intensité Saturation se veut plus près des perceptions humaines. Il est représenté par un double cône dont l'axe de symétrie correspond à l'Intensité, celle-ci variant de 0 (noir) au sommet du cône inférieur à 1 (toutes les couleurs d'intensité maximale) au sommet du cône supérieur. L'angle autour de l'axe de symétrie mesure la Teinte de la couleur

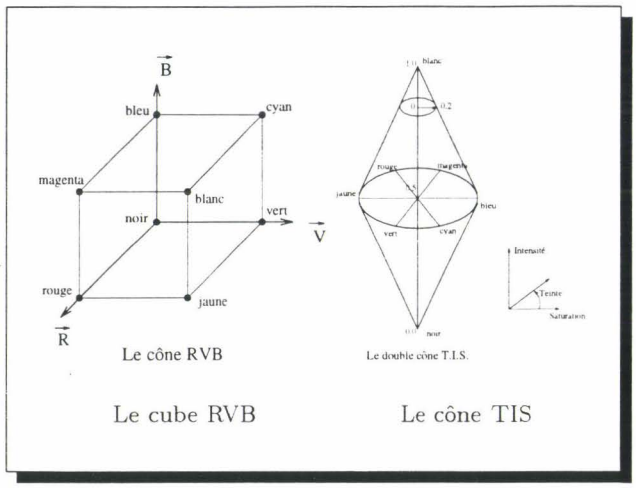


FIG. 2.4 - *Le modèle* RVB *et le modèle* TIS

en degré, arbitrairement le rouge est à 0°. Enfin la Saturation (qui correspond à la proportion de couleur pure par rapport au blanc) est donnée par la distance orthogonale à l'axe de symétrie (entre 0 et 1). Une couleur est saturée (*i.e.* pure) si elle se trouve sur le bord du cône. À l'inverse, si elle se rapproche de l'axe, sa proportion de blanc (ou de gris) augmente. L'axe de symétrie représente l'échelle des gris.

2.5.3 Les autres modèles

De nombreux autres modèles ont été proposés, beaucoup sont très proches du modèle *TIS*, d'autres sont très sophistiqués. En particulier la CIE (Commission Internationale d'Éclairage) a défini un modèle censé représenter toutes les couleurs visibles. Établi expérimentalement, ce modèle permet de définir n'importe quelle couleur visible à partir de trois primaires. Difficile à manipuler, celui-ci a donné naissance à d'autres modèles qui cherchent à se rapprocher des perceptions psycho-sensorielles, citons entre autres : $L^\star a^\star b^\star$ et $L^\star u^\star v^\star$. En fait, les recherches en colorimétrie ne sont pas achevées, aucun modèle ne fait actuellement l'unanimité, il semblerait même que la tendance soit à la définition de différents modèles spécifiques propres à une utilisation particulière.

2.6 Les interpolations

Le rendu projectif utilise beaucoup les interpolations linéaires. En effet, que ce soit pour le calcul des valeurs de profondeur et des couleurs, le principe est toujours le même. Pour une facette triangulaire, des valeurs sont calculées aux trois sommets du polygone, puis une interpolation bi-linéaire permet d'obtenir une valeur intermédiaire pour tous les autres points de la facette.

2.6.1 L'interpolation de Gouraud

Nous l'avons vu, les modèles d'éclairement nécessitent, pour calculer la couleur d'un objet, de connaître la normale de cet objet au point considéré. Or, la facettisation issue de la modélisation des objets ne donne la normale qu'aux sommets des polygones. Il n'est donc pas

possible de calculer directement la couleur d'une primitive en n'importe quel point. Il existe deux techniques d'interpolation pour résoudre ce problème. La méthode de Gouraud calcule les couleurs (trois intensités) aux sommets des primitives, puis interpolent de manière bi-linéaire ces valeurs sur l'ensemble de la facette. L'interpolation de Gouraud est très répandue en particulier pour sa rapidité, mais souffre de son impossibilité dans certains cas à rendre les réflexions spéculaires. En effet, si la tâche spéculaire d'un objet se trouve *à l'intérieur* d'une facette sans toucher les sommets, la méthode de Gouraud est incapable de la détecter et celle-ci n'apparaîtra pas.

Par ailleurs, l'interpolation bi-linéaire (sur un plan) des intensités pour chaque facette, ne garantit pas la continuité des dérivées des fonctions d'éclairement le long des arêtes partagées. Des défauts sous la forme de bande de Mach (perception des frontières entre les facettes) peuvent alors apparaître.

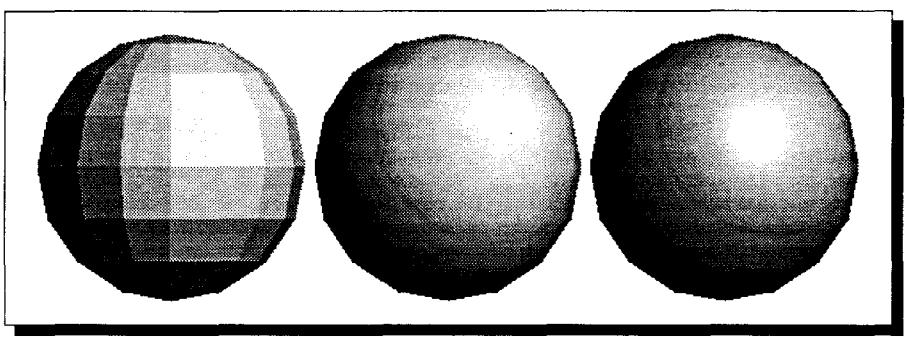


FIG. 2.5 - *Sphère facettisée éclairée à l'aide de la formule de Phong et représentée avec un ombrage constant, une interpolation de Gouraud puis une interpolation de Phong*

2.6.2 L'interpolation de Phong

En réponse à ces problèmes, l'interpolation de Phong (à ne pas confondre avec le modèle d'éclairement de Phong) ne s'applique pas aux valeurs d'intensités pour chaque primitive comme précédemment, mais aux valeurs de normales. Pour chaque point de la surface, une normale est calculée par interpolation bi-linéaire également, celle-ci étant utilisée pour le calcul d'intensité en ce point. Cette méthode résout les problèmes posés plus haut mais induit un coût de calcul considérable, les calculs d'éclairement se faisant ici en chaque point au lieu des seuls sommets de facettisation.

Nous verrons plus loin que le choix du modèle d'interpolation doit être lié à l'algorithme d'antialiassage et que l'interpolation de Gouraud par exemple est mal adaptée à l'échantillonnage surfacique.

2.6.3 Interpolations linéaires et transformations perspectives

Les algorithmes de conversion à ligne de balayage (en anglais *scan converting algorithms*) utilisent beaucoup de paramètres interpolés pour le rendu des images. Que ce soient les coordonnées de texture (u, v) pour le placage de textures planes, les valeurs (r, v, b) pour un éclairement de Gouraud ou les coordonnées des vecteurs normales $(\vec{N}_x, \vec{N}_y, \vec{N}_z,)$ pour un éclairement de Phong, tous ces paramètres sont interpolés dans l'espace écran au moment de la conversion des objets en pixels. À partir des valeurs calculées aux sommets de chaque

primitive, une interpolation bi-linéaire (en x et en y) permet de recalculer ces paramètres d'abord pour chaque ligne, puis pour chaque pixel.

Lors des projections perspectives (pour le rendu de scènes 3D), ces interpolations sont faites dans l'espace écran alors qu'elles ne sont pas linéaires dans ce repère. Lorsqu'il s'agit du placage de texture plane, les défauts qui en résultent sont particulièrement sévères (déformation de la texture), il faut alors mettre en place un système de correction soit en facettisant plus finement les polygones pour atténuer le phénomène (c'est la solution utilisée par les stations de travail *Silicon Graphics VGX*), soit en effectuant l'interpolation dans un système de coordonées homogènes. Nous ne traitons pas ici de ce problème, [Heckbert et al.91] donne une très bonne explication du problème et discute les différentes solutions.

Remarquons simplement que les mêmes erreurs se produisent lors des interpolations de couleur (Gouraud) ou de normales (Phong). Cependant, avant que le problème ne se présente et devienne célèbre pour les textures, il n'avait même pas été remarqué pour les autres paramètres, tant les effets sur l'éclairage (de Gouraud par exemple) sont faibles.

Retenons de cela que l'interpolation de deux couleurs mêmes très proches est une opération délicate et qui est rarement faite dans les règles de l'art, cependant les effets ne sont pas suffisamment notables pour être remarqués. Si les calculs sont exécutés avec suffisamment de précision (en utilisant par exemple 8 bits pour chaque composante) la différence de couleur entre des pixels voisins sur une même surface est si petite que l'on obtient des dégradés très *réalistes*. D'autre part, la résolution des écrans est assez fine pour que le fait de considérer la couleur d'un objet constante sur la surface d'un pixel n'entraîne aucun défaut *visible* (en tous cas en ce qui concerne l'éclairage).

2.7 Le matériel dédié au rendu projectif

Toutes les machines 3D temps réel actuelles implémentent ce que l'on appelle le pipeline de rendu Gouraud Zbuffer, les plus performantes ajoutant également le placage de textures planes. La figure 2.6 présente l'architecture générique d'un tel système.

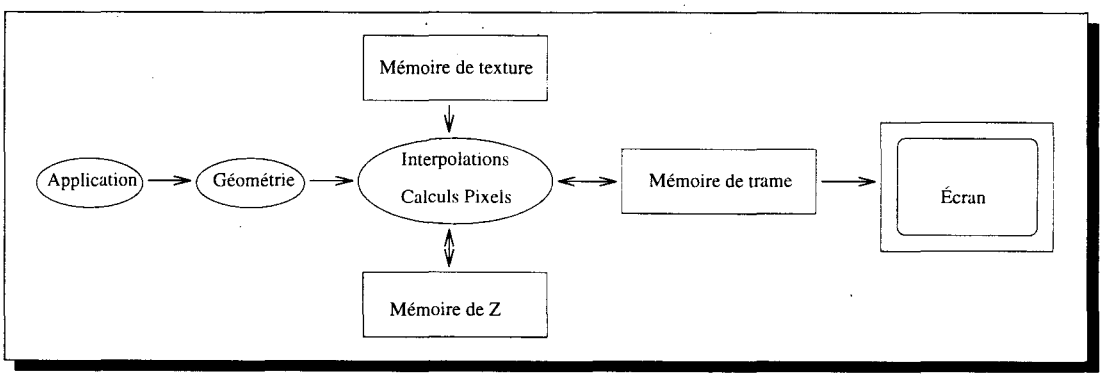


FIG. 2.6 - *Architecture d'un système 3D*

La partie *géométrie* travaille au niveau facette, et inclut les transformations géométriques, les calculs d'éclairement, le clipping et la transformation perspective. La partie *interpolations* travaille au niveau pixel, et inclut le remplissage des primitives, les interpolations de profondeur, couleur et coordonnées de texture, et la division perspective pour ces mêmes coordonnées. La partie *calculs pixel* inclut l'interpolation bi ou tri-linéaire sur la texture, ainsi qu'un

éventuel calcul de blending.

Chapitre 3

Quelques éléments du traitement de signal

Nous nous gardons bien de vouloir faire ici un cours de traitement du signal, discipline complexe dont nous ne sommes pas spécialistes. Néanmoins, c'est grâce aux connaissances acquises dans ce domaine que nous pouvons comprendre les phénomènes d'aliassage (ou ce qui y ressemble) qui nous préoccupent et par là même tenter de trouver des solutions nouvelles ou bien d'évaluer les solutions existantes. Cette partie a donc pour but de rappeler quelques éléments importants de **traitement de signal** qui sont nécessaires à la bonne compréhension des phénomènes d'**aliassage** tels que nous pouvons les observer en synthèse d'images ou dans tout autre exemple du processus discrétisation/reconstruction d'un signal continu, tel que les télécommunications, la technologie *Compact Disc*, etc ... Ces notions sont évidemment incomplètes, le choix des parties présentées a été fait arbitrairement en fonction de ce qui nous paraissait pertinent par rapport au problème considéré.

Les aspects détaillés sont illustrés avec des signaux monodimensionnels dans un souci de clarté. L'extension à des signaux bidimensionnels tels que des images est immédiate et n'entraîne aucune modification.

3.1 Notion de signal

3.1.1 Signal continu/signal discret

Il convient pour commencer de définir la notion de **signal**, qui n'est rien d'autre qu'une fonction véhiculant de l'information. Cette information peut dépendre d'une ou plusieurs variables. Il est courant de parler de signal temporel (*respectivement spatial*) quand la valeur de la fonction dépend du temps (*respectivement d'une distance quelconque*). Dans le cas qui nous préoccupe, nous dirons qu'une image est un signal spatial bidimensionnel. En effet, l'intensité d'un point de l'image est fonction de sa position en x et en y.

Nous parlerons par la suite de **signal continu** (ou *analogique*) et de **signal discrétisé** (ou *digitalisé* ou encore *échantillonné*). Un signal continu peut être vu comme un continuum de valeurs sur un intervalle alors qu'un signal discret est constitué d'un ensemble de valeurs distinctes.

La théorie de l'échantillonnage décrit les relations qui existent entre un signal analogique (continu) et sa version discrétisée constituée d'échantillons.

3.1.2 Deux domaines de représentation

Lorsqu'on considère un signal quelconque, il est indispensable d'avoir présent à l'esprit deux représentations possibles de ce signal, une représentation-temps (de la forme $y = f(t)$) dans laquelle la variable indépendante est le temps qui s'écoule et une représentation-fréquence ($Y = F(\nu)$) où la variable indépendante est cette fois la fréquence (dimension inverse du temps). La théorie de Fourier permet de manipuler ces deux représentations complémentaires, ce qui est indispensable pour comprendre et utiliser les méthodes de traitement de signal.

Notons que nous utilisons des signaux définis dans le domaine *Temps/Fréquence*, mais que la théorie peut s'étendre à tous les domaines. En particulier la variable indépendante peut être spatiale, dans le domaine associé la variable sera alors l'inverse d'une longueur. Par abus de langage, nous parlerons encore de fréquence (spatiale).

3.1.3 Décomposition de Fourier

Tout signal peut être considéré du point de vue *spectral*, *i.e.* en regard des différentes fréquences qui le constituent. En effet, un signal périodique se décompose en une somme (avec décalage de phase) de signaux sinusoïdaux de fréquences et d'amplitudes différentes. La théorie de Fourier a pour but de déterminer quels sinus constituent un signal particulier. Pour cela, on utilise les **transformées de Fourier** qui permettent de représenter une fonction spatiale (ou temporelle) $f(x)$ en une fonction fréquentielle équivalente que nous noterons $F(\nu)$. C'est la *représentation fréquentielle* (dans le domaine des fréquences) du signal, alors que $f(x)$ est la *représentation spatiale*. Il existe également une fonction duale appelée **transformée de Fourier inverse** qui permet d'effectuer l'opération contraire. Grâce à ces deux opérateurs, nous pouvons passer indifféremment d'une représentation à l'autre.

Nous allons voir que cette dualité est très pratique car il est plus facile de travailler dans l'un ou l'autre des domaines suivant l'opération à réaliser.

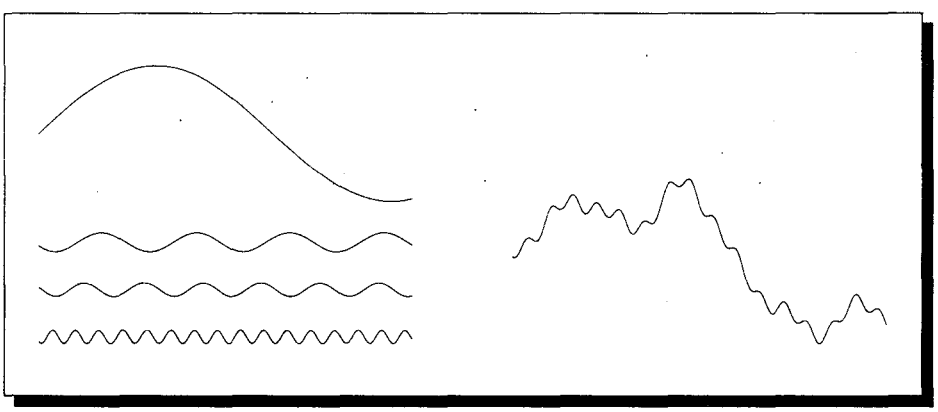


FIG. 3.1 - *La somme des 4 fonctions de gauche donne la fonction de droite.*

3.1.4 Définition spatiale et fréquentielle

La *Transformée de Fourier* d'un signal (*i.e.* d'une fonction) intégrable et continu $f(x)$ du domaine spatial vers le domaine fréquentiel est donnée par :

$$F(\nu) = \int_{-\infty}^{+\infty} f(x) e^{-i2\pi\nu x} dx \tag{3.1}$$

En retour, la *Transformée de Fourier Inverse* nous permet de transformer un signal intégrable $F(\nu)$ du domaine fréquentiel vers le domaine spatial :

$$f(x) = \int_{-\infty}^{+\infty} F(\nu) e^{i2\pi\nu x} d\nu \tag{3.2}$$

Rappelons que $i = \sqrt{-1}$ et que selon la formule d'Euler, on a :

$$e^{-i2\pi\nu x} = \cos 2\pi\nu x - i \sin 2\pi\nu x$$

Si $f(x)$ est une fonction réelle, $F(\nu)$ possède une partie réelle et une partie imaginaire, que nous notons respectivement $Re(\nu)$ et $Im(\nu)$.

Nous pouvons maintenant définir pour chaque fréquence ν, son **amplitude** $|F(\nu)|$ et son **décalage de phase** $\phi(\nu)$ par :

$$|F(\nu)| = \sqrt{Re^2(\nu) + Im^2(\nu)} \tag{3.3}$$

et

$$\phi(\nu) = \tan^{-1}\left[\frac{Im(\nu)}{Re(\nu)}\right] \tag{3.4}$$

Afin de simplifier les calculs, on utilisera plutôt les versions discrétisées de ces relations, en considérant qu'au delà d'une certaine fréquence, la **fréquence de coupure**, l'énergie du signal est négligeable par rapport à l'énergie totale. On dit alors que le signal est à **bande limitée**.

$$F(\nu) = \sum_{x \geq 0}^{N-1} f(x) e^{\frac{-2\pi\nu x}{N}}, 0 \leq \nu \leq N-1 \tag{3.5}$$

$$f(x) = \frac{1}{N} \sum_{x \geq 0}^{N-1} F(\nu) e^{\frac{2\pi\nu x}{N}}, 0 \leq x \leq N-1 \tag{3.6}$$

On représente habituellement la transformée de Fourier d'un signal sur un diagramme donnant l'amplitude en fonction de la fréquence, le décalage de phase de chaque composante n'est pas représenté.

3.1.5 Convolution

Il est temps maintenant de présenter un outil mathématique puissant et fortement utilisé, la **convolution** (on dit également *produit de convolution*). La convolution de deux fonctions spatiale $f(x)$ et $g(x)$ que nous noterons $f(x) \star g(x)$ est donnée par :

$$f(x) \star g(x) = \int_{-\infty}^{+\infty} f(\alpha) g(x - \alpha) d\alpha$$

La convolution de deux fonctions dans le domaine spatial correspond exactement au *produit de leurs transformées de Fourier* dans le domaine fréquentiel et de manière duale, un

produit de fonctions dans le domaine spatial équivaut à la convolution des transformées de Fourier.

Cet outil est utilisé lors de deux étapes importantes du processus à savoir l'échantillonnage et le filtrage.

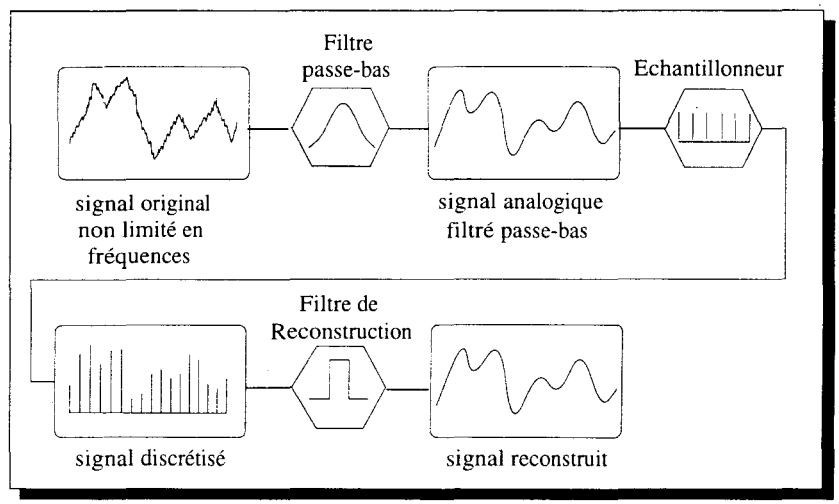


FIG. 3.2 - *Le processus échantillonnage/reconstruction (avec préfiltrage).*

3.2 Le processus discrétisation/reconstruction

Nous allons introduire le processus *discrétisation/reconstruction*, (*cf. figure 3.2*) avec le cas des transmissions téléphoniques numériques. Lors d'une conversation, la première étape consiste à transformer la voix d'un interlocuteur en signal analogique à l'aide d'un dispositif électronique, ensuite un convertisseur *analogique/numérique* le transforme en une suite de valeurs que l'on appelle *échantillons*. Après son transfert sur des lignes numériques, éventuellement accompagné par des opérations telles que compression/décompression la voix est reconstruite d'abord par l'intermédiaire d'un convertisseur *numérique/analogique (C.N.A.)* cette fois, suivi d'un dispositif qui permet de transformer le signal électronique en une voix audible. Sur cet exemple, nous voulons mettre en évidence les deux étapes essentielles du processus, à savoir l'échantillonnage et la reconstruction. Précisons que l'échantillonnage s'accompagne inévitablement d'une numérisation, c'est-à-dire de la *quantification* dans un ensemble discret de valeurs pour chacun des échantillons (l'erreur introduite est directement liée au nombre de bits utilisés pour coder les valeurs). Malheureusement, le terme discrétisation est souvent confondu avec échantillonnage et quantification qui sont pourtant deux notions bien distinctes. La *figure 3.2* détaille les trois étapes *Échantillonnage-Quantification-Reconstruction.*

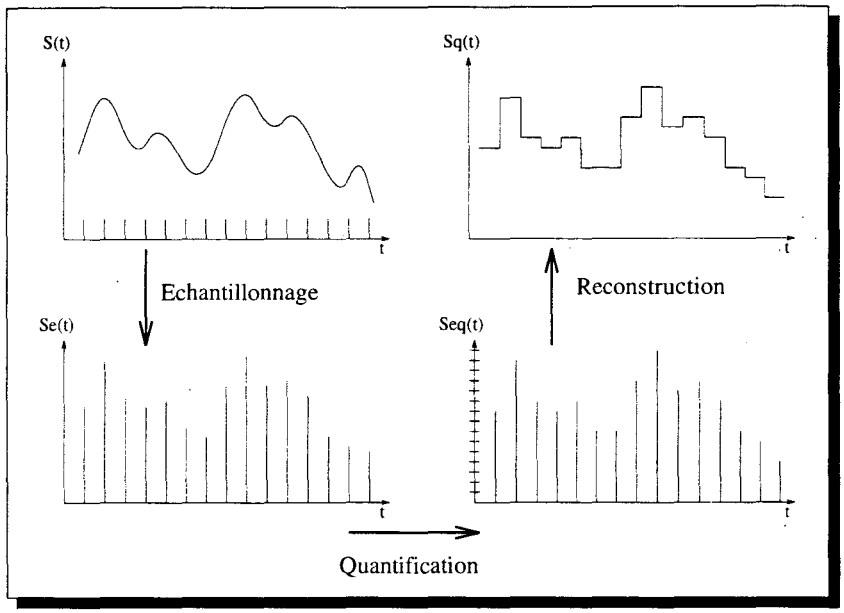


FIG. 3.3 - *Échantillonnage - Quantification - Reconstruction*

L'exemple du téléphone est un cas d'école car chacune des étapes est clairement identifiée. Mais la synthèse d'image n'entre pas complètement dans ce cadre général. En effet, d'une part il n'existe pas de signal analogique de départ : le procédé d'acquisition de l'image est complètement virtuel par échantillonnage de la base de données décrivant la scène, l'algorithme de rendu cumulant les fonctions de génération de l'image (*le signal*) et son échantillonnage (*cf. figure 3.4*). D'autre part, la reconstruction du signal continu est prise en charge par le dispositif d'affichage, dépendant de la technologie, et donc hors de contrôle.

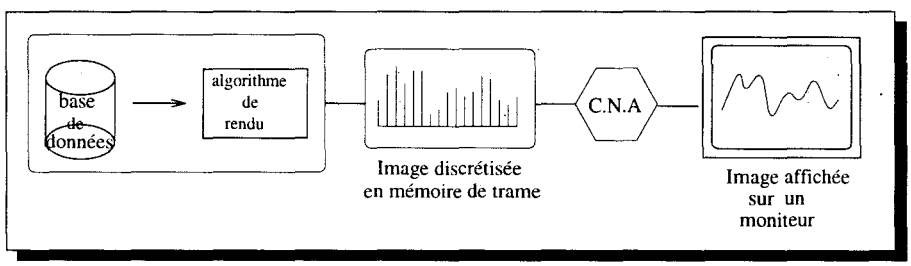


FIG. 3.4 - *Le processus de génération d'images de synthèse.*

3.2.1 Échantillonnage

Soit $s(x)$ un signal analogique, à bande passante limitée, de fréquence maximale f_{max}. L'échantillonner revient à le multiplier par un signal d'échantillonnage $e(x)$ qui est en général une suite périodique d'impulsions infiniment courtes, communément appelé *peigne de Dirac*. Chaque impulsion étant régulièrement espacée d'un intervalle Δx, la transformée de Fourier d'un tel signal est également un train d'impulsions dont la fréquence est $1/\Delta x$. Dans le domaine fréquentiel, échantillonner signifie faire la convolution des transformées de Fourier de s et de e.

Le résultat de l'échantillonnage est donc, dans le domaine spatial, une suite d'impulsions

avec la même fréquence que le peigne d'échantillonnage, et dont chaque amplitude correspond à la valeur du signal pour cet échantillon. Dans le domaine fréquentiel, le spectre du signal original est dupliqué sur chaque impulsion du signal d'échantillonnage, on obtient donc un signal infini qu'il faut filtrer pour ne garder que le spectre principal et éliminer les répliques.

L'échantillonnage, tel que nous venons de le décrire, est une étape primordiale qui doit respecter certaines règles pour permettre une bonne reconstruction. Le théorème de Shannon détermine la fréquence minimale qui permet d'éviter le repliement de spectre.

Le théorème de Shannon

La *figure 3.5* montre que la *"distance"* entre deux échantillons a une importance capitale dans le processus vis-à-vis de la fidélité du signal reconstruit.

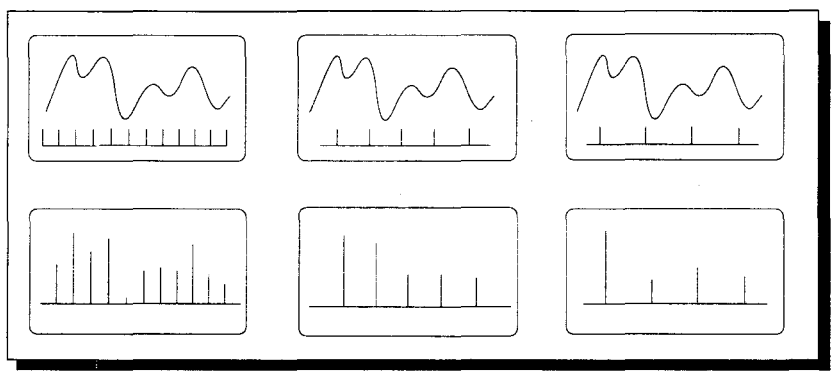


FIG. 3.5 - *Échantillonnage à différentes fréquences dans le domaine spatial.*

Si la fréquence d'échantillonnage est insuffisante, une partie de l'information véhiculée par le signal sera à jamais perdue et le résultat de l'échantillonnage ne permettra pas une reconstruction fidèle du signal original. Nous pouvons remarquer que plus les échantillons sont proches les uns des autres, et mieux la fonction est approchée par les valeurs d'échantillonnage. Le théorème de Shannon donne la fréquence minimum qu'il faut utiliser pour que le signal puisse être reconstruit intégralement. Cette fréquence, que l'on appelle la *fréquence de Nyquist*, est égale à deux fois la fréquence maximale du signal original ($f_n = 2f_{max}$).

La perte d'information (*i.e.* les hautes fréquences) n'est pas le principal inconvénient d'un mauvais échantillonnage. En effet, lors de la reconstruction, les différentes valeurs d'échantillons provenant de ces hautes fréquences sont présentes et *polluent* en quelques sorte le signal. la figure 3.6 montre un exemple où une haute fréquence mal échantillonnée se transforme en une basse fréquence lors de la reconstruction. Celle-ci était absente du signal de départ, elle vient d'être générée. C'est exactement ce processus qui se produit lorsqu'on observe des motifs de *moirés* sur certaines textures en synthèse d'images.

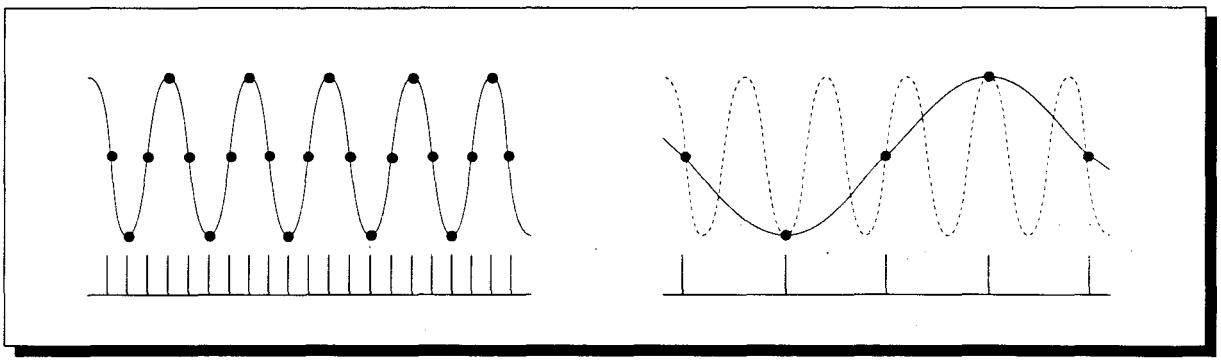


FIG. 3.6 - *Le même signal échantillonné à deux fréquences différentes*

Repliement de spectre

Ce résultat se comprend très bien lorsqu'on examine les choses dans le domaine fréquentiel ; la *figure 3.7* montre le résultat de l'échantillonnage d'une même fonction à différentes fréquences. Si la fréquence d'échantillonnage est inférieure à la fréquence de Nyquist, une partie du spectre principal est recouverte par une réplique (*cf. figure 3.7(b)*) : on a affaire à un *repliement du spectre*, ce qui a pour conséquence de générer des basses fréquences inexistantes dans le signal de départ. Ce phénomène de superposition est irréversible, aucun filtre ne pourra corriger l'erreur introduite. On parle parfois de *pré-aliassage* pour caractériser ce phénomène qui peut être vu comme résultant d'un sous-échantillonnage du signal ou bien d'une absence de pré-filtrage. Si on considère les images comme des signaux spatiaux bidimensionnels, il faut parler de fréquence spatiale (*grandeur inverse d'une distance*), nous utiliserons également le terme de *résolution* en pensant à la distance qui sépare deux valeurs.

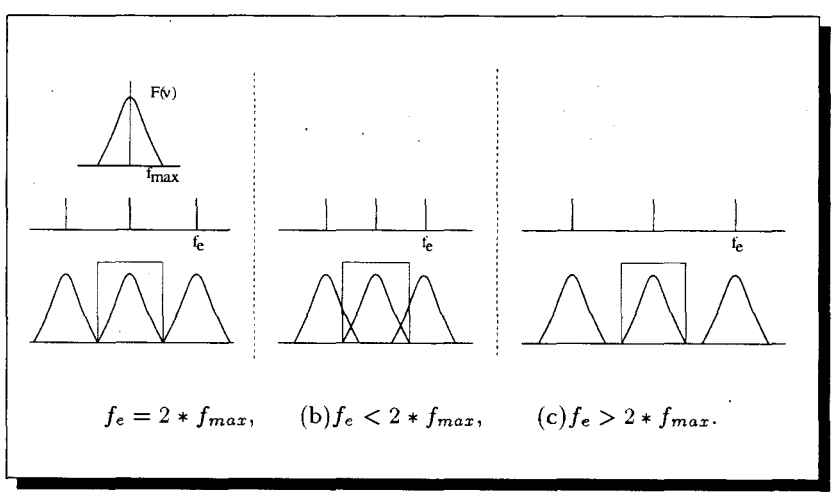


FIG. 3.7 - *Représentation fréquentielle de l'échantillonnage à différentes fréquences.*

3.2.2 Quantification

L'échantillonnage n'est en fait qu'une étape abstraite qui s'accompagne inévitablement d'une numérisation. Pratiquement, l'échantillonnage s'effectue soit par calcul, soit à l'aide

d'instruments de mesure. Dans l'un et l'autre des cas, les valeurs des échantillons sont *quantifiées* dans un ensemble discret de valeurs possibles, l'erreur introduite est alors directement liée au nombre de bits utilisés pour coder les valeurs.

3.2.3 Reconstruction

L'étape de reconstruction consiste à retrouver à partir des différentes valeurs du signal discrétisé, un signal continu aussi proche que possible du signal original. Pour cela, on utilise un *filtre de reconstruction* qui a pour fonction d'éliminer les répliques du spectre principal tout en laissant passer toute la bande du signal sans le déformer. Le meilleur filtre que l'on puisse espérer du point de vue théorique est une fonction *porte* dans le domaine fréquentiel, laissant passer la totalité du spectre principal et uniquement celui-ci (*cf. figure 3.7*). Malheureusement, ce type de filtre est en pratique difficilement réalisable et ce sont souvent des approximations qui sont utilisées.

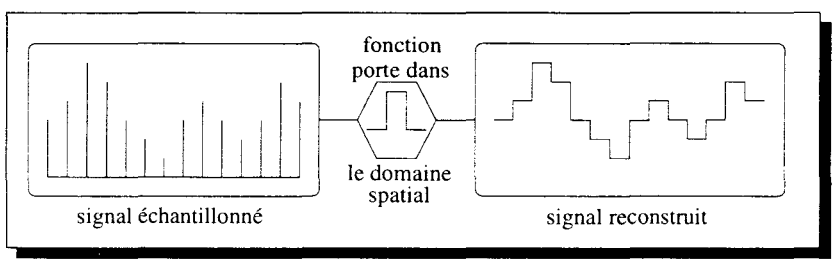


FIG. 3.8 - *Crénelage introduit par le filtre de reconstruction*

En synthèse d'images, le signal discret correspond aux valeurs calculées par les algorithmes, que l'on stocke généralement dans une *mémoire de trame*, et le signal continu est l'image visible sur le moniteur. La reconstruction est effectuée par le dispositif d'affichage qui se contente d'envoyer une valeur unique par pixel correspondant à la couleur stockée en mémoire. Ce mécanisme, lié à la technologie, et donc inaccessible au programmeur, introduit à cette étape un crénelage dans l'image (*cf. figure 3.8*) qui heureusement, est légèrement atténué d'abord par les convertisseurs numérique/analogique qui sont incapables de générer des sauts d'intensité brutaux d'un pixel à l'autre puis, sur les écrans à tube à rayon cathodique, par le spot de balayage qui possède les propriétés d'un filtre gaussien.

Les conséquences sur le signal à ce niveau (*cf. figure 3.8*) sont parfois appelées *post-aliassage* car elles interviennent après l'échantillonnage.

Le phénomène est en fait très différent de celui décrit précédemment car il correspond à l'introduction de hautes fréquences par le processus de reconstruction, il nous semble donc plus correct de parler de *crénelage*[1] et de garder le terme *aliassage* pour les cas de recouvrement de spectre.

3.2.4 Conclusion

Lorsque le signal reconstruit n'est plus le même que le signal de départ, on dit que c'est un *alias*. Deux phénomènes sont à l'origine de cette déformation : le *repliement du spectre* dû à une fréquence d'échantillonnage mal adaptée, et le *crénelage* introduit en fin de chaîne par

1. Dans [Glassner95] Andew S. Glassner utilise le terme de *Reconstruction errors*.

un mauvais filtre de reconstruction. La perte des très hautes fréquences (au-delà de la moitié de la fréquence d'échantillonnage) due à un préfiltrage (*cf.* ci-dessous) approprié n'est pas considérée dans le signal reconstruit comme de l'aliassage.

Dans le cas général, on pourra donc lutter contre l'aliassage en augmentant la fréquence d'échantillonnage et en préfiltrant (à l'aide d'un filtre passe-bas) le signal original. Par ailleurs, le crénelage peut être atténué en utililisant un filtre de reconstruction approprié.

3.3 Le préfiltrage

Nous avons vu que la reconstruction d'un signal est en fait une étape de filtrage. Nous expliquons maintenant l'étape préliminaire qui consiste à préfiltrer le signal avant l'échantillonnage. Cette opération élimine les hautes fréquences et permet donc d'échantillonner dans les conditions de Nyquist. La figure 3.9 montre un exemple dans le domaine fréquentiel. Le filtre délimite une fréquence de coupure f_c au delà de laquelle les fréquences du signal original seront éliminées.

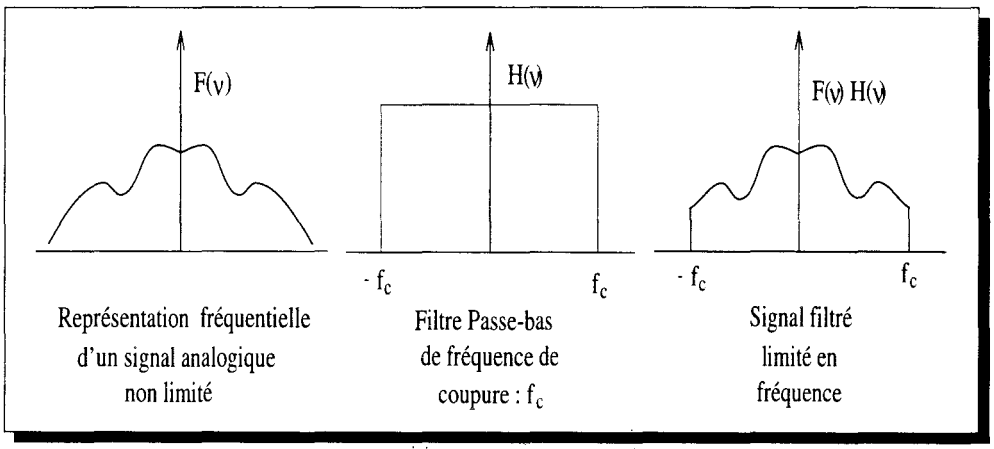


FIG. 3.9 - *Filtrage passe-bas d'un signal analogique dans le domaine fréquentiel.*

Le signal de départ qui pouvait être à bande illimitée (*i.e.* possédant des fréquences infinies), se retrouve après préfiltrage limité en fréquence. À partir de là, il devient facile de calculer la fréquence de Nyquist pour l'échantillonner correctement :

$$f_n = 2 \quad f_c$$

Notons qu'un tel filtre se doit de modifier le moins possible les fréquences qu'il laisse passer.

3.4 Antialiassage

Après ce qui vient d'être exposé, il apparaît clairement la chose suivante : si le rapport *fréquence d'échantillonage/fréquence maximale du signal* est inférieur ou égal à 2, le processus *échantillonnage/reconstruction* transforme les hautes fréquences du signal original en basses

fréquences dans le signal reconstruit. Le nouveau signal est alors un *alias* de l'original. Les solutions à ce problème sont de trois ordres :

- Pour une fréquence d'échantillonnage donnée, le signal analogique doit être préfiltré afin d'éliminer ses composantes se trouvant au-delà de la fréquence de Nyquist.
- Si la bande passante du signal n'est pas limitée, toute augmentation de la fréquence d'échantillonnage aura pour conséquence la diminution des phénomènes d'aliassage sans jamais les faire disparaître complètement.
- Le plus grand soin doit être apporté au filtre de reconstruction car son rôle est essentiel.

Or, il se trouve qu'en synthèse d'images la fréquence d'échantillonnage est fixée par la définition de la grille d'affichage (matrice de pixels). Se placer dans les conditions de Nyquist revient donc à pré-filtrer le signal de départ pour en éliminer les fréquences inaccessibles.

Chapitre 4

Aliassage en synthèse d'images

4.1 Le processus de synthèse d'images

Du chapitre précédent, nous pouvons tirer qu'il existe deux manières de lutter contre l'aliassage : augmenter la fréquence d'échantillonnage et filtrer le signal original, ces deux approches n'étant pas mutuellement exclusives. Cependant, il faut comprendre que le processus de synthèse d'images n'entre pas exactement dans le cas général décrit plus haut. En particulier, il n'existe pas de signal analogique de départ. Celui-ci est implicite dans la base de données décrivant la scène. La principale conséquence est qu'il n'est pas possible d'insérer un dispositif de filtrage entre le signal analogique et l'échantillonneur. En synthèse d'images, l'algorithme de rendu cumule les fonctions de génération et d'échantillonnage (*cf. figure 4.1*). D'autre part, la technologie de l'écran impose la définition du signal discrétisé avant reconstruction. Le signal reconstruit étant l'image affichée sur l'écran, chaque pixel correspond à une couleur unique. Même si le signal est sur-échantillonné par rapport à la grille d'affichage, il doit être filtré vers la résolution finale avant la reconstruction.

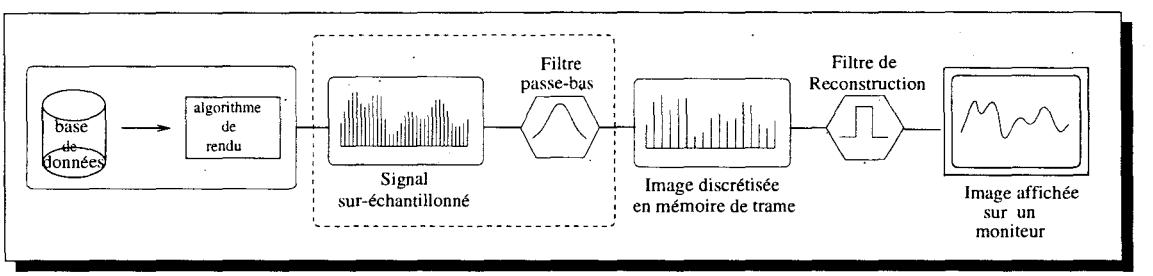


FIG. 4.1 - *Le processus de génération d'images de synthèse - Le cadre en pointillés correspond au cas où l'image est sur-échantillonnée.*

On distingue deux types d'aliassage, l'aliassage spatial et l'aliassage temporel, et lorsque les deux se combinent, cela donne des phénomènes d'aliassage spatio-temporel.

En synthèse d'images fixes, les deux principales manifestations de l'aliassage sont l'apparition de motifs réguliers sur les objets texturés (*moirés*) et la présence des marches d'escaliers (*jaggies*) sur les contours d'objets. Cependant, on répertorie d'autres phénomènes tels que des trous dans des objets fins ou sur des reflets. D'autre part, sur les scènes animées, on observe le clignotement de certains objets, des déplacements saccadés et le scintillements de certains motifs de texture.

Images de synthèse *vs* images digitalisées Avant de pousuivre, il est important de préciser la différence essentielle qui existe entre les *images de synthèse* (calculées à l'aide d'algorithmes) et les *images numériques* provenant par exemple d'une caméra ou d'un scanner. Ces dernières, issues du monde réel, ont été capturées à l'aide d'instruments d'optique pour lesquels l'échantillonnage n'est jamais strictement ponctuel. Les capteurs jouent le plus souvent le rôle de filtre, le spot d'une caméra par exemple peut être assimilé à un filtre gaussien. Ces images peuvent bien entendu présenter des défauts d'aliassage ou de crénelage provenant du processus d'acquisition et/ou de leur reconstruction (*i.e.* affichage), mais c'est cette fois un problème de technologie sur lequel nous ne nous attarderons pas. Nous plaçons notre étude dans le cadre algorithmique des méthodes de synthèse d'images, l'important pour nous est de mettre en évidence les conséquences visibles, sous forme de défauts de l'image, des différentes méthodes, en particulier des techniques d'échantillonnage ponctuel.

4.2 Aliassage spatial

4.2.1 Les moirés sur les motifs de texture

Le placage de texture planes permet d'augmenter considérablement le réalisme des scènes. On dispose au départ d'une image correspondant au motif de texture dont chaque point est appelé un *texel*[1]. Le principe consiste à déterminer pour chaque pixel de la facette à texturer, le texel correspondant dans le repère de la texture.

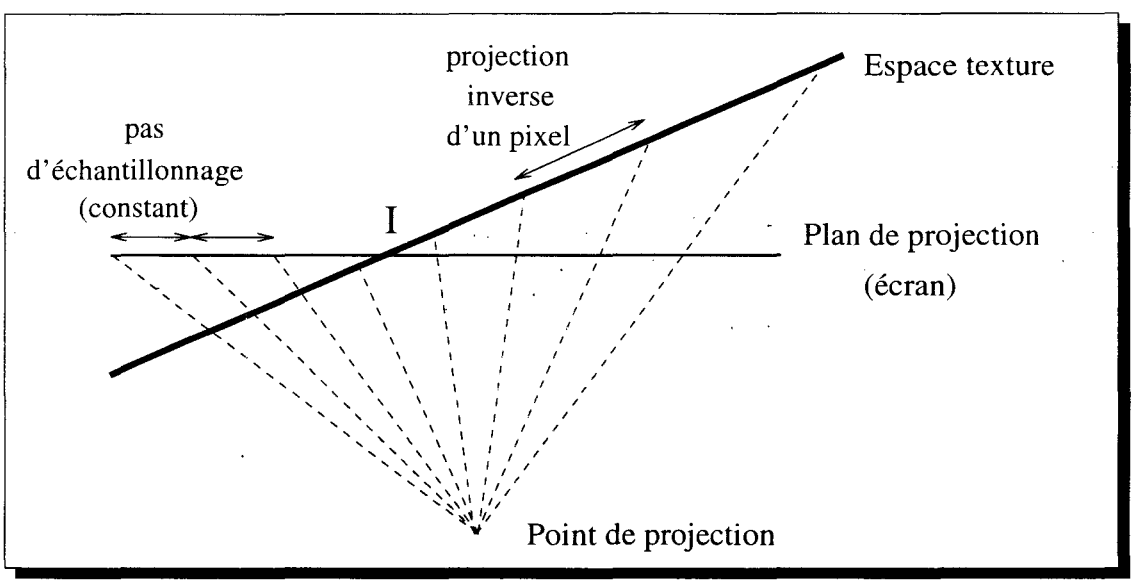


FIG. 4.2 - *Mise en perspective de texture plane,* (d'après [Gangnet et al.84])

En synthèse 3D, lors des transformations perspectives sur les facettes, les motifs de texture correspondants sont déformés. Ils sont soit compressés, soit dilatés. Il en résulte des défauts majeurs par la génération de *moirés* (*cf. photo cube1 et figure 4.3*) dans le premier cas et sous la forme d'un *crénelage* dans le second.

1. par similitude avec le pixel, on emploie texel pour *texture element*

Les moirés apparaîssant sur certains motifs sont une caractérisation du phénomène d'aliassage au sens le plus exact du terme. En effet, on assiste bien à la création de basses fréquences absentes du signal de départ, à cause d'un échantillonnage à une fréquence insuffisante.

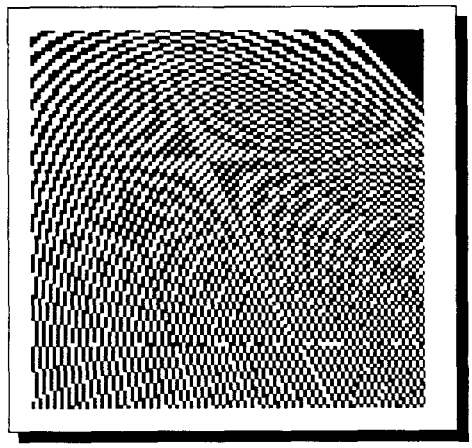


FIG. 4.3 - *Un exemple de moirés*

En appliquant une *transformation perspective inverse*, on peut calculer l'image dans l'espace texture de la surface de chaque pixel ; c'est sur cette surface qu'il faudrait intégrer pour obtenir la couleur à afficher. La *figure 4.2*, inspirée de [Gangnet et al.84] montre les phénomènes de compression (à droite du point I) et de dilatation (à gauche du point I) sur un exemple monodimensionnel. Dans la suite, on parlera de taux de compression pour représenter ces phénomènes.

Les techniques d'antialiassage permettant de résoudre ces problèmes sont décrites au §5.1.

4.2.2 Les bords d'objet

Il existe trois types de défauts sur les bords des objets: les marches d'escalier sur les arêtes obliques, le décalage des arêtes verticales et horizontales, les trous dans les objets fins ou pointus. Ces trois phénomènes se manifestent sur les images fixes et engendrent d'autres défauts sur les images animées.

Les marches d'escaliers (*jaggies*)

Le crénelage des bords d'objets et des segments de droites (*cf. figure 4.4*), si caractéristiques des images de synthèse, est souvent cité en même temps que les moirés lorsque l'on parle de l'aliassage. Or, le phénomène est différent. Il ne s'agit pas ici d'un recouvrement de spectre tel que nous l'avons décrit précédemment, mais plutôt de hautes fréquences introduites par le processus au moment de la reconstruction. Le terme aliassage est dans ce cas utilisé par abus de langage.

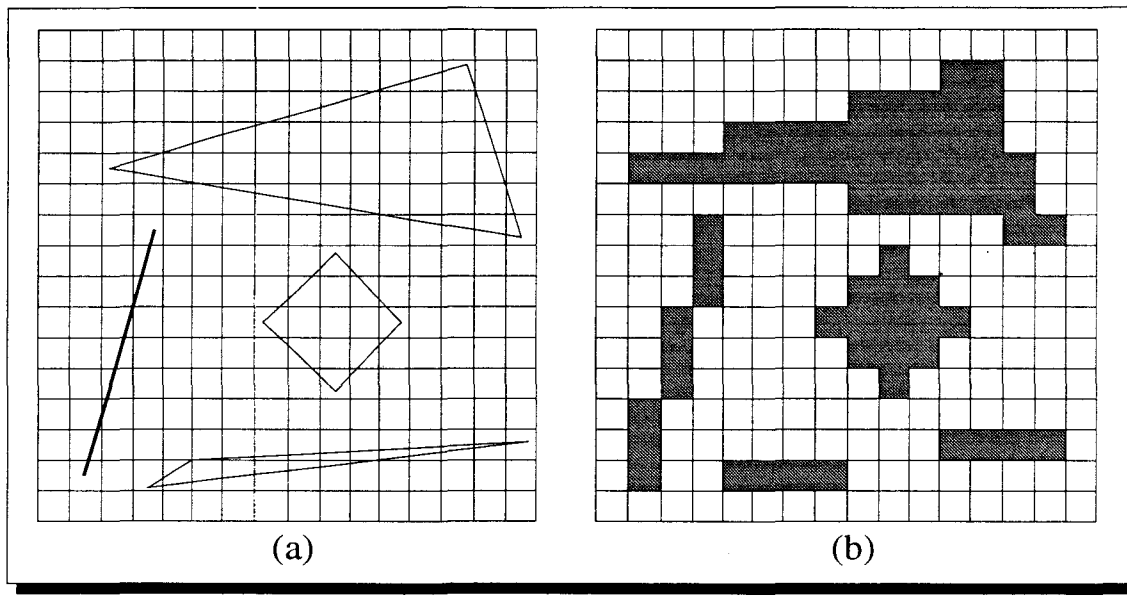


FIG. 4.4 - *crénelages et trou : (a) les objets réels - (b) leur projection sur la grille discrète.*

En effet, le caractère tout-ou-rien des algorithmes d'affichage fait qu'ils sont incapables de rendre compte des arêtes obliques. L'écran étant assimilé à une grille rectangulaire régulière, les bords d'objets sont approchés par des segments de droites verticaux et/ou horizontaux.

Décalage des arêtes parallèles aux axes x et y

La *figure 4.5* montre un autre exemple de défaut sur les bords des objets. Lorsque les arêtes des primitives sont parallèles aux axes de la grille d'affichage aucune marche d'escalier n'est visible. Pourtant une autre forme d'aliassage peut se manifester sous la forme d'un décalage des bords. En effet, la précision d'affichage étant de l'ordre du pixel, les objets seront déformés et apparaîtront plus petits ou plus grands. Ce défaut, qui peut paraître négligeable à première vue, ne l'est pas du tout car sur des scènes précises, cette déformation peut faire disparaître des détails (*par exemple certains objets paraissent collés alors qu'ils ne le sont pas*). D'autre part, nous verrons qu'en animation un phénomène de *saccade* résulte de ce défaut.

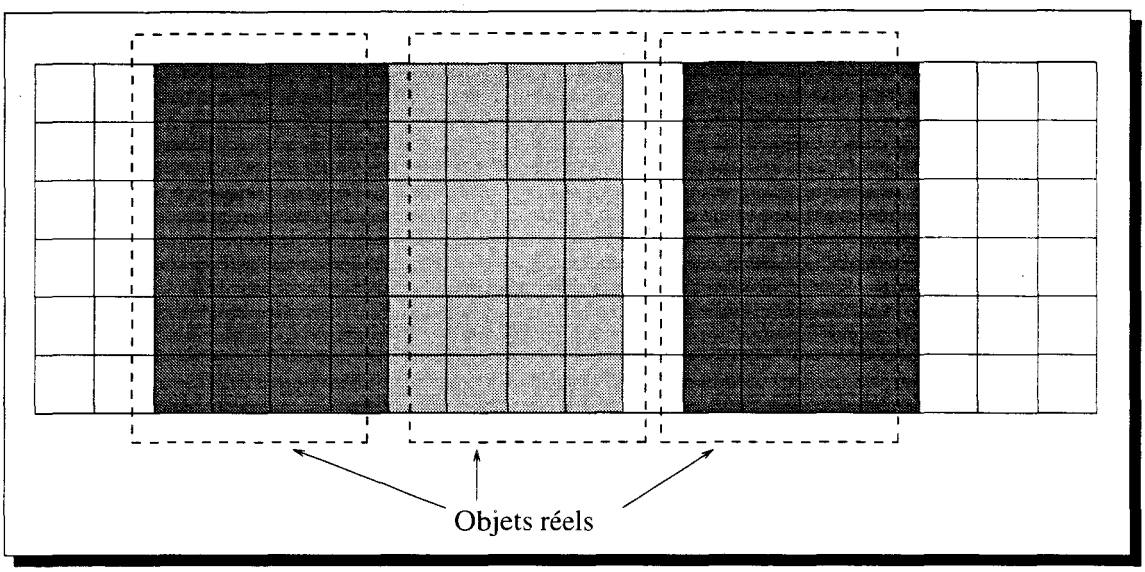


FIG. 4.5 - *Défauts de visibilité dus aux décalage des arêtes*

4.2.3 Trous et disparitions

Les trous dans les objets pointus (*cf. figure 4.4*) ainsi que la disparition des objets de taille inférieure au pixel sont la manifestation d'une double cause. Comme précédemment, la loi du tout ou rien de l'échantillonnage ponctuel associée à un mauvais filtre de reconstruction, introduit un crénelage. Mais surtout, il apparaît clairement qu'il y a dans ce cas **perte d'informations** due à un taux d'échantillonnage insuffisant. Les objets de taille inférieure au pixel générant des fréquences supérieures à la moitié de la fréquence d'échantillonnage, ces derniers ne peuvent pas être correctement traités.

Notons que ce phénomène dépend de l'algorithme de rendu utilisé. En effet, quelle que soit la taille de l'objet, les algorithmes de tracé de segments et de remplissage par contour/segments ne peuvent pas générer de trous. Pour les triangles fins, les deux cotés seront tracés par des algorithmes classiques qui assurent qu'au moins un pixel par ligne et par colonne est allumé. Au pire, les deux cotés allumeront les mêmes pixels, mais aucun trou ne pourra apparaître. Par contre, les algorithmes de remplissage par tests d'inclusion peuvent afficher des polygones avec des trous comme le montre la *figure 4.4*, il sufit que l'objet soit de taille inférieure au pixel et qu'il se trouve par endroit *entre les pixels.*

Dans ce cas, le déplacement des objets en animation va créer des clignotements aux endroits critiques.

4.3 Aliassage temporel et spatio-temporel

4.3.1 Les roues de chariots

Lorsque les images sont animées, une nouvelle forme d'aliassage intervient, il s'agit de l'**aliassage temporel**. On retrouve dans cette troisième dimension les problèmes liés à l'échantillonnage. Le phénomène très connu des roues de chariots dans les westerns qui donnent l'impression de tourner à l'envers est un exemple de génération de basse fréquence

à partir d'un signal insuffisamment échantillonné dans le temps (*cf. figure 4.6*). (Les roues semblent toujours tourner moins vite à l'envers que ce qu'elles tournent réellement à l'endroit).

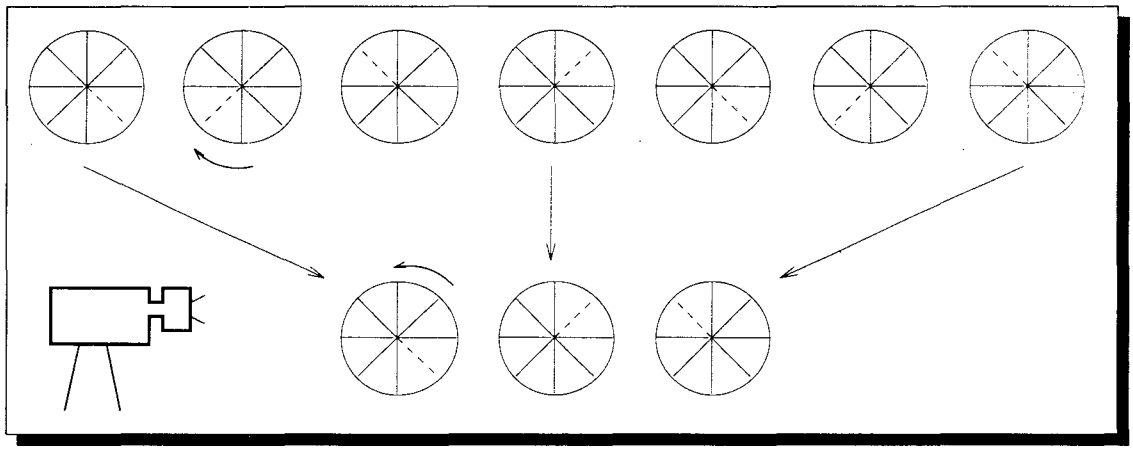


FIG. 4.6 - *Aliassage Temporel*

4.3.2 L'effet stroboscopique

L'animation en synthèse d'images est comparable au cinéma en ce sens que les mouvements sont décomposés en une suite d'images. L'obturateur d'une caméra ou encore une lampe stroboscopique créent le même effet que le calcul image par image d'une séquence d'animation en échantillonant les images dans le temps. Au moment de la projection, des mécanismes psycho-visuels font percevoir les suites d'images comme un mouvement continu, les objets qui se déplacent peuvent alors avoir un mouvement saccadé. Le phénomène est comparable au crénelage observé sur les images fixes, il provient de la reconstruction brutale image par image de la séquence, ce n'est donc pas à proprement parler un phénomène d'aliassage.

4.3.3 Conséquences de l'aliassage spatial en animation

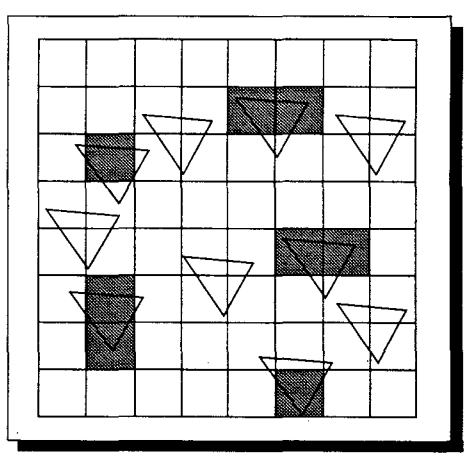


FIG. 4.7 - *Le clignotement en animation.*

Les deux phénomènes que nous venons de décrire sont indépendants de l'imagerie numérique puisqu'ils se manifestent également au cinéma. Cependant, ils peuvent se combiner avec des problèmes de hautes fréquences spatiales en synthèse d'images, pour donner ce que l'on appelle l'**aliassage spatio-temporel**. L'exemple le plus célèbre est certainement le clignotement des petits objets qui apparaissent sur certaines images et ne sont plus présents sur les suivantes pour réapparaître ensuite (*cf. figure 4.7*), ils sont le pendant en animation des trous sur les images fixes. On peut également citer l'effet dit de *cordelette* (twisted rope) qui se produit sur le bord des objets en décalant la position des marches d'escalier sur les images successives. D'autre part, le décalage des bords de polygones (*cf.* plus haut) fait qu'un objet qui se déplace parallèlement aux axes se décale au moins d'un pixel. Ceci est particulièrement visible sur les déplacements lents le long des axes où les objets bougent par à-coups. Enfin, l'animation d'objets texturés, en particulier lorsque les motifs sont compressés et qu'il existe des phénomènes de moirés, engendre un scintillement des motifs (*en anglais flicking*) particulièrement désagréable qui correspond au phénomène de cordelette étalé sur une surface.

Le tableau suivant récapitule les correspondances entre entre les différents types d'aliassage sur images fixes et en animation :

Les défauts d'aliassage

images fixes	marches d'escalier	trous	décalage	moirés
animation	effet de cordelette	clignotement	déplacements saccadé	scintillement

Les différentes techniques d'antialiassage temporel sont discutées au §5.4

4.4 Couleur et éclairement

4.4.1 Reflets/taches spéculaires

Lorsque la scène contient des reflets (par exemple des taches spéculaires) d'une largeur inférieure à deux pixels, des trous peuvent apparaître. Prenons par exemple un objet cylindrique de quelques pixels de diamètres. Avec un éclairage de Phong, cet objet peut présenter des taches spéculaires inférieures à la taille du pixel.

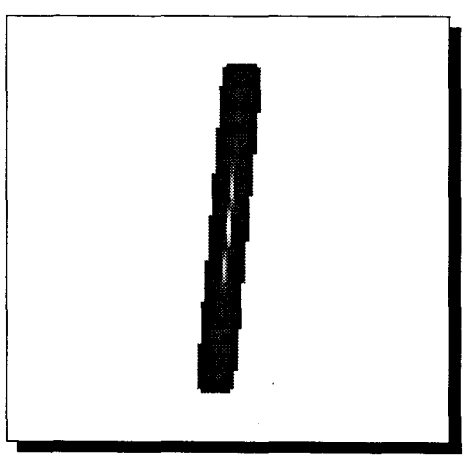


FIG. 4.8 - *Aliassage spéculaire*

L'effet est exactement le même que pour les objets fins. La *figure 4.8* montre un exemple sur une tache spéculaire fine à la surface d'un cylindre de petite taille. Comme pour les petits objets, l'animation peut créer le clignotement des reflets ou des taches spéculaires. La frontière responsable des hautes fréquences n'est pas géométrique mais due aux calculs de couleur.

Des solutions à ce problème sont présentées au §5.3.

4.4.2 Aliassage en radiosité

Il existe une autre forme d'aliassage qui apparaît sur les scènes de radiosité calculées par une méthode de projection sur une surface discrétisée. L'échantillonnage en facettes au moment des calculs des échanges lumineux dans une scène introduit des erreurs grossières. Cela se manifeste par un découpage plus ou moins marqué des objets de la scène en facettes. L'intensité entre deux facettes voisines variant de manière brutale.

Ce problème est détaillé au §8.6, et nous donnons une solution originale pour le résoudre.

4.4.3 Aliassage chromatique

Signalons enfin l'existence d'une forme particulière d'aliassage due au codage de la couleur dans un signal vidéo. Dans certains systèmes dits **composites** ($NTSC$[2], $SECAM$[3],PAL[4]), en opposition aux systèmes à **composantes**, la *chrominance* (couleur) est codée à l'intérieur du signal de *luminance* (le signal noir et blanc). Or, le codage et le décodage de tels signaux nécessitent l'utilisation d'un matériel électronique sophistiqué. Malheureusement le matériel courant ne possède pas les qualités requises, et dans le cas où le signal de luminance contient des hautes fréquences, il peut y avoir recouvrement de spectre avec le signal de chrominance. Il apparaît alors des *artefacts* colorés dûs à l'aliassage chromatique.

2. NTSC signifie *National Television System Commitee*, une mauvaise plaisanterie en a fait *Never Twice the Same Color.*

3. SECAM : SEquence de Couleur Avec Mémoire

4. PAL : Phase Alternation Line

Chapitre 5

Antialiassage en synthèse d'images

Ce chapitre regroupe les différentes techniques d'antialiassage en synthèse d'images, à l'exception des méthodes d'échantillonnage surfacique qui sont l'objet de la *partie II* du mémoire. Nous commençons par présenter les techniques de préfilttrage de textures, qui permettent d'éviter les moirés. La seconde partie traite du sur-échantillonnage sous toutes ses formes. Les deux dernières sont consacrées respectivement, aux traitements possibles pour l'éclairement spéculaire, et à l'antiliassage temporel.

5.1 Le préfiltrage des textures

Nous avons présenté au §4.2.1 les défauts de moirés qui apparaissent sur les motifs de texture lorsqu'ils sont compressés. Nous donnons maintenant les différentes solutions qui permette de résoudre ce problème dans le cadre du placage de textures planes.

En fait, en considérant un pixel comme un rectangle, sa projection dans l'espace des texels est un quadrilatère dont la forme et la taille dépendent de la perspective appliquée à la facette traitée. Une solution aux problèmes de moirés est donc d'appliquer un filtre sur l'ensemble des texels contenus dans le quadrilatère afin d'en déduire l'intensité du pixel. Le coût de cette opération est très élevé et dépend du taux de compression. Il existe actuellement un consensus pour exploiter un préfiltrage des textures.

5.1.1 Préfiltrage symétrique: le *mip-mapping*

Une méthode fait référence [Williams83], il s'agit du *mip-mapping*[1]. Le principe de cette technique est simple: au lieu d'utiliser une seule table de texture, plusieurs sont stockées à différentes définitions, les plus petites étant obtenues à partir de l'image de départ en moyennant vers des résolutions plus faibles. On peut représenter cette technique par une pyramide d'images (*cf. figure 5.1 (a)*).

1. mip provient du latin *multum in parvo*, signifiant beaucoup de choses dans peu de place

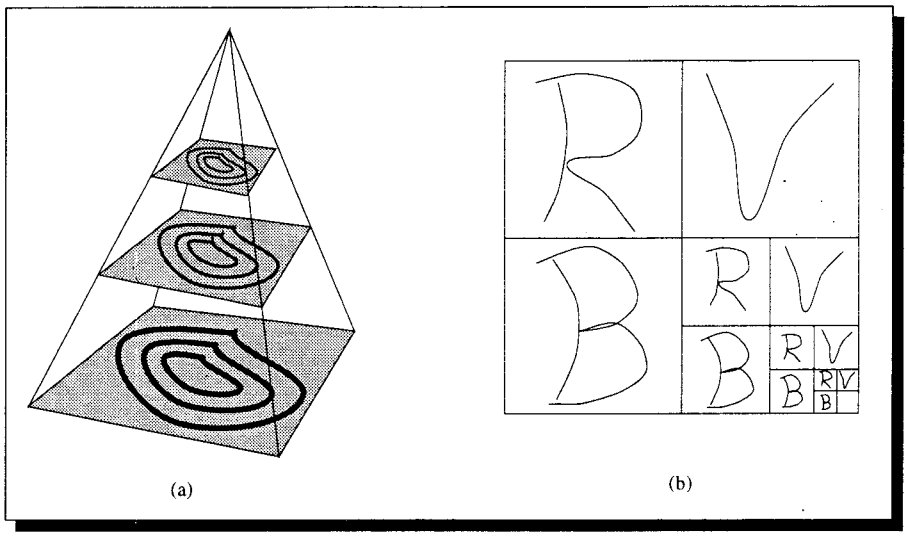


FIG. 5.1 - *Mip-Mapping : Pyramide et implémentation*

Après avoir calculé le taux de compression du pixel considéré (noté d), on identifie les *niveaux de mip-map* directement inférieur et supérieur. Dans chacun d'eux, on calcule la projection inverse du centre du pixel dans l'espace des texture (symbolisé par un point noir dans la figure 5.2). Le calcul de l'intensité d'un pixel se fait en interpolant entre les moyennes de quatre texels dans les taux de compressions directement inférieur et supérieur au taux de compression du pixel traité (*i.e.* interpolation bilinéaire dans le mip-map n et dans le mip-map $n+1$). Il reste à effectuer une interpolation linéaire entre le résultat des deux niveaux en fonction de la valeur de d. Cette technique est appelée filtrage tri-linéaire.

Comme le montre la *figure 5.1 (b)* la taille de la mémoire nécessaire est augmentée d'un tiers, surcoût faible en regard de l'amélioration apportée.

Cependant, le *mip-mapping* présente un défaut majeur : le filtrage est *symétrique*. Cela signifie que le taux de compression est supposé identique dans les directions x et y, alors que la mise en perspective peut engendrer des différences importantes suivant les axes. Afin d'éviter les moirés, on préfère générer du flou dans la direction la moins compressée en prenant comme taux de compression unique non pas la moyenne mais le taux le plus élevé. La photo cube2 a été calculée avec un mip-mapping.

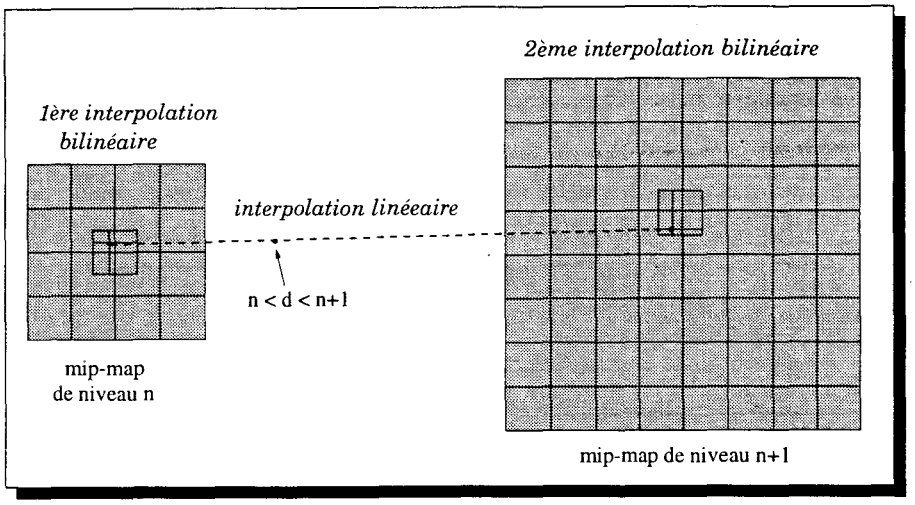


FIG. 5.2 - *L'interpolation trilinéaire du* mip-mapping

5.1.2 Le placage de texture câblé

Nous avons expliqué au §2.6.3 que le placage de texture (j'entends le calcul des coordonnées (u, v)) ne peut pas se faire par interpolation linéaire comme c'est le cas pour les calculs d'éclairement. Cependant, malgré les difficultés, plusieurs constructeurs ont proposé des solutions câblées pour effectuer le placage en temps réel.

Les premières machines ne disposant que d'interpolateurs linéaires, le placage de texture était réalisé en redécoupant finement les facettes texturées afin d'éviter les problèmes de déformation. Ensuite les interpolateurs quadratiques ont permis d'approcher un peu plus précisément les valeurs des coordonnées de texture. Enfin, les diviseurs câblés ont permis d'effectuer cette interpolation avec des coordonnées homogènes. L'étape suivante a été le préfiltrage des textures à l'aide du *mip-mapping*. Les premières machines utilisant cette technique ne faisaient qu'une seule interpolation bilinéaire (sur un seul niveau), on commence maintenant à trouver des stations implémentant une interpolation trilinéaire (*cf. ci-dessus*). La qualité obtenue par cette méthode (vrai *mip-mapping* tel que l'a proposé L. Williams) a un coût considérable, c'est pourquoi certains constructeurs ont choisi de ne câbler qu'un interpolateur bilinéaire et de diviser les performances par deux si l'utilisateur choisi un filtrage trilinéaire.

À titre d'exemple, et pour donner un ordre d'idée, prenons une machine qui voudrait afficher 1 Million de polygones texturés, avec une moyenne de 100 pixels par facette (c'est à peu près ce que propose la $RE2$ [2] actuellement). Le générateur de pixels doit avoir un débit de 100 Millions de pixels/s. Le *mip-mapping* nécessite, nous l'avons vu, 8 accès à la mémoire de texture, il faut donc fournir 800 Millions de texels/s. Si on considère que la mémoire de texture stocke des couleurs sur 3 octets (*rvb*), nous arrivons à une bande passante de $2.4GO/s$. Cela peut se résoudre de différentes façons, soit par l'utilisation de mémoire avec des technologies très performantes, soit en parallélisant les accès (il faut alors dupliquer les mémoires de textures). Dans tous les cas, le placage de texture correctement filtré est aujourd'hui ce qui coûte le plus cher dans les machines spécialisées.

2. $RE2$: Reality Engine 2

5.1.3 Préfiltrage asymétrique

Extension de la technique de la pyramide

[Gangnet et al.84] ont proposé une extension de la méthode de Williams avec un filtrage asymétrique qui évite la génération de flou dû à un surfiltrage dans une direction. Alors que le *mip-mapping* effectue un filtrage sur le plus petit carré englobant la projection inverse du pixel dans l'espace texture, cette méthode permet d'utiliser le plus petit rectangle afin de mieux respecter les différences entre les taux de compression en x et en y. Une table quatre fois plus grande que l'image de départ est utilisée; pour chaque niveau de compression dans une dimension, l'image est stockée à tous les niveaux dans l'autre direction (*cf. figure 5.3 (a)*). Sur la diagonale (partie grisée) on retrouve exactement la pyramide simple du *mip-mapping*. Pour la génération de la table, les auteurs proposent un filtrage passe-bas utilisant une fonction de pondération et un support de filtre très large au lieu d'une moyenne simple comme proposé par L. Williams.

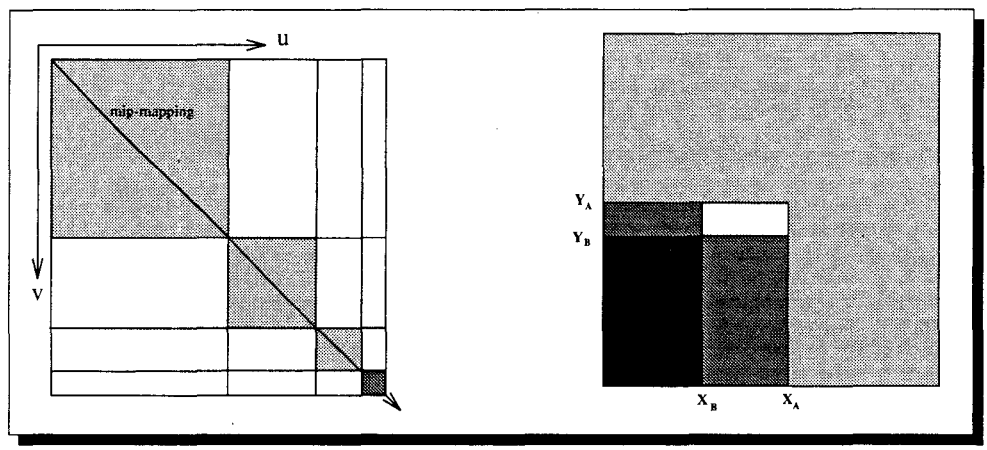


FIG. 5.3 - *Filtrage asymétrique*

Tables d'aires sommées

F. C. Crow [Crow84] a également proposé une amélioration de la méthode de Williams en permettant l'utilisation de zones rectangulaires dans l'espace de texture comme approximation de la projection inverse des pixels. La différence essentielle avec la méthode précédente tient dans le codage utilisé, une table d'aires sommées (*en anglais Summed Area Tables*) permet de représenter tous les niveaux de compression possibles de la texture au lieu d'un nombre arbitraire. Cette particularité évite d'effectuer une interpolation entre deux niveaux discrets, ce qui rend la méthode plus exacte. Dans une table de la même taille (en x et y) que la zone à texturer, chaque intensité est remplacée par une valeur représentant la somme des intensités de tous les texels contenus dans le rectangle défini par le texel considéré et l'origine de l'image de texture (*cf. figure 5.3 (b)*). À partir de ce codage, il devient facile de retrouver la somme des intensités de n'importe quel rectangle parallèle aux axes de la table de texture. L'intensité moyenne à travers un rectangle quelconque s'obtient en divisant cette somme d'intensités par la surface du rectangle. La somme des intensités d'un rectangle défini par les points A et B (respectivement coin supérieur droit et coin inférieur gauche) de coordonnées (x_A, y_A) et (x_B, y_B) s'obtient par la relation :

$$S = S_{(x_A,y_A)} - S_{(x_A,y_B)} - S_{(x_B,y_A)} + S_{(x_B,y_B)}$$

Si le *mip-mapping* est beaucoup utilisé lors d'implémentations matérielles, la méthode de Crow est très répandue dans les implémentations logicielles en particulier pour son efficacité. La *photo cube3* a été calculée avec cette méthode. On peut toutefois noter deux défauts de cette technique : d'une part, l'approximation de la projection inverse du pixel par un rectangle, bien que meilleure qu'avec un carré, reste peu précise en particulier lorsque la déformation n'est pas parallèle aux axes x et y ; d'autre part le codage *a priori* de la table sous la forme d'une somme d'intensités interdit le filtrage par une fonction plus efficace qu'une moyenne simple.

Enfin, remarquons que si les dimensions d'une table d'aires sommées sont celles du motif de texture, les nombres qui y sont stockés sont beaucoup plus grands que ceux de l'image (ce sont des sommes d'intensité). Ceci augmente considérablement la taille de la mémoire nécessaire, ce qui est un problème majeur dans certaines applications (la TVHD par exemple). Actuellement il n'existe aucune implémentation matérielle de cet algorithme.

5.1.4 Préfiltrage avec précision adaptative

Tout en étant meilleures que la représentation pyramidale, les techniques de filtrage asymétrique présentent encore des défauts de précision dans certains cas. En particulier, lorsque la déformation ne s'effectue pas parallèlement aux axes, la surface du rectangle englobant la projection inverse du pixel en est une mauvaise approximation. [Glassner86] propose une amélioration de la méthode de Crow qui tient compte de cet argument en utilisant un filtre dont la forme s'adapte à chaque projection inverse de pixel par approximations successives. En fait, cette méthode est très coûteuse et donc peu utilisée. Nous pouvons remarquer qu'elle reste inexacte, car d'une part elle cherche à approcher un quadrilatère qui n'est qu'une approximation de la projection des pixels, d'autre part le processus d'intégration qui sert à calculer la couleur du pixel ne devrait pas se faire de façon constante mais tenir compte d'une variation de densité à l'intérieur de la projection du pixel dans l'espace texture.

5.1.5 Conclusion sur le préfiltrage de textures

Les techniques de préfiltrage telles que nous venons de les décrire donnent de bons résultats et sont souvent utilisées en rendu projectif aussi bien dans les implémentations logicielles que matérielles. Cependant, elles sont incompatibles avec le lancer de rayons car le préfiltrage implique la connaissance du taux de compression de la texture pour l'objet traité ainsi que la taille de la projection inverse des pixels dans l'espace texture. Or, si on considère un rayon quelconque dans un arbre qui rencontre un objet texturé, ne connaissant que le point d'intersection, il est impossible de calculer la projection inverse du pixel dans l'espace texture en ne lançant qu'un seul rayon. D'autres techniques appropriées au lancer de rayons doivent être développées [Cook et al.84], en particulier le lancer de cônes ou le lancer de pyramides [Ghazanfarpour92]. Le sur-échantillonnage stochastique, que nous présentons plus loin (*cf.* §5.2.3), très utilisé en lancer de rayons, permet également d'antialiasser les textures.

5.2 Le sur-échantillonnage

5.2.1 Introduction

Comme nous l'avons montré durant le premier chapitre, les défauts que nous rencontrons, qu'ils relèvent de l'aliassage ou du crenelage, peuvent se résoudre soit en préfiltrant le signal, soit en augmentant la résolution d'échantillonnage, l'une et l'autre solution permettant de se rapprocher des conditions de Nyquist.

Nous venons de consacrer une section au préfiltrage des textures. Le préfiltrage de la scène, en vue de l'antialiassage des bords de polygones (*i.e.* l'échantillonnage surfacique), fera à lui seul l'objet de la *partie II* de ce document. Il nous reste donc à traiter des techniques dites de sur-échantillonnage. Très générales, ces méthodes traitent (avec plus ou moins de succès) l'ensemble des problèmes, c'est pourquoi nous les regroupons dans cette section.

Il existe en fait deux grandes catégories de méthodes de rendu. [Cook86] utilise les termes d'algorithmes analytiques et discrets pour différencier les techniques en fonction de leur approche de l'antialiassage. Les premiers éliminent les hautes fréquences avant l'échantillonnage (on parlera d'*échantillonnage surfacique*) alors que les seconds augmentent la résolution (le *sur-échantillonnage*).

Les deux méthodes sont concurrentes: actuellement dans les implémentations logicielles, il y a une dichotomie entre les solutions à base d'échantillonnage surfacique associé à des rendus projectifs (*exemple: A-Buffer, rapide et avec antialiassage de qualité, cf. §9.3*), et des implémentations de lancer de rayons (*réputées plus réalistes*) avec sur-échantillonnage. Lors d'implémentations matérielles, la simplicité de mise en œuvre et sa capacité à réutiliser l'existant font du sur-échantillonnage la seule méthode réellement câblée. Le but de cette thèse est justement de proposer une alternative à cette suprématie. Mais avant d'en arriver là, nous présentons les différentes techniques du sur-échantillonnage.

5.2.2 Le sur-échantillonnage régulier

Une approximation de la surface couverte peut être obtenue en augmentant la définition de la grille d'échantillonnage (*cf. figure 5.4*), ce qui donne plusieurs valeurs par pixels (des *sous-pixels*). Plus formellement, on définit le sur-échantillonnage comme la succession de trois étapes:

- Le signal continu est échantillonné à une résolution n fois supérieure à la résolution finale, ce qui crée une image virtuelle haute définition.
- L'image virtuelle haute définition est filtrée à l'aide d'un filtre passe-bas,
- L'image haute définition filtrée est rééchantillonnée à la résolution du dispositif d'affichage.

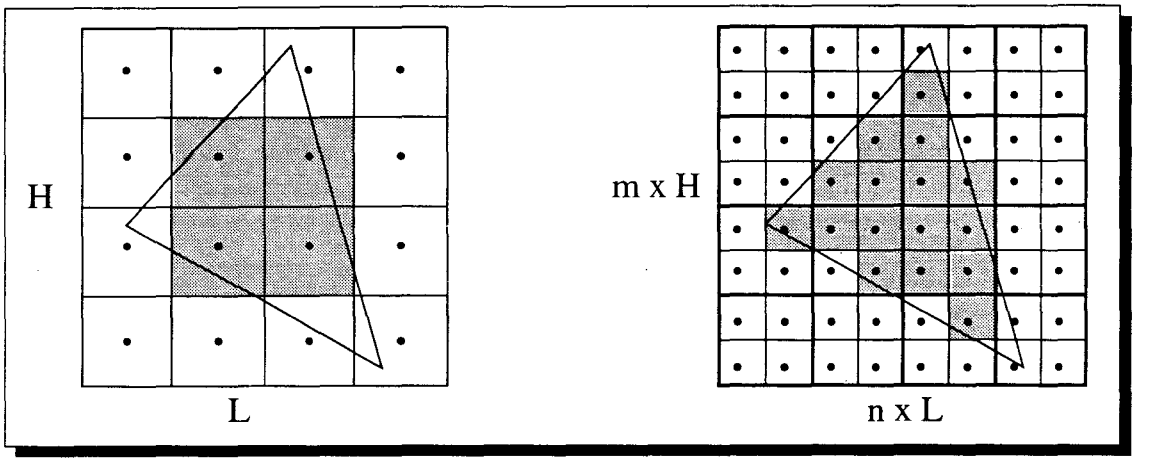


FIG. 5.4 - *Sur-échantillonnage régulier*

En fait, dans les différentes implémentations du sur-échantillonnage le filtrage passe-bas peut être soit effectué lors du rééchantillonnage, soit intégré dans la première phase en jouant sur la localisation des échantillons.

L'augmentation de la résolution s'effectue souvent de façon récursive : les pixels sont considérés comme des carrés que l'on subdivise en n sous-pixels réguliers. Cette technique permet d'adapter facilement le niveau de subdivision du pixel. Lorsque n est suffisament grand, la taille critique entraînant les défauts est repoussée, mais le problème n'a pas été résolu. Quelle que soit la valeur de n, toute grille d'échantillonnage régulier a une fréquence limite associée au delà de laquelle des phénomènes d'aliassage apparaîtront. La *figure 9.3* montre deux problèmes liés à l'échantillonnage ponctuel régulier. À gauche (*figure 9.3 (a)*), le dessin met en évidence le saut d'intensité qui se produit en animation quand un objet se déplace le long d'un axe. L'objet qui n'était pas présent dans le pixel se retrouve instantanément avec un taux de couverture de 1/4. À droite (*figure 9.3 (b)*), le polygone fin qui est dessiné compte pour 1/4 dans les pixels des deux extrémités et est absent du pixel central.

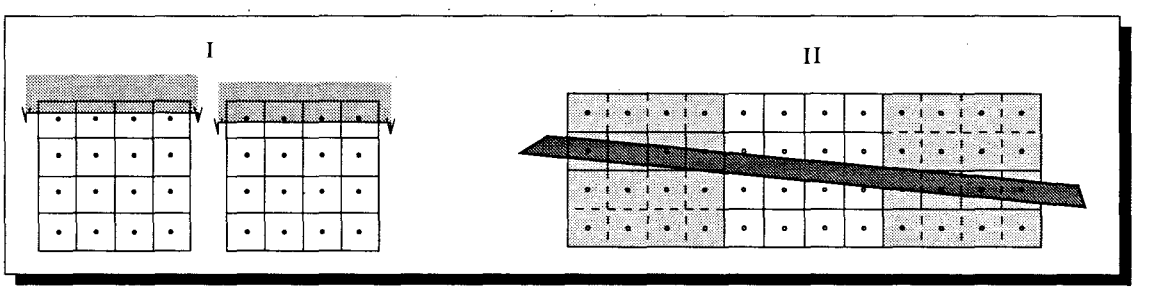


FIG. 5.5 - *Défauts du sur-échantillonnage régulier sur une grille* 4×4.

Les techniques sophistiquées d'échantillonnage stochastique peuvent être utilisées pour résoudre ce problème (*cf.* §5.2.3).

5.2.3 Répartition stochastique

En remplaçant les coordonnées régulières des échantillons par des valeurs aléatoires, le bruit corrélé dû à un échantillonnage régulier (aliassage) est remplacé par un bruit aléatoire,

celui-ci étant beaucoup mieux accepté par le système visuel humain. Le type de fonction aléatoire utilisé pour l'échantillonnage contrôle les caractéristiques spectrales du bruit. Il existe en pratique deux techniques, à savoir *la loi de distribution de Poisson* et la méthode de *perturbation aléatoire* (en anglais *jittering*). Une étude comparative détaillée de ces deux méthodes se trouve dans [Dippe et al.85]. Ces techniques développées pour le lancer de rayons ont été reprises par certains constructeurs pour l'antialiassage de bords de polygones.

Disque de Poisson La distribution de Poisson donne une répartition uniforme des échantillons. Ceux-ci s'obtiennent par *lancer de fléchettes* à l'aide d'une méthode de type *Monte Carlo*. On parle de disque de Poisson car une distance minimum est respectée entre deux valeurs. Des observations ont montré que la répartition des cellules photoréceptrices de l'oeil sur la rétine autour de la fovea[3] ressemble à une distribution de Poisson avec distance minimum [Yellot83]. Certains auteurs utilisent cet argument pour expliquer que l'oeil est très tolérant au bruit généré par cette technique.

Perturbation aléatoire (*jittering*) La méthode par perturbation aléatoire est une approximation de la technique précédente plus simple à mettre en œuvre. À partir d'une grille régulière, chaque point est *perturbé* de façon à l'éloigner aléatoirement de sa position initiale (*cf. figure 5.6 (b)*). Les valeurs de décalage doivent évidemment être différentes (et non corrélées) pour chacun des points de la grille. Par le calcul des transformées de Fourier, on montre que cette technique donne des résultats assez similaires (pour un coût beaucoup moindre) à une loi de distribution de Poisson avec distance minimum. Les photos sur la planche couleur 1 montrent quelques résultats comparatifs de sur-échantillonnage réguliers et stochastiques.

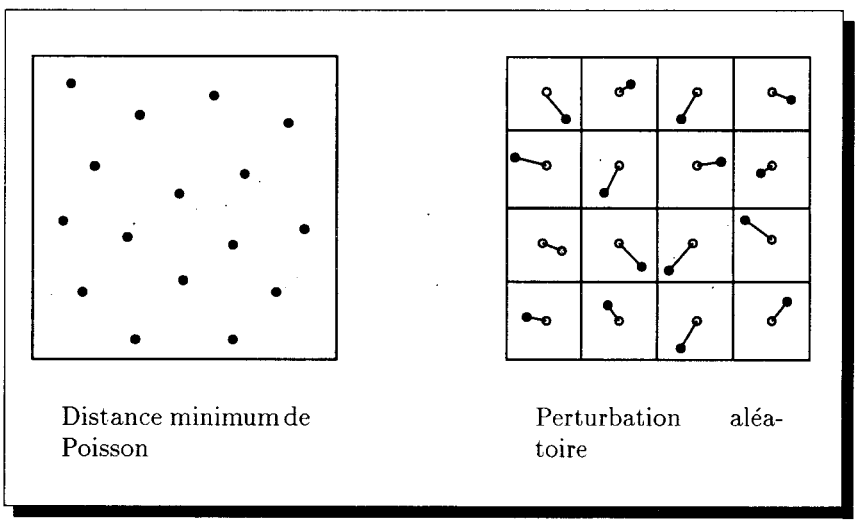


FIG. 5.6 - *Échantillonnage Stochastique.*

Filtrage par pondération

Les différentes techniques présentées jusqu'à présent calculent l'intensité finale des pixels en effectuant une moyenne simple sur les valeurs des sous-pixels. Ceci correspond à appliquer

3. À l'intérieur de la fovea, les récepteurs sont situés hexagonalement avec une densité deux fois supérieure à la fréquence maximale que le système Iris/lentille/Cornée laisse passer, le critère de Nyquist est donc satisfait.

un filtre passe-bas dont la forme est une fonction porte dans le domaine spatial sur la largeur du pixel. Un autre champ d'investigations visant à améliorer le sur-échantillonnage porte sur la pondération de la surface d'un objet en fonction de sa localisation à l'intérieur du pixel. L'idée est de donner plus d'importance au centre du pixel qu'aux zones périphériques, c'est-à-dire appliquer sur la scène un filtre spatial de type gaussien ou conique centré sur chaque pixel. Une des motivations principales est d'obtenir un déplacement plus fluide des objets. L'objet ne quitte pas brutalement le pixel, mais la couleur subit un dégradé. Il existe deux façons de mettre en oeuvre ce filtrage : soit en répartissant les échantillons de manière non uniforme (importance sampling), soit en pondérant les valeurs des échantillons répartis uniformément, en fonction de leur distance au centre.

De nombreux travaux ont porté sur la forme du filtre la mieux appropriée [Mccool et al.92] et sur la façon de générer les coordonnées des échantillons. Après une étude empirique sur les effets secondaires des filtres de reconstruction, [Mitchell87] explique qu'il doit y avoir un compromis entre des défauts d'anisotropie (les résultats sont différents selon les directions), de flou et des effets de *ringing* (vaguelettes autour des contours d'objets). [Mitchell et al.88] concluent que la fonction *sinc* n'est pas la meilleure à cause des défauts de ringing qu'elle génère, et qu'il faut lui préférer un filtre cubique (polynôme de degré 3).

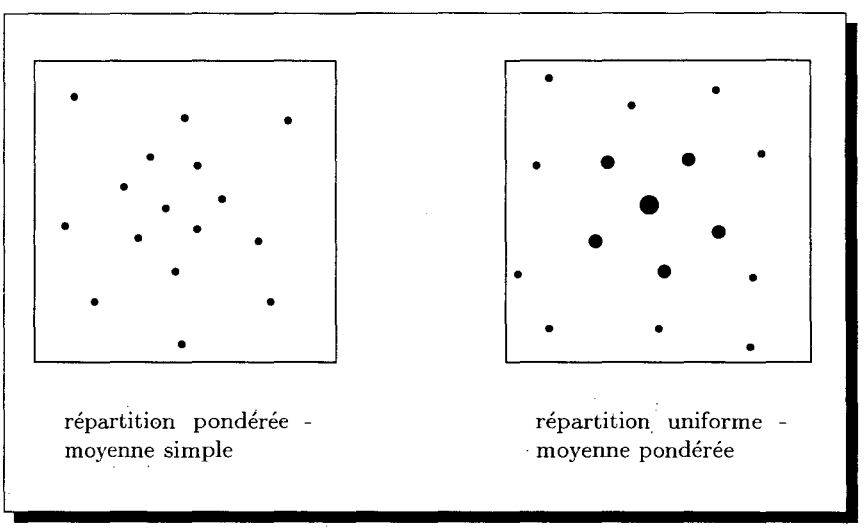


FIG. 5.7 - *Filtrage par pondération*

Réduire le nombre d'échantillons

Les études qualitatives des techniques que nous venons de présenter sont souvent effectuées en utilisant un grand nombre d'échantillons pour que les résultats soient significatifs. Dans la pratique, une même question se pose pour tous les algorithmes : « *Combien faut-il d'échantillons ?* » Les deux réponses classiques sont : « *Plus il y en a, mieux c'est* » et : *Nous pensons que n échantillons suffisent* », n pouvant varier de 4 à 256. Ce problème n'étant pas définitivement réglé, des travaux portent sur l'optimisation de la localisation des échantillons afin d'en diminuer le nombre. [Lee et al.85] et [Kajiya86] proposent d'augmenter le nombre d'échantillons en fonction de la variance des échantillons déjà tirés.

Ces techniques ont principalement été développées pour l'antialiassage des textures en lancer de rayons. Cependant, elles peuvent évidemment être utilisées pour répondre à d'autres

problèmes. L'antialiassage de bords de polygones, qui est un problème beaucoup plus simple à traiter (un sur-échantillonnage 3×3 régulier donne déjà de bons résultats) pourra être amélioré par des techniques stochastiques. Certaines machines proposent des méthodes de sur-échantillonnage pseudo-aléatoires pour traiter les bords de polygones et les objets fins (on sélectionne 8 sous-pixels dans une grille régulière de 64 positions). À qualité donnée, un choix judicieux des sous-pixels permet d'en diminuer le nombre. Par ailleurs, donner des poids différents aux pixels (répartition uniforme, moyenne pondérée) augmente sensiblement la qualité du résultat.

5.2.4 Le sur-échantillonnage dans les machines spécialisées

Comme son nom l'indique, le sur-échantillonnage nécessite la multiplication du processus de rendu par un facteur n. Pour cela, il existe deux écoles : la première consiste à utiliser une machine n fois plus grande en mémoire et plus puissante, c'est le sur-échantillonnage monopasse. Actuellement, seules les *RealityEngine* (et *RealityEngine2*) de *Silicon Graphics*, les « *Rolls Royce* » des machines graphiques, ont fait le choix de multiplier jusqu'à 16 fois la mémoire (couleur et profondeur), ce qui leur permet d'afficher des scènes antialiassées sans faire chuter leurs performances. La seconde idée est d'effectuer plusieurs passes sur une mémoire de même définition que l'afficheur : à chaque passe, le résultat est accumulé pour affiner la précision. Pour une puissance donnée, les performances baissent au fur et à mesure que la qualité s'améliore. Ce type de méthodes est regroupé sous le terme d'*Accumulation Buffer* dont nous donnons maintenant un petit historique.

Accumulation Buffer. Avec la définition de Pixel-Planes, [Fuchs et al.85] ont été les premiers à proposer une méthode à raffinement progressif utilisant l'accumulation. Les images antialiassées sont créées en répétant le rendu de la scène, avec à chaque passe, un décalage de sous-pixel. Une autre machine, *GSP-NVS* [Deering et al.88] (qui n'a jamais été réalisée), a repris cette technique en proposant d'utiliser la méthode de *perturbation aléatoire* pour les décalages de sous-pixels afin de réduire encore l'aliasssage. En 1989, A. Mammen a décrit la technique de la « *Virtual Pixel Maps* » de la machine *Stellar* [Mammen89] qui reprend l'idée d'une deuxième mémoire et l'utilise pour faire de l'antialiassage, mais aussi et surtout des effets tels que la transparence ou la composition d'images. La méthode a définitivement acquis ses lettres de noblesse depuis que *Silicon Graphics* l'a adoptée [Haeberli et al.90] sous le nom de *Accumulation Buffer*.

5.3 Traitement des problèmes liés à l'éclairement

Nous présentons dans ce paragraphe des méthodes spécifiques à l'antialiassage de l'éclairement spéculaire (*cf.* §4.4.1).

Échantillonnage adaptatif La première solution n'est rien d'autre qu'un sur-échantillonnage classique tel qu'il a été décrit auparavant. Comme pour les autres cas, les défauts dûs à l'aliassage peuvent être atténués mais ne disparaissent pas complètement. Il faut parfois augmenter considérablement le nombre d'échantillons pour que les défauts deviennent invisibles. Afin de minimiser le surcoût du sur-échantillonnage, il est possible d'utiliser une méthode *adaptative* qui n'augmente la fréquence d'échantillonnage que lorsque la variation de la normale à l'objet dépasse, dans un pixel, un seuil fixé [Crow82].

Utilisation de texture sphérique [Williams83] a proposé d'utiliser la technique du placage de textures. À partir d'une texture sphérique représentant l'éclairage depuis la scène, le calcul de l'éclairement en chaque pixel est remplacé par un accès à la table de texture indexée par les coordonnées de la normale.

Modification de l'exposant spéculaire [Amanatides92] fait une très bonne description du problème ainsi que de deux algorithmes. Le premier permet de détecter les cas où l'aliassage spéculaire va se manifester, l'autre est une solution par pré-filtrage pour traiter ces cas.

Se refusant à tout sur-échantillonnage pour le calcul de couleur par objet et par pixel, Amanatides détermine en fonction de la valeur de la puissance du cosinus dans la composante spéculaire de la formule de Phong, un seuil de variation de la courbure d'un objet dans un pixel.

Fonction d'illumination de Phong :

$$I = K_{spec} cos^n(\alpha)$$

En examinant la transformée de Fourier de cette fonction, on s'aperçoit que les hautes fréquences croissent en même temps que n. Pour une valeur du paramètre n donnée (correspondant à un coefficient de réflectance particulier), on peut en déduire un seuil de variation de la normale entre deux échantillons $\Delta\alpha_n$ ($\Delta\alpha_n = \pi/n$) au-dela duquel l'aliassage va se manifester. Les cas critiques sont détectés lorsque dans un pixel, la variation de la normale d'un objet engendre une variation de l'angle α supérieure au seuil ($\Delta\alpha > \Delta\alpha_n$).

Lorsque ce cas est détecté, au lieu de sur-échantillonner comme le proposait Crow, Amanatides élimine les hautes fréquences en diminuant la valeur de n, la puissance du cosinus. Ce qui a pour effet de diminuer le seuil de variation de normale acceptable. En effet, plus n est grand, plus le spéculaire sera étroit et donc plus il existera de hautes fréquences. Cette méthode s'accompagne d'une normalisation de la fonction d'éclairement afin de pas modifier les basses fréquences. Cette solution est donc une méthode de préfiltrage du signal de départ. Les hautes fréquences sont éliminées avant l'échantillonnage, ce qui a pour conséquence la génération de flou (la tache spéculaire est un peu plus large et moins marquée). La *figure 5.8* montre un exemple de cette technique.

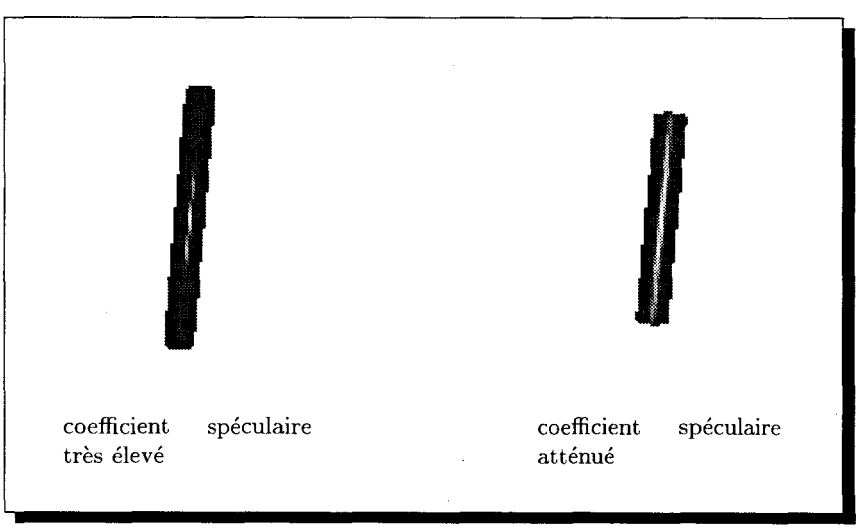


FIG. 5.8 - *Aliassage spéculaire*

5.4 L'antialiassage temporel

Comme pour l'aliassage spatial, les effets des deux types de défauts (aliassage et crénelage) diminuent lorsqu'on augmente la fréquence des images.

Motion Blur (ou flou de bougé) L'effet de saccade dans les déplacements peut être éliminé à l'aide d'un préfiltrage temporel, c'est la technique du **motion blur** [Potmesil et al.83]. On peut par exemple multiplier par deux la fréquence et composer les images obtenues deux à deux pour calculer les images finales. Des filtres plus complexes peuvent être utilisés [Shinya93], il faudra toujours faire un compromis entre les phénomènes de *strobbing* et le *flou de bougé.*

Vitesse de déplacement/fréquence d'animation Lorsqu'un mouvement est trop rapide (vitesse supérieure à la fréquence d'animation), il faudrait pouvoir en éliminer les hautes fréquences. Cela reviendrait soit à supprimer ce mouvement, soit à le ralentir, ce qui dans les deux cas n'est pas acceptable. La seule issue est donc cette fois, d'augmenter la fréquence d'animation, sachant que l'on peut potentiellement toujours atteindre les conditions de Nyquist car il n'existe pas de vitesse infinie.

Antialiassage spatio-temporel

Le Cinéma Au cinéma, le grain du film est suffisament fin pour considérer chaque image comme un signal continu dans les deux dimensions. Un film est une séquence de ces images, on peut donc le traiter comme un signal tridimensionnel continu dans le plan 2D et discrétisé dans le temps.

La Vidéo A cause du balayage du spot, le signal vidéo n'est continu que horizontalement : dans les directions verticale et temporelle il est discrétisé. De plus, l'aliassage spatio-temporel est accentué par la technique de la **vidéo entrelacée** qui affiche alternativement les lignes paires et les lignes impaires d'une image. Un effet de *scintillement* désagréable se produit sur l'écran nécessitant un filtrage approprié. Dans [Amanatides et al.90], on trouve une étude des différents filtres possibles pour compenser les effets indésirables dans ce cadre précis.

Continuité temporelle affichage/animation La technique de l'affichage entrelacé pose un premier problème de discontinuité en n'affichant que la moitié d'une trame à chaque image. Afin d'y remédier on peut calculer des images à 50 Hz (au lieu de 25 Hz) et composer, pour chaque couple, les lignes impaires de l'une avec les lignes paires de l'autre. On crée ainsi une séquence à 25 Hz.

Continuité spatiale Des images calculées par sur-échantillonnage que l'on vient de créer, on ne garde que la moitié des lignes pour chacune d'elles, ce qui engendre une perte d'informations spatiales. Il faut donc, pour chaque ligne, prendre en compte les lignes immédiatement inférieure et immédiatement supérieure. Ainsi on calculera une ligne impaire en la composant avec les lignes paires voisines. Ce filtrage permet de garder toutes les informations spatiales pour chaque image de l'animation. Cette méthode donne d'excellents résultats, y compris pour des animations lentes et le long des axes [Dumas94].

Deuxième partie

L'échantillonnage surfacique

Chapitre 6

Le principe

À l'opposé de l'échantillonnage ponctuel, l'échantillonnage surfacique considère les pixels comme des surfaces élémentaires et non plus comme des points. Les objets ne sont alors plus simplement présents ou absents d'un pixel, mais ils peuvent le recouvrir partiellement. Cela signifie que les pixels, qui sont des entités technologiques indivisibles affichant une couleur unique, doivent représenter le *mélange* des différentes couleurs visibles pondérées par leur occupation du pixel. Du point de vue du traitement de signal, cela revient à appliquer sur la scène un filtre passe-bas dont la forme et la taille correspondent à la définition du pixel.

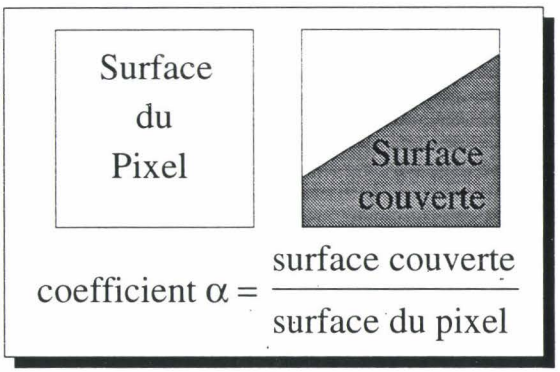


FIG. 6.1 - *Définition du taux de couverture*

6.1 Échantillonnage surfacique *vs* échantillonnage ponctuel

Les algorithmes de rendu à échantillonnage ponctuel peuvent se résumer en trois étapes : 1/ déterminer si l'objet traité est présent dans le pixel traité (*i.e.* le centre du pixel est-il couvert par la projection de la primitive ?) 2/ effectuer l'élimination des parties cachées 3/ calculer une valeur de couleur, considérée constante sur la surface du pixel, obtenue à partir des coordonnées du centre du pixel (le plus souvent par interpolation). L'objet qui gagne au grand jeu de l'élimination des parties cachées, fournit la couleur qui sera affichée sur le moniteur. Lorsque la scène est sur-échantillonnée, toutes ces étapes se déroulent pour chacun des *sous-pixels*, la couleur finale des pixels (celle que l'on affiche) est alors calculée en effectuant la moyenne pondérée des couleurs des différents sous-pixels.

Les choses se compliquent avec l'échantillonnage surfacique : chacune des étapes décrites ci-

dessus doit être modifiée. La détection de la présence d'un objet dans un pixel se transforme en un calcul d'une grandeur communément appelée la **composante** α (entre 0 et 1) [1] permettant de mesurer la proportion de surface couverte. Ensuite, le calcul de la couleur d'un objet dans un pixel peut être générateur de débordements liés au fait que certains objets couvrent moins de la moitié du pixel, alors que le calcul d'intensité s'effectue avec les coordonnées du centre (c'est-à-dire en dehors de la primitive). Enfin, l'élimination des parties cachées devient beaucoup plus complexes qu'avec un échantillonnage ponctuel à cause des éventuels recouvrements partiels à l'intérieur du pixel. Lorsque l'objet le plus proche de l'observateur ne couvre pas entièrement le pixel, cela signifie que plusieurs objets sont visibles et que leur couleur respective doit être prise en compte.

La simple information quantitative donnée par le coefficient α n'est d'ailleurs pas suffisante pour tenir compte correctement des éventuels recouvrements de primitives à l'intérieur du pixel. Plusieurs techniques ont été proposées pour ajouter des informations de type géométrique, de manière à préciser la localisation dans le pixel de la partie couverte. De la simple valeur d'orientation à l'utilisation de *masques de sous-pixels*, différentes méthodes ont été élaborées, nous les décrivons dans les chapitres suivants.

Cependant, elles ont toutes pour point de départ le calcul d'un taux de couverture de l'objet dans le pixel (la composante α), nous détaillons maintenant les diverses possibilités pour ce calcul.

Terminologie

Il faut distinguer clairement deux aspects fondamentalement différents : la conversion des objets en pixels et l'élimination des parties cachées. Le terme d'échantillonnage surfacique ne devrait en fait recouvrir que la partie *conversion des objets en pixels*. Cette étape diffère essentiellement d'un échantillonnage ponctuel par la prise en compte de l'aspect surfacique des pixels. On a l'habitude d'appeler *composante* α la proportion de surface du pixel couverte par la projection d'une facette. L'échantillonnage surfacique devrait donc se résumer à cette partie. Or, comme nous l'avons dit précedement, ce type d'échantillonnage impose un algorithme spécifique d'élimination des parties cachées, cette étape qui n'a pourtant rien à voir avec un échantillonnage, est souvent comprise dans le vocable *échantillonnage surfacique*.

6.2 Calcul de la composante α

Le calcul de la composante α dépend bien entendu de la méthode utilisée pour l'affichage et le remplissage des polygones. Nous distinguons deux familles d'algorithmes : les méthodes de remplissage par suivi de contours et celles qui utilisent les tests d'inclusion (*cf.* §*2.3.2*). Nous détaillons maintenant pour chacune de ces méthodes, les techniques possibles pour le calcul de la composante α.

6.2.1 Calcul d'α avec remplissage par contour-segments

Avec ce type d'algorithmes, initialement décrits dans le cadre de l'échantillonnage ponctuel (*i.e.* sans antialiassage), si on veut effectuer un calcul de taux de couverture (ce qui concerne

1. $\alpha = 0$ signifie que la primitive ne touche pas le pixel, $\alpha = 1$ quand le polygone le couvre entièrement, tandis que $\alpha = 0.5$ si le pixel est à moitié couvert.

uniquement les pixels de bord), on doit le faire au moment du tracé incrémental des segments de contour, ce qui nous ramène au problème du tracé de segments antialiassés.

6.2.2 Le tracé de segments antialiassés

Deux algorithmes concurrents existent pour l'affichage de segments en infographie. Ces algorithmes ont été souvent décrits, il s'agit de l'algorithme dit *de Bresenham* et du *DDA* [Foley et al.90]. Pour les segments 2D, *Bresenham* est encore préféré car il travaille avec une arithmétique entière. Par contre, pour le calcul de segments 3D qui nécessite une interpolation en z, l'algorithme de *Bresenham* devient inutilisable (l'argument vaut également pour l'interpolation sur les composantes R, V, B) [Swanson et al.86]. Le *DDA* est alors utilisé, le plus souvent en virgule fixe pour des raisons de rapidité. Les algorithmes 2D que nous listons maintenant sont tous dérivés de Bresenham. Cependant, le DDA qui est plus utilisé quand l'algorithme est câblé, peut être adapté pour exploiter les mêmes idées.

- En 1980, [Pitteway et al.80] ont proposé une extension directe du *Bresenham* en utilisant la valeur du paramètre de décision pour pondérer l'intensité du pixel. Cet algorithme, bien que très simple, présente plusieurs défauts : tout d'abord comme il est basé sur le tracé de *Bresenham*, un seul pixel par ligne ou par colonne (suivant l'octant) est traité, alors que plusieurs peuvent être traversés par le segment réel (*cf. figure 6.2*), ce qui minimise l'effet d'antialiassage (transition brutale entre certains pixels). Par ailleurs, et ceci est beaucoup plus gênant, les extrémités des segments (ou les sommets des polygones) doivent être traités comme des cas particuliers. Chaque cas (position du sommet dans le pixel + pente du segment) doit faire l'objet d'une entrée spécifique dans une table d'indirection donnant le coefficient qui permet d'atténuer l'intensité du pixel. Une telle table peut prendre une taille considérable si l'on veut une bonne précision.

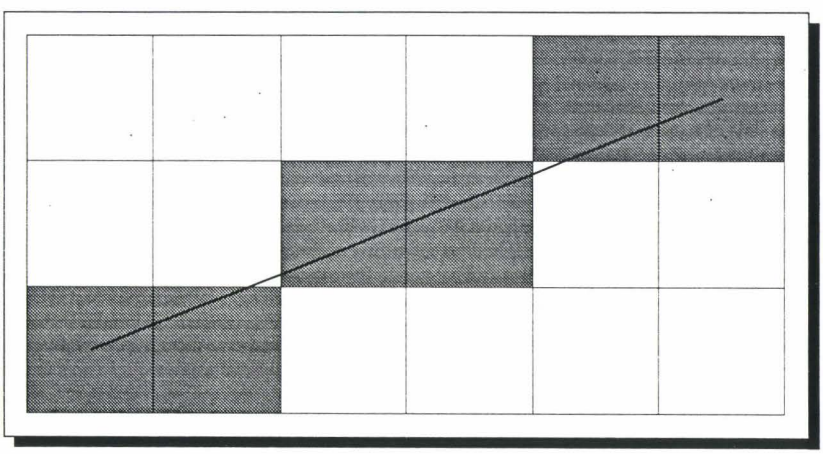


FIG. 6.2 - *Les différents cas possibles d'intersection d'un segment avec les pixels : seuls les pixels coloriés (correspondant au tracé de Bresenham) sont traités par l'algorithme et donc antialiassés, alors que deux autres pixels sont traversés par le segment (exemple dans le premier octant).*

- L'année suivante [Gupta et al.81] ont présenté une méthode similaire mais qui utilise une distance orthogonale au segment plutôt que la distance verticale, ce qui permet

de garder une intensité constante quelle que soit l'orientation du segment. Par ailleurs, le support de filtre utilisé est conique, ce qui donne également de meilleurs résultats qu'avec une moyenne simple comme dans la méthode précédente. Néanmoins, comme pour l'algorithme de Pitteway et Watkinson, le problème majeur reste le traitement des sommets de segments comme des exceptions (voir ci-dessus).

- La méthode de [Wu91] est tirée de celle de [Fujimoto et al.83] qui est un peu moins efficace. Deux idées intéressantes la caractérisent, la somme des intensités des pixels concernés par colonne est constante, et la symétrie par rapport au milieu du segment est utilisée pour accélérer le tracé.

- T. Lindgren [Lindgren90] a également proposé un algorithme d'antialiassage de segments. Mais on notera surtout [Lindgren et al.93] qui est une méthode permettant d'évaluer de manière qualitative, en fonction de paramètres psycho-visuels, les différents algorithmes.

On peut se référer à [Ghazanfarpour85] pour une étude comparative des trois premiers de ces algorithmes.

Toutes ces techniques peuvent être modifiées pour prendre en compte le fait que dans l'affichage de polygone, il ne faut plus considérer des segments avec une épaisseur mais des bords de demi-plans. Au moment du tracé des segments de contour, les pixels concernés sont donc calculés avec une composante α obtenue par l'un de ces algorithmes.

6.2.3 Calcul d'α avec un remplissage par test d'inclusion

La méthode que nous avons décrit dans la première partie de ce manuscrit au §2.3.2 permet de dire si le centre d'un pixel de coordonnées (x_i, y_i) se trouve à l'intérieur ou à l'extérieur d'un triangle. Il s'agit maintenant de déterminer si une des droites délimitant une facette, traverse le pixel, auquel cas il faut calculer la proportion de surface couverte (*i.e.* la composante α). Si $f(x, y) = 0$ est l'équation d'une droite, $f(x_i, y_i)$ est une grandeur proportionnelle à la distance euclidienne séparant le point (x_i, y_i) de cette droite. Il suffit donc de normaliser les coefficients de la droite pour obtenir une valeur exploitable et déterminer un *seuil* en deçà duquel on considère que la droite se trouve à moins d'un demi-pixel du point (x_i, y_i). La valeur $f(x_i, y_i)$ permet alors de déduire directement la composante α. Nous renvoyons le lecteur à la dernière partie de ce manuscrit (*cf.* §*14.1*) pour de plus amples détails sur cette étape, nous discutons en particulier de la taille et de la forme du pixel, ainsi que des étapes de normalisation du calcul de la composante α.

6.2.4 Cas des sommets de polygones

Dans ce qui vient d'être dit, nous n'avons considéré que les bords de polygones, c'est-à-dire les cas où une seule droite traverse le pixel. Aux sommets des triangles et dans le cas des facettes de taille inférieure au pixel, il existe deux ou trois valeurs d'α qu'il faut combiner pour obtenir la composante α de l'objet et non plus la surface couverte par un demi-plan.

Remarquons que les méthodes de remplissage par suivi de contours traitent toutes les sommets de segments comme des cas particuliers, les sommets de polygones (ou les polygones de taille inférieure au pixel) sont par conséquent très difficiles à traiter correctement. Sauf à les considérer eux aussi comme des cas particuliers (et il y en a beaucoup), il n'existe pas

de méthode simple (*i.e.* sans passer par des calculs trigonométriques couteux) permettant de calculer de manière exacte la composante α des sommets de polygones à partir des coordonnées des sommets et des pentes des droites.

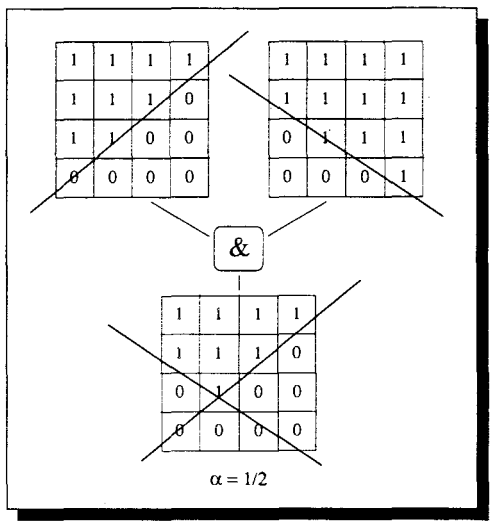


FIG. 6.3 - *Calcul d'α par combinaison de masques*

Depuis [Fiume et al.83] la solution consiste donc à utiliser des masques de sous-pixels pour transformer une information quantitative: α, en une information géométrique au niveau du sous-pixel, et ce quelle que soit la méthode d'affichage (contour-segment ou tests d'inclusion). Différentes techniques ont été proposées [Carpenter84], [Abram et al.85], [Schilling91]. Elles se caractérisent toutes par la conversion des différentes valeurs d'α en masques de bits, la pente de la droite étant utilisée pour déterminer les bits de valeur 1 (par convention un sous-pixel vaut 1 si le demi-plan le couvre, 0 sinon). Ces masques sont ensuite combinés à l'aide d'opérateurs logiques pour connaître le masque final (*cf. figure 6.3*). De ce masque final, on tire la couverture totale du polygone dans le pixel en comptant le nombre de bits valant 1. Cette technique donne une précision dépendante de la résolution du masque. Nous verrons par la suite que cette technique du masque de sous-pixel peut également être utilisée pour effectuer l'élimination des parties cachées entre les différents polygones présents dans un pixel.

6.3 Le *blending* (mélange des couleurs)

Les difficultés apparaissent dès que l'on veut afficher dans un pixel une couleur résultant du mélange de couleurs de différents objets. Cette opération, essentiellement propre à l'*antialiassage* (pré-filtrage des textures, résultat d'un sur-échantillonnage, α-blending) revient en fait à effectuer une *moyenne* de ces différentes couleurs. Ce moyennage (ou interpolation) se fait dans le système de représentation utilisé pour le calcul de couleur, nous discutons maintenant de la pertinence des différents modèles.

6.3.1 Dans le modèle *RVB*

Nous ne discuterons pas ici de la fidélité du modèle à reproduire les couleurs réelles, il a de toute façon était montré qu'un système trichromatique était insuffisant et qu'il fallait

augmenter le nombre de composantes (en terme de longueur d'onde) pour représenter les couleurs. La question du meilleur modèle (voire du modèle réel) reste ouverte. Par contre, nous allons nous intéresser aux faiblesses du cube *RVB* dans le cadre précis de l'antialiassage.

Comme nous l'avons vu plus haut, quelle que soit la méthode utilisée, antialiasser une scène revient à prendre en compte plusieurs couleurs calculées indépendamment les unes des autres, car représentant différents objets visibles en un même pixel. En niveaux de gris, cela ne pose aucun problème, les valeurs représentent des intensités et il suffit de faire la moyenne de celles-ci en les pondérant par la place qu'elles occupent à l'intérieur du pixel. Ainsi un objet blanc couvrant un pixel à 75 % sur un fond noir donnera un pixel gris clair d'intensité 0.75 si l'échelle se trouve entre 0 et 1. En *RVB*, il est courant d'utiliser cette technique séparément sur chacune des trois composantes, en faisant l'hypothèse qu'elles représentent des niveaux d'intensité dans leur couleur primaire. Le calcul de la couleur finale d'un pixel s'effectue donc par *moyennage* des couleurs présentes. Prenons l'exemple d'un pixel se trouvant à la jonction de trois objets, l'un entièrement rouge, l'autre bleu et le troisième vert, ces trois objets participent à proportion égale à la couleur du pixel (leur couverture est de 1/3 chacune). La couleur du pixel sera obtenue par :

$$Rouge_{pixel} = \frac{Rouge_{objet_1} + Rouge_{objet_2} + ... + Rouge_{objet_n}}{n}$$

$$Bleu_{pixel} = \frac{Bleu_{objet_1} + Bleu_{objet_2} + ... + Bleu_{objet_n}}{n}$$

$$Vert_{pixel} = \frac{Vert_{objet_1} + Vert_{objet_2} + ... + Vert_{objet_n}}{n}$$

ce qui donne ici :

$$Rouge_{pixel} = \frac{1+0+0}{3}$$

$$Bleu_{pixel} = \frac{0+1+0}{3}$$

$$Vert_{pixel} = \frac{0+0+1}{3}$$

c'est-à-dire un gris foncé, alors qu'on devrait s'attendre à un blanc éclatant. Cet exemple est le cas le plus défavorable et se produit rarement, il a le mérite de montrer que l'interpolation simple dans le cube RVB engendre une baisse d'intensité sur la couleur résultante. Ainsi, on observe facilement que la frontière entre deux objets de forte intensité mais de couleur complémentaire, se traduit à l'écran après antialiassage par cette méthode, par une ligne certes lissée, mais plus sombre que les objets se trouvant de part et d'autre.

Notons que ce qui vient d'être expliqué avec l'antialiassage de bord de polygone reste vrai dans le cas des filtrages de textures (par sur-échantillonnage ou par des méthodes de pré-filtrage). Les textures compressées ayant subi des moyennages peuvent présenter des baisses d'intensités.

6.3.2 Dans le modèle *TIS*

Lors d'une interpolation entre deux couleurs dans l'espace *TIS*, l'intensité est prise en compte et le défaut décrit ci-dessus n'apparaît plus. De ce point de vue, le modèle *TIS* est mieux adapté que le modèle *RVB*, pourtant la lourdeur des conversions entre ces différents espaces de représentation est rédhibitoire pour les applications recherchant la rapidité.

6.3.3 Conclusion

Si l'on prend comme hypothèse que la couleur calculée en un pixel pour un objet de la scène est exacte, l'opération d'antialiassage qui consiste à moyenner différentes couleurs parce qu'on ne sait en afficher qu'une par pixel est totalement artificielle. Elle dépend bien évidement de l'espace de représentation utilisé. Cette opération n'existe pas dans la nature, il n'est par conséquent pas concevable qu'un quelconque modèle donne *la bonne* représentation de quelque chose qui n'existe pas. Il reste néanmoins que le résultat visuel et subjectif (comme souvent en antialiassage) met en évidence une faiblesse du modèle RVB, que l'on peut résoudre en passant par un autre modèle (par exemple TIS).

Chapitre 7

Algorithmes de composition d'images (α-blending)

Nous présentons maintenant une première technique d'antialiassage de bords de polygones à partir d'un échantillonnage surfacique.

7.1 Le principe

Les algorithmes de composition d'images utilisent uniquement les coefficients d'occupation, aucune information géométrique supplémentaire n'est calculée. Le principe est simple, pour tous les pixels utiles, les composantes *RVB* de chaque objet sont accompagnées d'une valeur α représentant la proportion de surface occupée dans le pixel par l'objet. À chaque fois qu'un objet est présent dans un pixel, sa couleur est *mélangée* avec celle précédemment stockée dans le pixel au *prorata* du coefficient α. Nous allons voir que cette méthode donne de très bons résultats dans certains cas simples mais présente des incohérences sur des scènes plus complexes.

7.2 Les algorithmes

Dès que plusieurs objets se recouvrent dans un pixel, la seule information sur la composante α d'un pixel, ne suffit pas à traiter correctement les recouvrements. En effet, en plus de l'information numérique, il faut connaître une information de type géométrique précisant la zone du pixel couverte afin de combiner deux objets le couvrant partiellement.

[Porter et al.84] ont proposé un algorithme utilisant des opérateurs spécifiques qui permet de combiner la couleur d'un nouveau pixel avec celle déjà stockée, en tenant compte de leur coefficient α respectif. Les auteurs font preuve d'un grand optimisme en prenant comme hypothèse que si deux objets couvrent partiellement un même pixel avec des taux de couverture respectif α_1 et α_2, alors l'objet le plus avant couvre celui de derrière avec le même taux que pour le pixel entier. Cela signifie que les cas de recouvrement total ou d'absence de recouvrement ne sont pas traités. Outre les imprécisions dans les cas favorables, cette technique est incapable de gérer les cas de polygones adjacents de couleur différente et, (dans certains cas) fait intervenir des couleurs qui ne devraient pas être visibles.

[Duff85] a voulu amélioré la version précédente en ajoutant un opérateur (appelé **comp**) qui permet de détecter les arêtes implicites et de les traiter. Pour cela, l'auteur propose de

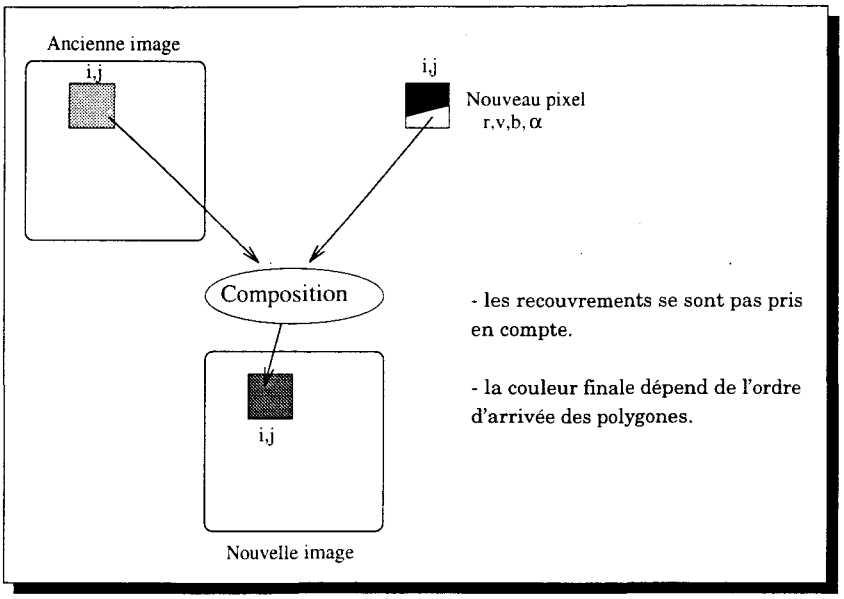


FIG. 7.1 - *La composition d'images*

comparer pour les deux primitives traitées, leur profondeur à chaque coin du pixel. Le signe de ces comparaisons permet de distinguer parmi 16 cas possibles. Lorsque cela correspond à un cas d'arête implicite, une approximation des taux de couverture respectifs est obtenue par interpolation des valeurs de profondeur aux quatre coins du pixel. L'auteur a visiblement voulu prendre en compte des problèmes spécifiques à la synthèse d'images 3D, alors que la méthode était initialement développée pour le dessin animé assisté par ordinateur. Il reste que cette méthode souffre des mêmes défauts que la précedente, elles sont toutes deux contraintes à ne traiter que des primitives préalablement triées en profondeur, sinon de graves défauts de visibilité peuvent intervenir.

Nous retiendrons de ces méthodes qu'elles sont bien adaptées par exemple, à l'incrustation d'images de synthèse dans une image réelle (photo ou film). Après numérisation, la photographie est prise comme image de fond et l'incrustation d'un objet *de synthèse* se fait par une de ces techniques de composition d'images. Les bords de l'objet rajouté sont alors correctement antialiassés avec le fond réel. Dans cette situation, d'une part peu de plans existent, et d'autre part l'ordre de profondeur est connu *a priori.*

Cette technique a peu à voir avec la synthèse d'images 3D, et les approximations des méthodes de composition d'images les rendent inefficaces dans un contexte général.

Remarquons pour l'anecdote que c'est depuis [Porter et al.84] que le taux de couverture d'une primitive dans un pixel s'appelle le coefficient α (*cf. figure 6.1*), cette notation est devenue un standard.

7.3 α-blending sur les segments

Bien qu'en marge du contexte dans lequel nous avons effectué notre travail, il est intéressant de citer le cas de l'affichage *fil de fer* très utilisé en C.A.O. et qui représente actuellement la principale utilisation des méthodes de composition d'images. En effet, de nombreuses ma-

chines spécialisées proposent ce type d'affichage avec des segments antialiassés, en temps réel. La méthode utilisée est toujours la composition d'images. En représentation fil de fer, il n'y a pas d'élimination des parties cachées, tout au plus l'élimination des faces mal orientées (en anglais : *back face culling*). Les segments sont calculés à l'aide d'un des algorithmes cités plus haut de tracé de segments antialiassés. En général, les segments sont affichés sur un fond uniforme, le blending s'effectue donc entre la couleur du segment et la couleur du fond. Quand deux traits se coupent, un *blending* est effectué avec la technique de composition d'images telle que nous l'avons décrite plus haut, l'ordre d'affichage ayant peu d'influence sur le résultat. Cette technique donne d'ailleurs de très bons résultats.

Chapitre 8

Les méthodes à voisins

Nous décrivons dans ce chapitre une autre catégorie de méthodes d'antialiassage par échantillonnage surfacique.

8.1 Le principe

L'échantillonnage surfacique nécessite de prendre en compte dans les pixels de bords plusieurs couleurs d'objets. Les algorithmes de composition d'images traitent cette question en mélangeant *au vol* les couleurs des primitives présentes en chaque pixel. Les éventuels recouvrements ne sont pas pris en compte et certains objets normalement non visibles, car cachés par d'autres plus proches de l'observateur, peuvent participer à la couleur finale du pixel.

Les méthodes que nous appelons *méthodes à voisin* résolvent cette question en différant le calcul de la couleur finale et en ajoutant une information géométrique relative à la pente de l'arête qui traverse le pixel. Pendant l'élimination des parties cachées, on utilise un simple z-buffer afin de ne garder que le fragment d'objet le plus proche. Lorsque toutes les primitives ont été traitées, une seconde passe exécute le calcul de la couleur finale en utilisant la couleur de l'objet le plus proche pondérée par sa composante α et la couleur d'un des huit pixels voisins comme couleur complémentaire. Ce dernier est choisi en fonction de la pente qui a été stockée. L'hypothèse qui est faite ici est que ce voisin doit posséder soit la couleur de fond du pixel courant (dans le cas d'un bord d'objet sur un fond uniforme), soit la même couleur que celle d'un polygone adjacent le cas échéant.

A priori les recouvrements ne sont pas pris en compte puisqu'une seule couleur est conservée dans le pixel, cependant l'utilisation d'une information d'orientation, donnée par la pente, permet de considérer comme seconde couleur, celle d'un *voisin* qui devrait contenir la même couleur que la partie non recouverte du pixel.

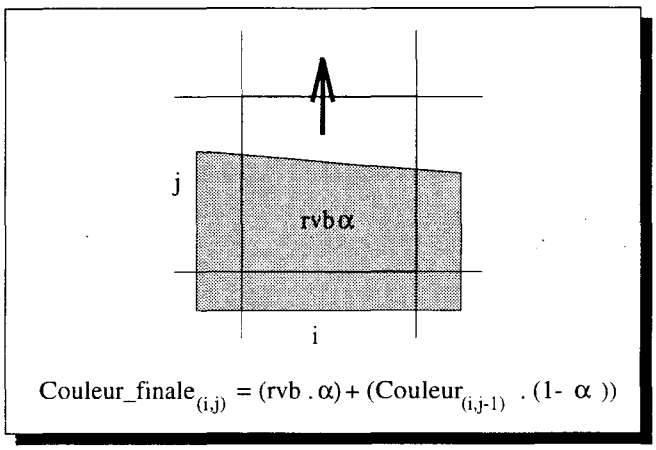


FIG. 8.1 - *Les méthodes à voisin*

8.2 Le choix du voisin

Le *voisin* d'un pixel traversé par une droite, se trouve dans la direction perpendiculaire à la droite et à l'extérieur du polygone. C'est la couleur de ce pixel qui est utilisée comme couleur complémentaire au moment du calcul de la couleur finale. Le choix parmi les huit pixels voisins peut s'effectuer soit lors de l'échantillonnage en conservant une information géométrique (*i.e.* la pente de la droite qui traverse le pixel), soit en stockant la composante α sous la forme d'un masque de sous-pixels. Il faut alors examiner les sommets du masque pour connaître la localisation du fragment dans le pixel et en déduire le bon voisin.

Remarquons tout de suite que toutes ces méthodes ne prennent en compte au maximum que deux couleurs par pixel : celle de l'objet le plus proche et celle d'un voisin, correspondant à la couleur de fond ou à la couleur d'un polygone adjacent. Cette restriction se fait sentir dans l'affichage de scènes complexes où les objets proches ne sont pas sur des fonds unis. Par contre, à l'opposé des algorithmes de composition d'images que nous avons décrits précédemment, les algorithmes à voisins ne sont pas dépendants de l'ordre d'arrivée des polygones.

8.3 Le GZ-Buffer

Le *gz-buffer* présenté dans [Ghazanfarpour85], [Beigbeder et al.86] et dans [Ghazanfarpour et al.87] utilise un tampon géométrique associé au tampon de profondeur. Ce g-buffer de petite taille (4 bits pour coder 16 valeurs), stocke les informations géométriques, c'est-à-dire la composante α, de l'objet dans la mémoire d'images. Les facettes sont traitées en plusieurs passes. Les informations géométriques stockées dans le g-buffer servent à marquer les pixels de bords durant la première passe, puis à les antialiasser dans les suivantes. Cette méthode souffre essentiellement du fait qu'elle antialiasse les bords de polygones au moment de leur affichage, ce qui rend le résultat dépendant de l'ordre d'arrivée des primitives d'une part, et qui d'autre part fait intervenir la couleur de certains polygones invisibles.

Cette méthode a été améliorée dans [Ghazanfarpour et al.89], en particulier les trois passes ne s'effectuent plus successivement sur chaque polygone, mais chacune traite l'ensemble des facettes. Ceci permet de différer l'antialiassage et ainsi de ne pas prendre en compte les

polygones éliminés par un Z-Buffer dès la première passe. Les calculs couteux d'antialiassage (et de mise en perspective de textures) sont donc faits seulement pour les primitives visibles. Les trois passes se déroulent ainsi :

- la première affiche les pixels *intérieurs* (*i.e.* entièrement couverts) des polygones à l'aide d'un z-buffer,
- la seconde passe sert à traiter les arêtes implicites. Lorsqu'un pixel intérieur à un polygone *touche* un pixel intérieur d'un autre polygone, une arête implicite est soupçonnée, et un sur-échantillonnage local permet la traiter,
- la dernière passe permet d'antialiasser les pixels de bord des polygones. Les arêtes (qui n'ont pas encore été affichées) sont tracées par un algorithme de tracé de segments antialiassés (dans la première version l'algorithme utilisé était celui de [Pitteway et al.80] et dans la seconde version les auteurs préconisent plutôt [Gupta et al.81]). Durant cette étape, le moyennage des intensités s'effectue en prenant en compte deux couleurs :
  - la couleur du polygone courant (celui dont on trace le contour) associée à sa composante α (calculée par l'algorithme de tracé de segment),
  - la couleur d'un des huits voisins, obtenue en fonction de l'orientation de l'arête. Cette information est codée dans dans la mémoire appelée g-buffer.

La deuxième version du GZ-Buffer est beaucoup plus robuste que la première, en particulier grâce l'indépendance vis-à-vis de l'ordre d'arrivée des polygones. Elle donne des résultats intéressants d'autant qu'elle permet d'utiliser un éventuel Z-Buffer câblé, si la machine en est dotée. Notons que si l'augmentation de la mémoire nécessaire est très modeste (4 bits), les traitements sont conséquents : - trois passes, - sur-échantillonnage local pour les arêtes implicites. Par ailleurs, la méthode pose quelques problèmes aux sommets de polygones pour le choix du voisin, ainsi que dans le cas de polygones fins (inférieurs à la taille d'un pixel) ou, ce qui revient au même, dans le cas de polygones très proches mais non jointifs. Cela peut donner des résultats erronnés.

8.4 [Gros91]

[Gros91] décrit également une méthode d'antialiassage de bords de polygones utilisant des coefficients d'occupation et des voisins pour le calcul de la couleur finale. Comme la méthode précédente, elle utilise les voisins par arête uniquement (*i.e.* les voisins par arête sont les pixels qui partagent un coté avec le pixel courant, en opposition avec les voisins par sommet), mais elle ne gère pas les arêtes implicites. Le traitement d'antialiassage est effectué en deux passes : lors de la première les objets sont affichés à l'aide du Z-Buffer, les informations relatives au taux d'occupation ainsi qu'à l'orientation (pour le choix du voisin) sont calculées pendant cette passe. La seconde passe est effectuée directement sur la mémoire d'image, en chaque pixel les couleurs sont moyennées en utilisant les coefficients d'occupation et la couleur du pixel voisin. L'algorithme tient compte de plusieurs voisins dans le cas où le pixel est traversé par plusieurs arêtes de la primitive (par exemple dans le cas des triangles fins), cependant aucune information supplémentaire ne permet de savoir quelle est la part respective de chacun des voisins. Ils interviennent l'un et l'autre en proportion égale quelle que soit la surface qu'ils représentent, ce qui introduit une certaine erreur.

8.5 le PZ-Buffer

Nous avons proposé des améliorations à ce type d'algorithmes dans [Preux et al.92]. En premier lieu, nous avons montré qu'il était plus judicieux d'utiliser les voisins par sommet que les voisins par arête, ceci afin de pouvoir antialiasser plusieurs pixels par ligne et/ou par colonne. Par ailleurs, nous avons proposé de ne ne pas limiter notre information d'orientation à une seule direction, ceci afin d'éviter dans certains cas malheureux de choisir comme voisin un pixel qui serait lui-même un pixel de bord.

8.5.1 Voisins aux sommets *vs* voisins par arête

Le schéma sur la *figure 8.2* montre que tous les pixels traversés par un bord d'objet ne peuvent pas être traités correctement en prenant un voisin par arête. En effet, si l'on prend comme principe que les pixels ne conservent que le fragment de l'objet le plus proche le traversant, dans le cas décrit à gauche de la figure, le pixel (i,j-1) contient la même couleur que le pixel (i,j). Ce dernier se retrouve donc après application de l'algorithme d'antialiassage, de la couleur du polygone : il n'est pas antialiassé. En fait, sur cet exemple, seul le voisin NordOuest contient la couleur de fond. Notons que le phénomène ne peut jamais se produire lorsque les voisins sont choisis aux sommets à condition que le support de filtre utilisé pour le calcul de couverture ne dépasse pas la taille du pixel.

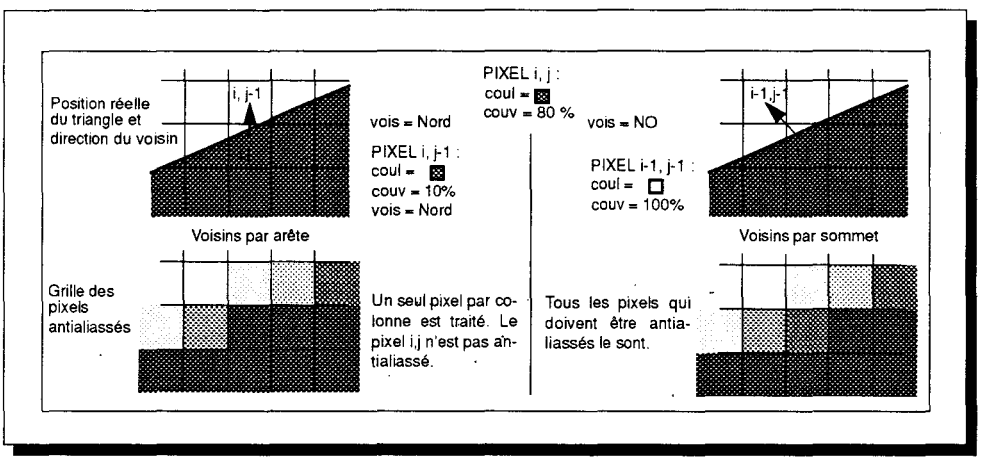


FIG. 8.2 - *Voisin par sommet ou voisin par arête.*

Nous pouvons remarquer que ce problème de plusieurs pixels à traiter par ligne ou par colonne a déjà été discuté dans le chapitre consacré aux algorithmes de tracé de segments antialiassé (*cf. 6.2.2*).

8.5.2 Deux voisins au lieu d'un seul

La figure 8.3 montre un exemple où le pixel voisin *"naturel"* est lui même traversé par une autre arête. En fait, bien souvent au moins deux pixels parmi les huits voisins contiennent la couleur recherchée (couleur du fond ou du polygone adjacent), nous avons expliqué que dans le cas général il était préférable d'utiliser les voisins par sommet, néanmoins lorsque ce voisin est lui même un pixel de bord ($\alpha < 1$), il a de bonnes chances de contenir une *mauvaise*

couleur. Il devient donc judicieux de se rabattre sur le voisin par arête. En ajoutant à la direction du voisin une information binaire donnant un sens de rotation, il est très simple de *changer de voisin* au cours de l'étape d'antialiassage si le premier est susceptible d'engendrer des erreurs.

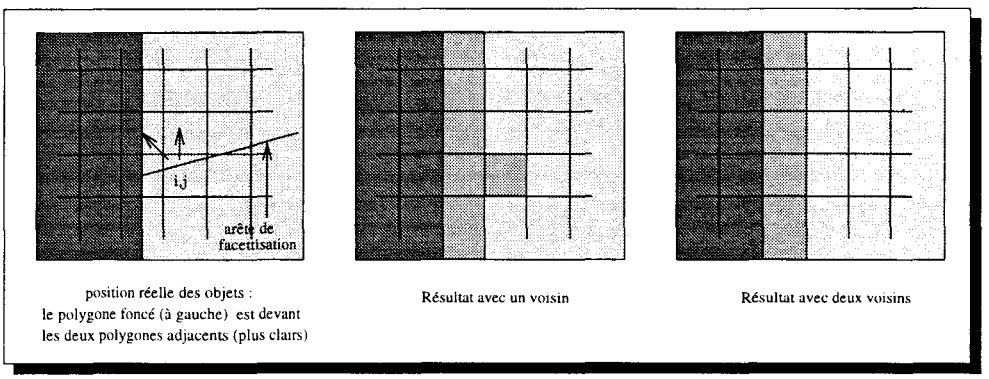


FIG. 8.3 - *Le choix du pixel voisin*

8.5.3 L'algorithme

Nous avons voulu pousuivre nos travaux dans la recherche des *deux meilleures couleurs* et après avoir trouvé quelques améliorations dans le choix du voisin pour la couleur complémentaire (*cf.* ci-dessus) nous nous sommes demandé si le Z-Buffer était le bon algorithme pour sélectionner un objet dans un pixel, ce qui revient à poser la question :
"Faut-il toujours garder l'objet le plus devant quel que soit son taux de couverture ? "
En effet, dans certains cas de figure, si deux objets couvrent partiellement un pixel sur un fond uni, et si le plus proche a un taux de couverture de 5% contre 90% à celui qui se trouve un peu plus loin, la tentation de se dire qu'il faut garder l'objet le plus profond comme couleur de base et éliminer le plus proche est grande. C'est en effet celui qui intervient le plus pour la couleur finale. Dans cet exemple les écarts sont suffisament grands et significatifs pour préférer garder le deuxième, mais quand les écarts diminuent le choix devient beaucoup plus délicat et il faut bien sûr définir une limite. Nous avons donc envisagé plusieurs techniques plus ou moins complexes et tenant compte des taux de couverture des différentes primitives présentes.

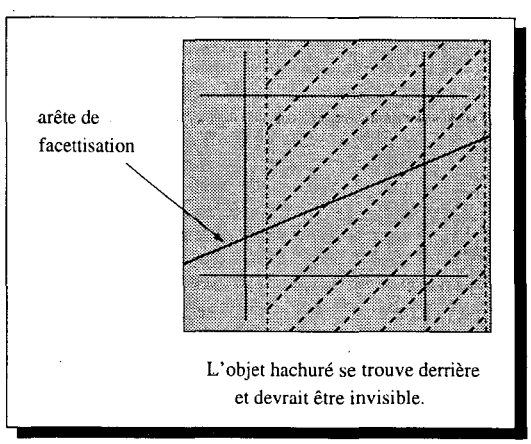


FIG. 8.4 - *Le taux de couverture ne peut pas être le principal critère de sélection*

Le taux de couverture comme unique critère de sélection n'est pas suffisant, en effet comme le montre la *figure 8.4*, cela peut générer des erreurs sur les arêtes de facettisations. Pour résoudre ce problème, nous faisons l'hypothèse que sur ce type d'arêtes, au moins l'un des polygones concernés a un taux de couverture de 50% (ce qui peut être faux dans le cas des sommets de polygones, nous assumons cette marge d'erreur) et nous ne gardons un polygone de couverture partielle mais plus profond que celui déjà stocké si et seulement si sa couverture est au moins du double. Ce compromis entre la profondeur de la primitive en un pixel (la valeur **Z**) et son **P**ourcentage de couverture est à l'origine du nom de la méthode *PZ-Buffer*, dont l'algorithme est le suivant :

- tous les pixels sont initialisés avec une couleur de fond, une profondeur et un coefficient α égal à 1.
- dans chaque pixel,
  - si l'objet courant est plus proche que celui déjà stocké et qu'il couvre entièrement le pixel ($\alpha = 1$), il le remplace (intérieur d'un objet).
  - si l'objet courant est plus proche que celui déjà stocké, qu'il couvre partiellement et que celui déjà stocké a un α de 1, il le remplace (bord d'objet sur un fond uniforme).
  - si l'objet courant est plus proche que celui déjà stocké, qu'il couvre partiellement et que celui déjà stocké a un α inférieur à 1 (conflit), il le remplace si sa couverture est au moins la moitié de celui déjà stocké.
  - si l'objet courant est plus éloigné que celui déjà stocké, qu'il couvre partiellement et que celui déjà stocké a un α inférieur à 1 (conflit, même cas que ci-dessus mais dans un ordre différent), il le remplace si sa couverture est au moins du double.

Pour mettre en oeuvre cette solution, il est nécessaire de stocker une seconde valeur de profondeur correspondant à l'objet le plus proche couvrant entièrement le pixel. En effet, cela permet d'éliminer tous les objets couvrant partiellement le pixel mais se trouvant au-delà de cette profondeur, donc invisibles (*cf. figure 8.5*).

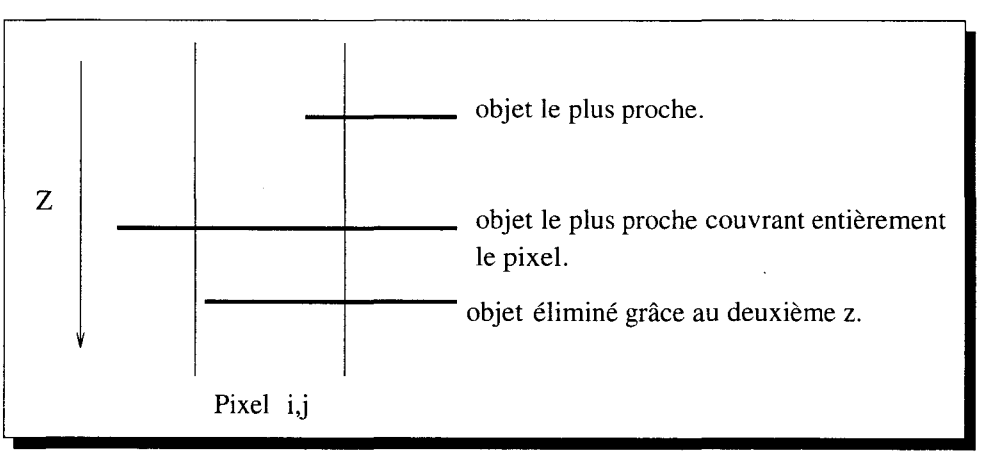


FIG. 8.5 - *Utilisation d'un deuxième z*

8.5.4 Conclusion sur le PZ-Buffer et les méthodes à voisin

Nos travaux sur le PZ-Buffer nous ont beaucoup appris. Après avoir apporté à moindre frais quelques améliorations notables sur le choix du pixel voisin pour obtenir la deuxième couleur nécessaire à l'antialiassage, nous nous sommes attaqués à un défaut qui nous paraissait important dans les méthodes à voisin, à savoir le fait que deux couleurs seulement étaient utilisées. Nous avons donc considérablement augmenté la complexité de l'algorithme pour choisir avec plus de précision la *première* couleur. Or les améliorations apportées par la méthode, même si elles sont réelles, ne sont d'une part, pas à la mesure de la complexification des traitements, mais surtout nous ont paru presque négligeables en regard des autres défauts intrinsèques aux méthodes à voisin, à savoir les problèmes aux sommets des polygones et sur les objets fins ou faiblement espacés. En conséquence, nous pensons que ce type de méthode peut être très efficace pour l'antialiassage de scènes simples (grandes facettes, peu de recouvrement), mais nous paraît une solution insuffisante dans le cas de scènes plus complexes (*i.e.* contenant beaucoup d'objets finement facettisés).

8.6 Les méthodes à voisins pour l'antialiassage de l'hémicube en radiosité

Il existe pour certaines implémentations de la méthode de radiosité (calcul des facteurs de formes par projection des facettes sur une surface discrétisée) des défauts d'éclairement qui sont, par similitude avec d'autres phénomènes, appelés *aliassage*. Nous avons eu l'idée, après avoir observé cette similitude, d'appliquer notre algorithme à voisin pour résoudre ce problème.

Il n'est pas possible, dans le cadre de ce mémoire, de rappeler les principes de la méthode d'éclairement global dite de radiosité. Nous considèrerons donc que le lecteur est familier avec cette technique. Pour plus de détails nous renvoyons par exemple, à [Renaud93] puisque ce travail a été fait en collaboration avec l'auteur.

8.6.1 Le problème

Si nous prenons l'hémicube comme exemple de surface de projection, la méthode qui calcule les facteurs de forme peut se résumer de la façon suivante :

- Les cinq plans de l'hémicube sont discrétisés en surfaces élémentaires appelées *proxels* (pour *projective element*), chacune d'elles ayant un facteur de forme pré-calculé.
- Chaque primitive de la scène est projetée sur l'hémicube, en utilisant un algorithme d'élimination des parties cachées de type Z-Buffer.
- Les proxels dont le centre est recouvert par la projection d'une facette en stockent le numéro (comme le ferait une mémoire de trame dans le cas du rendu).
- Quand tous les polygones ont été projetés, le facteur de forme de chacun d'eux est obtenu en sommant des facteurs de forme élémentaires des proxels qu'il recouvre.

Cette technique présente beaucoup de ressemblances avec l'algorithme de rendu. Il est d'ailleurs courant d'utiliser le même algorithme pour la projection et l'élimination des parties

cachées dans le cas de :

- la projection des facettes sur l'écran pour le rendu,
- la projection des facettes sur l'hémicube pour le cacul des facteurs de forme.

Dans les deux cas, il s'agit d'échantillonner les projections sur une surface discrétisée. Il est donc tout à fait naturel de retrouver dans le cas du calcul des facteurs de forme, des phénomènes de *crénelage* (abusivement appelés *aliassage*), identiques à ce que nous observons sur les bords de polygones avec un rendu par échantillonnage ponctuel.

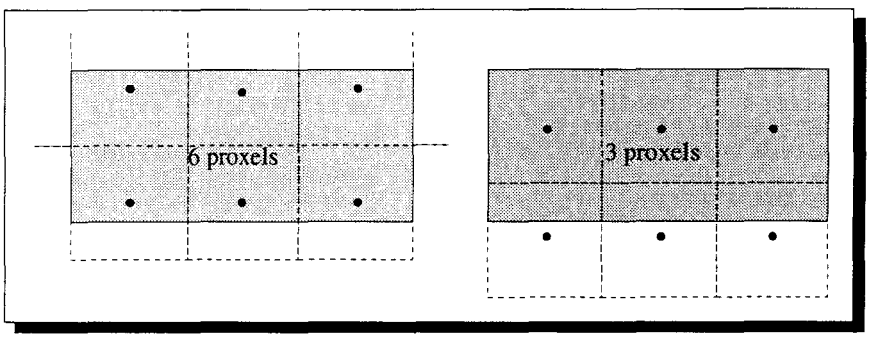


FIG. 8.6 - *Aliassage dû à l'échantillonnage ponctuel et régulier de l'hémicube*

8.6.2 Les différences entre le rendu et l'hémicube

Malgré les similitudes certaines entre ces deux algorithmes, la manifestation de l'aliassage est différente. Dans les deux cas les erreurs sont introduites au moment de la projection à cause de l'échantillonnage ponctuel : on considère que ce qui est visible au centre du pixel (respectivement du proxel) est présent sur l'ensemble de la surface. Cependant, si pour le rendu l'erreur commise en un pixel est directement visible (c'est la couleur stockée qui est affichée), ce n'est pas le cas pour l'hémicube. En effet, les proxels ne sont qu'une étape intermédiaire : on n'affiche pas les valeurs des proxels, mais chacun d'eux participe au calcul d'énergie d'une facette. On peut tout de suite conclure que la précision de l'antialiassage aura beaucoup moins d'importance dans ce cas que dans le cas du rendu. La *figure 8.6* montre comment se caractérise l'aliassage lors de la projection des facettes : le nombre de proxels recouverts n'est pas toujours proportionnel à la surface de la facette. Les polygones qui couvrent moins de la moitié de plusieurs proxels obtiennent moins d'énergie que ce qu'ils devraient, et ceux qui couvrent un peu plus de la moitié en obtiennent trop. La juxtaposition de ces facettes sur de grandes surfaces fait apparaître des sauts d'energie sous forme de bandes ou de carreaux (délimitant les primitives d'affichage) comme le montre la *figure 8.7*. Cette erreur ne se commet que sur les proxels de bord des polygones, la précision du facteur de forme d'une facette dépend donc du rapport entre le nombre de proxels de bord et le nombre total de proxels recouverts.

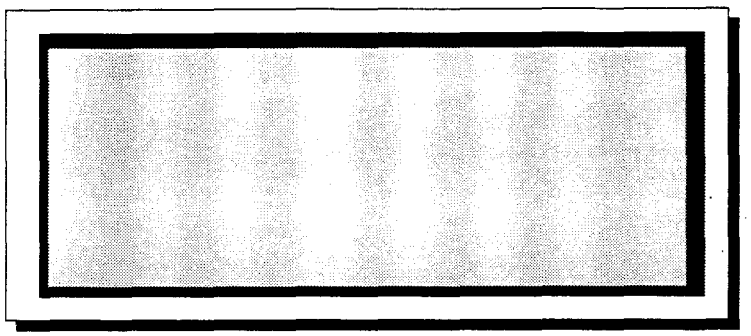


FIG. 8.7 - *Résultats visuels de l'aliassage de l'hémicube*

8.6.3 Les précédentes solutions

De la même façon qu'on diminue l'aliassage en rendu par le sur-échantillonnage, l'augmentation du nombre de proxels atténue les sauts d'intensité en approximant de manière plus précise la surface projetée de chaque facette. Néanmoins, le coût de calcul d'un hémicube, ou de toute autre surface de projection, dépend directement de la résolution utilisée.

[Meyer90] propose d'utiliser les techniques d'échantillonnage stochastique pour diminuer l'aliassage sans augmenter la résolution de l'hémicube. L'auteur se base sur le fait que l'erreur commise sur les facteurs de forme est maximale lorsque les facettes de la scène sont alignées avec la grille de proxels. Dans ce cas, pour une facette donnée, la même erreur est répétée sur tous les proxels de bord d'une arête (sans compensation) ce qui donne une erreur globale importante.

La solution proposée consiste alors, pour chaque nouvelle facette émettrice, à changer l'orientation de l'hémicube en effectuant une rotation aléatoire (par une méthode de *jittering*) autour de son axe Z, ce qui permet de répartir l'erreur commise sur des facettes jointives.

8.6.4 Adaptation de notre méthode à voisin

Nous avons proposé d'adapter l'algorithme d'échantillonnage surfacique que nous avons développé à l'échantillonnage des facettes sur la grille de proxels. Comme pour le rendu, les facettes sont échantillonnées en calculant un taux de couverture sur chaque proxel (une composante α) ainsi qu'une direction pour déterminer le *proxel voisin* (*i.e.* le proxel qui a le plus de chance de contenir le numéro de la facette jointive ou de l'objet en arrière plan). Lors du calcul de la somme des facteurs de forme pour chaque polygone, si un proxel n'est que partiellement couvert, sa contribution pour la facette qu'il représente est pondérée par la composante α, ce qui évite la sur-évaluation de l'énergie totale. De plus, l'information sur le voisin sert à *rendre* à la facette sous-évaluée l'énergie restante $((1-\alpha) \times FFE)$ [1]. En traitant les deux aspects du problème, notre solution réduit les écarts et évite ainsi les sauts d'intensité entre les facettes.

1. FFE = Facteur de Forme Élémentaire

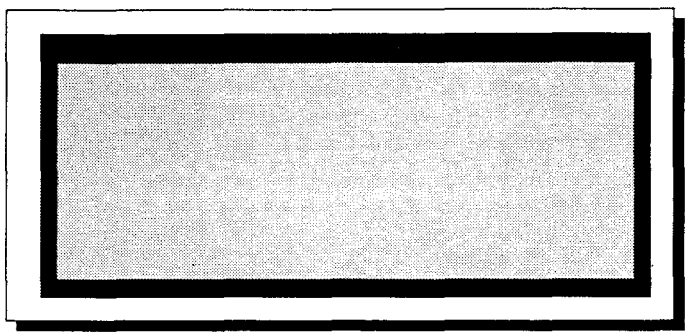


FIG. 8.8 - *La même surface avec antialiassage de l'hémicube par notre méthode à voisin*

8.6.5 Efficacité sur l'hémicube

La technique des voisins souffre, dans le cadre de l'antialiassage de l'hémicube, des mêmes défauts et limitations que pour le rendu (*cf.* plus haut). Cependant, les erreurs introduites sont beaucoup moins sensibles dans le cas de l'hémicube.

Prenons l'exemple des sommets de polygones : dans le cas du rendu de l'image, si un pixel contient un sommet de facette, cela signifie que deux arêtes au moins traversent le pixel : il y a alors ambiguïté sur le choix du voisin. La couleur qui sera choisie ne sera peut être pas représentative et le pixel risque de présenter une discontinuité dans l'image. Pour l'hémicube, le problème se décompose en deux étapes :

- la facette stockée n'est pas sur-évaluée grâce à la pondération du coefficient $\cos\theta$.
- le reste de l'énergie du proxel est redistribué à l'une des facettes jointives qui était sous-évaluée.

La deuxième étape introduit une erreur : en effet, la part restante d'énergie devrait être redistribuée sur plusieurs facettes, alors qu'une seule en profite. Cependant, d'une part l'écart entre deux facettes a déjà été réduit grâce à la première étape, d'autre part l'erreur engendrée porte sur la contribution d'un seul proxel, ce qui une fois ramené à l'énergie totale de la facette, n'a que très peu d'influence.

8.6.6 Conclusion

En conclusion, dans tous les cas notre méthode améliore le résultat en évitant la sur-évaluation de certaines facettes. Dans la grande majorité des cas toutes les facettes sont traitées de manière exacte et dans quelques cas rares l'erreur commise est très légère. Une implémentation de cette technique a été réalisée dans le cadre d'un mémoire de DEA [Patel95] que j'ai co-encadré avec Christophe Renaud (la *figure 8.8* montre des résultats visuels de cette implémentation sur une simple surface discrétisée en facettes).

Chapitre 9

Les méthodes à masque

9.1 Le principe du masque

Nous venons de voir que les méthodes à voisins étaient une première étape vers la prise en compte de la localisation de l'information de couverture de l'objet sur le pixel. Néanmoins cette information est peu précise et ne permet pas de tenir compte correctement de certains recouvrements. [Weinberg81] a proposé de convertir la valeur de la composante α et la pente de la droite qui coupe le pixel en un masque de bits représentant le pixel. Chaque bit correspond à un sous-pixel, si le sous-pixel est couvert par la primitive sa valeur est positionnée à 1, elle vaut 0 sinon (*cf* figure 9.1).

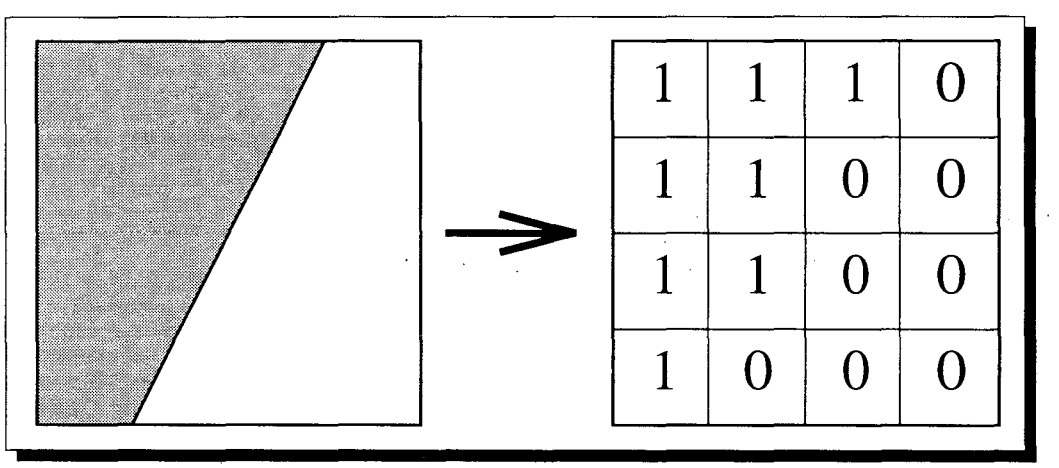


FIG. 9.1 - *Le masque de sous-pixels (ou masque d'occupation)*

Cette technique a déjà été abordée plus haut à propos du calcul de la valeur d'α aux sommets des polygones. Il s'agissait alors de combiner les masques des différentes droites délimitant la primitive pour en tirer un masque global pour le polygone et à partir duquel on pouvait, en comptant les bits à 1, retrouver la valeur d'α. Les méthodes que nous décrivons maintenant fonctionnent sur le même principe, mais stockent le masque de bits en lieu et place de la composante α. Cette représentation de la géométrie du pixel va permettre d'effectuer l'élimination des parties cachées au niveau des sous-pixels un peu comme le ferait un algorithme de sur-échantillonnage. La comparaison s'arête là, car les autres grandeurs du

pixel (couleur, profondeur) ne sont calculées qu'une seule fois pour l'ensemble du pixel, ce qui pose d'ailleurs nous allons le voir, certains problèmes.

9.2 Quel type de masque?

Une des premières questions qui se posent pour ce type d'approche, c'est la taille et la forme du masque. De ces caractéristiques vont dépendre la précision du résultat. Comme nous le verrons dans la partie 4 (ref mise en oeuvre taille et forme du filtre), la forme du masque peut être différente de la forme du support de filtre servant à calculer la composante α. Par contre le nombre de sous-pixels est déterminant pour la qualité des résultats. On peut implémenter un masque avec un nombre quelconque de bits, le principe de fonctionnement de l'algorithme reste le même. Dans ce chapitre, nous batissons nos exemples avec des masques carrés 4×4, cette solution est d'ailleurs la plus usitée.

Le calcul du masque de bits correspondant à la couverture d'un objet dans un pixel dépend bien entendu de la résolution choisie et de l'algorithme de tracé de polygone utilisé (suivi de contour, ou test d'inclusion). Cependant, dans tous les cas cela revient à utiliser une table d'indirection prenant en entrée la pente de la droite et la distance du centre du pixel à cette droite et donnant à la sortie le masque de sous-pixel correspondant.

9.3 Elimination des parties cachées, l'exemple de référence: Le A-Buffer

Les masques permettent de tenir compte des recouvrements en testant la présence des objets en chaque sous-pixel à la manière du sur-échantillonnage. Cependant, contrairement à ce dernier, les méthodes à masques ne stockent pas la couleur et la profondeur en chaque sous-pixel mais seulement une fois par pixel, ce qui empêche de calculer *au vol* la couleur des sous-pixels. Il est donc nécessaire de procéder en deux étapes:

- Toutes les facettes sont échantillonnées, en chaque pixel les masques d'occupation de tous les polygones présents sont stockés.
- Pour chaque pixel, la liste de tous les objets présents est parcourue en respectant l'ordre de profondeur, ce qui permet d'effectuer l'élimination des parties cachées au niveau des sous-pixels.

Le A-Buffer est un algorithme très populaire et très utilisé malgré sa complexité. Il a initialement été développé dans le cadre du projet *REYES 3-D*, un système de rendu 3D qui a été utilisé pour créer des effets spéciaux dans de grandes productions cinématographiques de la société *Lucasfilm*. Nous détaillons maintenant l'algorithme du A-Buffer [Carpenter84] qui, bien qu'il ne soit pas chronologiquement le premier à avoir utilisé ce type de méthode, sert de référence et est même devenu un terme générique pour décrire les algorithmes d'échantillonnage surfacique à liste de priorité utilisant des masques de sous-pixels.

Le A-Buffer définit la notion de *fragment* d'un objet, qui correspond en fait à la représentation sous forme d'un masque de sous-pixels, de la frontière d'un objet à l'intérieur d'un pixel (ce terme est lui aussi devenu générique et s'applique dans ce type d'algorithme à la projection d'une primitive sur un masque de bits correspondant à un pixel). Pour cela deux

structures de données particulières *pixelstruct* et *fragment* sont utilisées. Nous les décrivons ci-dessous en code C.

```
typedef struct {
   float z; /* positif quand l'objet couvre entièrement le pixel,
               négatif sinon */
   union {
      Fragment *fliste; /* pointeur vers une liste de fragments;
                           jamais nul */
      struct { /* les données du pixel si l'objet couvre entièrement */
         unsigned char r, v, b; /* la couleur */
         unsigned char α; /* la couverture */
      }pixel;
   }fragmentOUpixel;
}Pixelstruct;

typedef struct Frag { /* élément de la liste d'objets couvrant
                         patiellement le pixel. */
   struct Frag *suivant;
   short r, v, b;
   short opacité; /* 1-transparence (0..1) */
   short surface;
   short numObjet; /* numéro servant à repérer les facettes jointives
                      issues de la même primitive */
   pixelmasque m; /* le masque de sous-pixels: 4x8 */
   float zmin, zmax;
}Fragment;
```

ALGO. 1 - *Les structures de données utilisées dans le A-Buffer*

Lorsque dans un pixel, l'objet le plus proche couvre entièrement la surface, l'algorithme se comporte comme un z-buffer. Par contre, dès que l'objet traité ne couvre que partiellement le pixel et qu'il se trouve devant ce qui est déjà stocké, la valeur de profondeur devient négative (pour servir de drapeau) et des fragments sont créés pour constituer une liste des différents objets visibles. Ensuite, les autres fragments éventuels viennent s'insérer dans cette liste triée en profondeur.

Quand tous les polygones ont été convertis en fragments, une seconde passe appelée *packing* calcule en chaque point de l'image où le z est négatif (c'est-à-dire là où plusieurs objets sont visibles) la couleur finale du pixel. Pour cela la liste de fragments est parcourue en respectant l'ordre de profondeur (le plus proche d'abord), chaque fragment participant à la couleur finale du pixel en fonction du nombre de sous-pixels qu'il représente.

Le packing

Au départ, un masque de bits de sous-pixels est initialisé en mettant toutes les valeurs à 1, c'est le masque de recherche, noté M_{search} (*search mask*), qui servira tout au long du

processus. À chaque fragment rencontré, le masque d'occupation correspondant, que l'on appelle M_{frag} est combiné (ET logique) avec le masque de recherche pour donner un masque intermédiaire M_{in} :

$$M_{in} = M_{frag} \& M_{search}$$

M_{in} représente donc la contribution réelle de l'objet courant dans le pixel, c'est le nombre de bits à 1 dans M_{in} qui permet de pondérer la couleur du fragment dans le calcul de la couleur finale. Cependant, la surface encore libre, notée M_{out}, est donnée par :

$$M_{out} = M_{search} \& \overline{M_{frag}}$$

Après le traitement de chaque fragment, M_{out} devient M_{search} et la fonction continue récursivement.

En notant A_{in} et C_{in} la couverture réelle (obtenue à partir de $M - in$ et la couleur du fragment courant, on calcule la couleur finale du pixel de manière récursive avec la formule suivante :

$$C = C_{in} \dot{A}_{in} + C_{out}(1 - A_{in})$$

C_{out} étant bien entendu la couleur résultante du masque M_{out}, calculée récursivement. L'algorithme s'arête quand tous les fragments ont été traités où quand M_{search} ne contient plus aucun bit à 1.

Les arêtes implicites avec le A-Buffer

Chaque fragment contient deux informations de profondeur $zmin$ et $zmax$, l'auteur propose de les utiliser afin de détecter les intersections entre polygones d'une part, et d'autre part de calculer leur couverture respective. Pour cela il propose de vérifier si les intervalles définis par les deux valeurs de profondeur se recouvrent ou pas. S'ils se recouvrent, on suppose qu'il y a intersection et la visibilté de chacun des fragments (notés A et B) est déterminée par :

$$Vis_A = \frac{Zmax_B - Zmin_A}{(Zmax - Zmin)_A + (Zmax - Zmin)_B}$$

Cette formule permet en effet de calculer la visibilité dans le cas idéal (*cf. figure 9.2 (a)*)où les deux polygones se coupent véritablement. Or, [Schilling et al.93] donne un exemple où l'algorithme donne un résultat complètement faux (*cf. figure 9.2 (b)*). On peut donc dire que le A-Buffer tel que l'a décrit Loren Carpenter, même s'il propose une ébauche de solution, ne traite pas correctement les arêtes implicites générées par l'intersection de deux polygones.

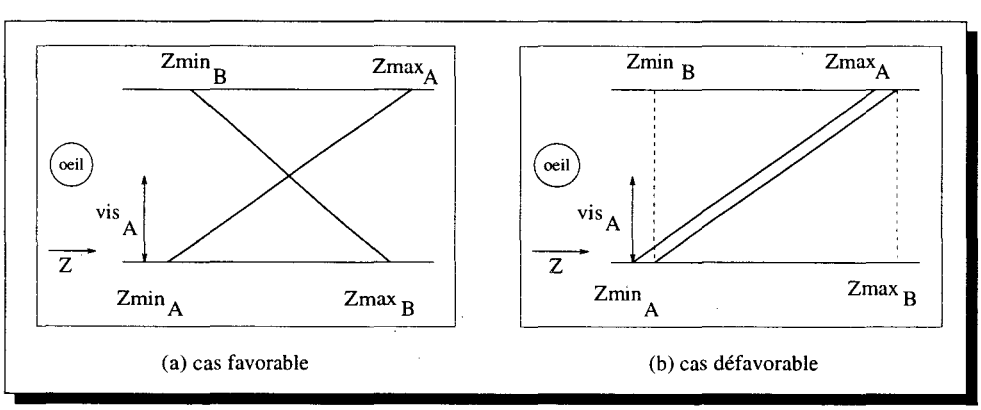


FIG. 9.2 - *Erreur commise par le A-Buffer*

Remarques : La technique décrite dans [Carpenter84] est en fait une version simplifiée de l'opérateur **Comp**, présenté dans [Duff85], qui ne prend en compte que deux valeurs (Zmin et Zmax) au lieu de quatre (une à chaque coin du pixel) dans l'autre version.

Les arêtes de facettisation avec le A-Buffer : le *merging*

Lorsqu'un fragment est inséré dans la liste triée en profondeur, son numéro d'objet (correspondant à des primitives géométriques continues ne s'auto-recoupant pas) est comparé avec les fragments directement *au-dessus* et *en-dessous*. Si le numéro est le même et qu'il y a un recouvrement des intervalles de profondeur, alors l'algorithme considère que les deux fragments sont issus du même objet de la scène et leur masque de bits sont combinés par un OU logique (*merging*). Cette technique permet d'alléger la liste des fragments, ce qui diminue la taille de la mémoire nécessaire et augmente l'efficacité de l'étape de *packing*. Elle possède quelques ressemblances avec ce que nous appelons le *cumul* des valeurs de couverture dans notre algorithme (*cf. 12.1.1*).

Les transparences avec le A-Buffer

Lors du calcul de la couleur finale d'un pixel, pour prendre en compte la transparence d'un objet, il suffit de multiplier sa couverture réelle (A_{in}) par un coefficient de transparence. C'est ce que fait le A-Buffer en utilisant le champ *opacité* de la structure *Fragment*. En fait le calcul de couleur est effectué en pondérant la couleur de l'objet par un coefficient α qui est le produit de l'occupation réelle du fragment et du coefficient d'opacité :

$$\alpha = A_{in} \times opacite$$

Il reste bien sûr à prendre en compte pour un coefficient de $(1 - opacite)$ les fragments se trouvant sous le fragment d'objet transparent, ce qui est fait en rappelant la procédure récursive avec M_{in} comme masque de recherche (M_{search}). La couleur obtenue par cette étape est appelée C_{behind}. La couleur C_{in} finalement prise en compte pour le fragment transparent courant M_{in} est donc obtenue par :

$$C_{in} = opacite_{frag} \times C_{frag} + (1 - opacite_{frag}) \times C_{behind}$$

La taille du masque

Notons que l'article précise qu'après plusieurs expérimentations, un masque 4×8 bits a été sélectionné pour représenter le pixel. Nous pensons que la facilité d'implémentations d'un tel masque est la principale raison de ce choix dissymétrique. L'ensemble des auteurs actuels s'accordent sur l'efficacité d'un masque carré, le masque 4×4 étant de loin le plus utilisé pour la plupart des applications.
Par ailleurs, l'année précédente [Fiume et al.83] publiaient une autre méthode d'antialiassage à base de masque de sous-pixels, leur choix s'étant arêté sur un masque carré 8×8. Bien que plus approximatif, l'algorithme est très proche du A-Buffer, simplement lorsque plusieurs fragments sont présents en un pixel, seul le plus proche est traité de manière exacte, les suivants intervenants tous dans la couleurs finale du pixel sans tenir compte des éventuels

recouvrements. Cette détermination à traiter de manière exacte l'objet le plus proche est également une idée que nous retrouvons dans nos algorithmes (*cf.* §*12.1.1*). Précisons que l'article ne décrit aucune solution pour les arêtes implicites.

Les autres méthodes

Rappelons que [Weinberg81] est l'ancêtre du A-Buffer bien qu'il ait en fait été décrit pour une machine parallèle. Ramenée à une algorithmique séquentielle, cette méthode, bien qu'un peu plus rustique, utilise la même technique en deux passes, la première pour éliminer les pixels *pleins* et créer des listes d'objets avec leur masque pour les pixels *coupés*. La deuxième passe parcourt les listes d'objets en respectant l'ordre de profondeur et calcule la couleur finale de manière similaire au A-Buffer. Cependant, rien n'est proposé ni pour les objets transparents ni pour les arêtes implicites.

Beaucoup plus récemment, [Lau et al.95] ont proposé un algorithme dérivé du A-Buffer. Au lieu de stocker les masques de bits pour chaque fragment présent en un pixel, les auteurs conservent un pointeur vers ces masques. Ils reprennent l'idée développée dans [Fiume et al.83] où tous les masques possibles étaient pré-calculés et stockés dans un table d'indirection. Les positions d'intersection entre une arête de polygone et les frontières du pixel servent d'index pour adresser cette table. Mais alors que Fiume et Fournier récupéraient le masque de bits et le stockaient pour chaque objet, les auteurs du *Compositing Buffer* préfèrent garder les index. Cette solution est d'une part plus économique en mémoire surtout pour des masques hautes résolutions, et d'autre part elle permet d'adapter cette résolution dynamiquement en cours de traitement. Cela permet de traiter plus finement certains pixels délicats et facilite également l'opération de grossissement (zoom). Au lieu de multiplier les valeurs des pixels, ils suffit de transformer les index en masques de bits de haute résolution pour obtenir une image zoomée antialiassée.

Les auteurs proposent également d'utiliser cette technique pour remplacer les représentations *fil de fer* durant l'étape de modélisation (*image editing*) par un rendu de qualité interactif. Pour cela ils définissent la notion d'objet *dynamique* et d'objet *statique*. Les objets dynamiques sont les objets dont la définition n'est pas terminée et qui peuvent être modifiés. Une fois l'image rendue, si un objet dynamique est modifiée (par exemple retiré) les objets statiques sont disponibles pour être réaffichés sans nouveau calcul.

Le principal inconvénient de cette méthode est la complexité algorithmique ajoutée par les index de masques de bits. Les auteurs font remarquer que cette complexité ralentit considérablement le système dans le cas de scènes complexes constituées de nombreuses petites facettes. Il nous paraît alors paradoxal de proposer une méthode qui se veut d'une grande précision (la résolution des masques peut aller jusque 32×32) mais qui est surtout adaptée aux scènes simples (*i.e.* modélisées à l'aide de grandes facettes). Nous avons vu plus haut que dans ce cas des méthodes beaucoup plus simples étaient suffisantes.

9.4 Comparaison des méthodes à masques avec le sur-échantillonnage

L'utilisation de masques de sous-pixels permet donc d'effectuer l'élimination des parties cachées au niveau sous-pixel à la manière du sur-échantillonnage. On peut effectivement comparer les deux techniques en prenant par exemple comme base un sur-échantillonnage 4×4

régulier et un A-Buffer avec un masque de 16 bits. Au niveau du calcul de la couleur finale, le taux de couverture des différents fragments peut dans certains cas (*cf. figure 9.3* se trouver beaucoup plus précis grâce à l'échantillonnage surfacique, l'échantillonnage ponctuel, même à haute résolution, générant toujours des sauts d'intensité d'un pixel à l'autre. Comme le montre la figure ci-dessous, les défauts seront également sensibles en animation.

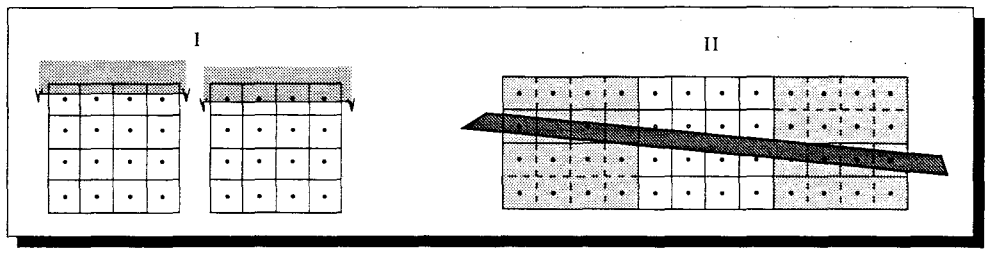


FIG. 9.3 - *Lechantillonnage surfacique plus précis que le sur-échantillonnage*

Cependant, avec un sur-échantillonnage, les valeurs de couleur et de profondeur sont recalculées en chaque sous-pixel, ce qui n'est pas le cas des méthodes à masques pour lesquelles une seule valeur est disponible pour l'ensemble du pixel : celle qui est calculée au centre du pixel. Toutes les méthodes d'échantillonnage surfacique souffrent de ce décalage qui existe entre la bonne précision géométrique de la projection du fragment d'objet dans le pixel et la valeur unique de profondeur et de couleur. Cet inconvénient est sans aucun doute la contrepartie de l'économie de temps de calcul et de place mémoire qui est nécessaire pour recalculer ces informations.

9.4.1 Le z unique et les arêtes implicites

Nous avons vu plus haut que Loren Carpenter avait déjà analysé le problème des arêtes implicites impossibles à detecter avec une seule valeur de profondeur. Il a proposé une ébauche de solution en gardant deux valeurs *zmin* et *zmax* par fragment. Cette solution approximative est tout à fait insuffisante, mais [Schilling et al.93] donne une solution à ce problème délicat, solution qui se traduit évidement par une augmentation de l'information à stocker. Les auteurs de l'article parlent de masques de priorité (P-Mask) qui vont permettrent, lorsque deux primitives se coupent à l'intérieur d'un pixel, de déterminer la visibilité de chacune d'elles. Pour cela, une seule valeur de profondeur (correspondant au centre du pixel) est calculée et stockée mais les pentes en x et en y (respectivement dz_x et dz_y) du plan support de la facette sont également conservées. Notons que ces informations ne nécessitent aucun calcul supplémentaire car elles sont de toute façon utilisées par tout système de *rasterisation* qui interpole les valeurs de couleur et de profondeur. À l'aide de ces informations, il est possible de reconstruire l'équation d'un plan délimitant les zones de visibilité de chacun des deux plans de départ. Les paramètres de cette équation de plan s'obtiennent par :

$$z = z_1 - z_2$$

$$dz_x = dz_{1,x} - dz_{2,x}$$

$$dz_y = dz_{1,y} - dz_{2,y}$$

On peut également voir ces coefficients comme ceux de l'équation de la droite supportant l'arête implicite à partir de laquelle il devient simple par les techniques classiques décrites plus haut de définir un masque de visibilité, appelé dans l'article masque de priorité.
Il reste ensuite à déterminer les cas où il y a réellement intersection de primitives. Les auteurs montrent que la relation :

$$z2 - z1 < (|dz_{2,x} - dz_{1,x}| + |dz_{2,y} - dz_{1,y}|)/2 \qquad (9.1)$$

est un critère fiable pour la détermination de la présence d'une intersection de polygones dans un pixel. Une implémentation matérielle est également proposée dans ce papier.

9.4.2 Les erreurs de visibilité

La solution décrite ci-dessus pour les arêtes implicites, bien que coûteuse, est très performante et permet également de traiter un autre problème légèrement différent mais ayant la même cause, à savoir l'unicité de la valeur de profondeur. Il s'agit du calcul erroné de certaines valeurs à l'extérieur du pixel. L'échantillonnage surfacique prenant en compte des fragments avec des couvertures partielles, celles-ci peuvent évidement être inférieure à 50% de la surface d'un pixel. Cela signifie que le calcul de la valeur de profondeur (ainsi que pour la couleur) s'effectue (par interpolation rappelons-le) en dehors de la primitive puisque le point d'évaluation d'un pixel est son centre. Bien pire que dans le cas précédent, la valeur calculée n'est pas seulement uniforme pour l'ensemble du fragment, elle est fausse et ne correspond à aucun des sous-pixels. La figure 9.4 montre que cela peut conduire à des erreurs grossières de visibilité.

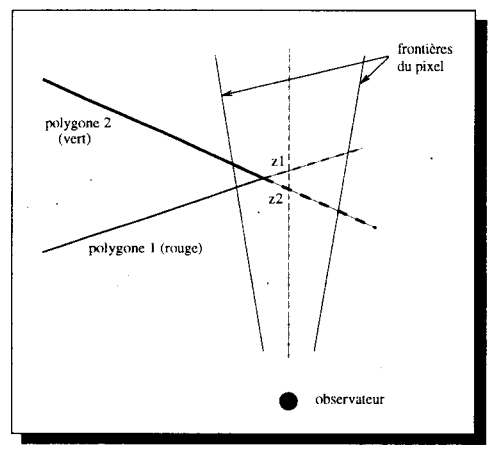


FIG. 9.4 - *Erreur de visibilité dûe au calcul de profondeur à l'extérieur des frontières du polygones.*

9.5 Débordement des couleurs

9.5.1 Le phénomène

Tous les algorithmes d'échantillonnage surfacique souffrent de cet inconvénient que l'on retrouve également dans le calcul (toujours par interpolation) de la couleur du pixel. Dans ce

cas, il arrive souvent que la valeur obtenue par interpolation sorte de l'intervalle permis (par exemple une intensité négative).

9.5.2 Différences entre interpolation de Gouraud et de Phong

Bien que le rendu avec éclairage de phong (interpolation des normales en chaque pixel) ne soit pas le cadre de notre travail (encore trop couteux pour permettre une implémentation matérielle en vue d'un affichage temps réel), nous expliquons maintenant l'intêret qu'il y aurait à utiliser cet algorithme à la place de l'éclairage de Gouraud. Rappelons les principales caractéristiques de ces deux techniques :

ombrage de Gouraud

- les couleurs sont calculées en chaque sommet de la primitive d'affichage (dans le repère écran donc après projection),
- pour chaque composante (r, v, b) un *plan de couleur* de la forme $c_i = A_i x + B_i y + C_i$ est calculé à partir des valeurs aux sommets,
- à partir d'une valeur initiale $c0_i$, on calcule incrémentalement les valeurs en chaque pixel du polygone ($+B_i$: pour obtenir la valeur de la ligne suivante, $+A_i$: la valeur du pixel suivant.

ombrage de Phong

- Le principe de l'interpolation de Phong est identique à celle de Gouraud, excepté qu'elle s'effectue sur les composantes de normales (N_x, N_y, N_z) à la place des valeurs de couleurs. Le calcul d'éclairement doit donc être refait en chaque pixel visible.
- Cette technique permet en particulier de détecter des cas de relexion spéculaire à l'intérieur de certaines facettes, alors qu'ils peuvent échapper à l'éclairement de Gouraud s'ils ne sont pas *capturés* par les sommets.

Interpolation en dehors de la primitive

Comme nous l'avons vu précédemment pour les valeurs de profondeur, l'échantillonnage surfacique nous oblige à effectuer des interpolations en dehors de la primitive. Lorsque les plans supports ont un très fort gradient, les valeurs obtenues par interpolation peuvent sortir de l'intervalle valide (par exemple 0..255 pour les couleurs). Ces cas de*débordement* sont très gênants car ils provoquent des décalages dans le codage des couleurs (les valeurs d'intensité peuvent par exemple devenir négatives).

Ce problème est très délicat, plusieurs solutions ont été proposées pour le résoudre [?], [Molnar91]. Toutes ces solutions sont complexes et couteuses, dans la pratique on a souvent recours à un simple test de débordement qui permet de borner l'intervalle.

Avec une interpolation de Phong (*i.e.* sur les normales), le problème est différent. Calculées en dehors de la primitive, les composantes du vecteur peuvent être fausses, la couleur calculée ne correspondra donc pas forcément à l'objet. Cependant, les vecteurs étant normalisés, aucune valeur ne peut poser de problème de calcul en sortant de l'intervalle autorisé. De plus, la normale intervient dans le calcul de la couleur surtout au niveau de la composante

spéculaire et un peu pour la composante diffuse, la valeur de l'ambiant reste donc sans erreur. Enfin, s'il y a échantillonnage en dehors de la primitive c'est que l'objet couvre moins de 50% du pixel, sa participation à la couleur finale ne sera donc pas prédominante. Cet argument vaut bien sûr également pour l'interpolation de Gouraud, mais dans ce cas, l'erreur peut être tellement importante qu'une contribution même inférieure à la moitié peut se révéler très visible.

Pour résumer, l'ombrage de Phong paraît beaucoup mieux adapté à l'échantillonnage surfacique que le l'ombrage de Gouraud.

Chapitre 10

De l'évaluation des méthodes...

Il aurait évidemment été intéressant de tirer des résultats numériques de cette étude afin de pouvoir conclure définitivement. Malheureusement comme cela sera le cas également par la suite, nous avons été confronté au problème de l'évaluation des méthodes d'antialiassage. Depuis [Fiume et al.83] il existe un *consensus* pour évaluer lesdites méthodes par les critères suivants :

- Quels sont les cas que la méthode traite correctement?
- Quels sont les cas que la méthode ne sait pas traiter et quelle erreur commet-elle?
- Quand il y a des erreurs, celles-ci sont-elles homogènes?

On peut bien sûr répondre à ces trois questions pour chacune des méthodes, et nous avons tenté de le faire pour chaque algorithme que nous avons décris. Mais ce qui nous intéresse maintenant est d'en comparer plusieurs. Or, comparer signifie sortir du cadre de l'évaluation qualitative décrite par [Fiume et al.83] pour proposer une analyse quantitative permettant d'ordonner les algorithmes. Il faudrait donc commencer par comparer les erreurs potentielles (*i.e.* les cas que l'algorithme ne traite pas correctement) en considérant que certains défauts sont plus *graves* que d'autres. Puis, indépendamment de la nature de l'erreur, il faudrait tenir compte de sa fréquence sur une image. Nous arriverions là à un critère très subjectif puisque dépendant de la scène. Enfin, en supposant que nous parvenions à classer les différents défauts susceptibles d'apparaître, et que nous ayons un jeu de scènes suffisamment grand et représentatif (de quoi au fait?) pour qu'il nous serve à faire des statistiques sur la fréquence d'apparition des erreurs recencées, il nous faudrait alors décider s'il vaut mieux une image avec une seule grosse erreur ou une image avec beaucoup de petites erreurs.

En fait, n'ayant à notre disposition que les représentations *numériques* des images , toutes ces décisions ne pourraient être qu'arbitraires. Or, il faut garder à l'esprit que l'antialiassage de bords de polygones, qui est de ce point de vue très différent du filtrage de texture (il est très facile par exemple, de comparer des techniques de préfiltrage *cf. 5.1*), n'est qu'un artifice qui permet d'obtenir un résultat visuellement plaisant. D'ailleurs les défauts générés par tel ou tel algorithme d'antialiassage auront plus ou moins d'importance selon le contexte visuel dans lequel ils se trouvent (la nature de la scène), ce qui ajoute encore à la difficulté d'évaluation. D'autre part, certains défauts comme le décalage de quelques sous-pixels dans une direction de tout un objet, pourrait passer comme un défaut important alors que dans certains cas il peut paraître complètement inaperçu. Enfin, les erreurs numériques qui pourraient être

détectées quant à la valeur des couleurs de certains pixels en rapport avec une valeur qui servirait de référence (calculée par une *bonne méthode*) peuvent s'avérer très éloignées de ce qu'un utilisateur perçoit. Quel que soit le modèle utilisé, les écarts entre deux valeurs dans le système de représentation des couleurs utilisé peuvent être très différents des sensations en terme de perception visuelle.

En conséquence et par déontologie, nous nous sommes refusés à effectuer ce genre de travail d'analyse qui aurait bien entendu pu donner des résultats numériques présentables mais dont la pertinence aurait été tout à fait contestable. Cela dit, notre incapacité à établir une méthode fiable d'évaluation et de comparaison des techniques d'antialiassage restera le grand regret de cette thèse.

Troisième partie

Notre proposition

Chapitre 11

L'analyse du problème

Dans ce chapitre nous commençons par présenter deux approches différentes pour analyser les problèmes d'aliassage liés au Z-Buffer. Nous précisons ensuite les choix que nous avons faits et qui sont à la base de notre démarche.

11.1 L'approche *antialiassage au vol*

Dans son analyse du problème de l'antialiassage de bords de polygones, Djamchid Ghazanfarpour [Ghazanfarpour85] reprend le découpage classique des algorithmes de rendu les plus usités, en deux catégories :

- les algorithmes de rendu projectif utilisant le Z-Buffer pour l'élimination des parties cachées,
- les algorithmes de lancer de rayons.

Dans la première partie qui est celle qui nous intéresse, après avoir rappelé les propriétés importantes du *tampon de profondeur*, il liste trois types de problèmes qui en découlent.

Les propriétés du Z-Buffer

- Les primitives sont affichées dans un ordre quelconque, le résultat ne dépend pas de cet ordre.
- Les intersections entre polygones sont naturellement prises en compte par la méthode.

Les problèmes d'aliassage potentiels liés au Z-Buffer

- Changement de couleur de fond.
- Les polygones adjacents.
- Intersection de polygones (*i.e.* les arêtes implicites).

Sans que cela soit formulé explicitement, l'hypothèse qui est faite est que les polygones sont convertis en pixels à l'aide d'un échantillonnage surfacique et qu'ils sont antialiassés au fur et à mesure de l'affichage à la manière des algorithmes de composition d'images (*α-blending*). Les trois cas décrits représentent d'ailleurs l'ensemble des problèmes rencontrés avec l'algorithme de [Porter et al.84] lorsque les primitives ne sont pas préalablement triées.

11.2 Notre approche

Pour notre part, nous préférons analyser le problème sous un autre angle. Lorsqu'une scène est rendue à l'aide d'un Z-Buffer à la résolution du pixel (*i.e.* sans sur-échantillonnage), un seul objet est stocké. Plusieurs cas de figure sont alors possibles :

1. l'objet le plus proche couvre entièrement le pixel,
2. l'objet le plus proche couvre partiellement le pixel sur un fond différent (*figure 11.1 cas A*),
3. l'objet le plus proche couvre partiellement le pixel ainsi qu'un (ou plusieurs) autre(s) polygone(s) qui lui est (sont) adjacent(s) (*figure 11.1 cas B*),
4. l'objet le plus proche couvre partiellement le pixel ainsi que d'autres polygones intermédiaires sur un fond différent (*figure 11.1 cas C*).

Les cas 2 et 3 sont alors des sources d'aliassage, la figure 11.1 décrit avec plus de détails les cas de figure potentiels. Nous avons pris comme hypothèse que toutes les scènes ont un fond (ayant comme caractéristique une couleur et une profondeur) que nous nommons Obj_f. Nous appelons Obj_z la primitive conservée par le Z-Buffer.

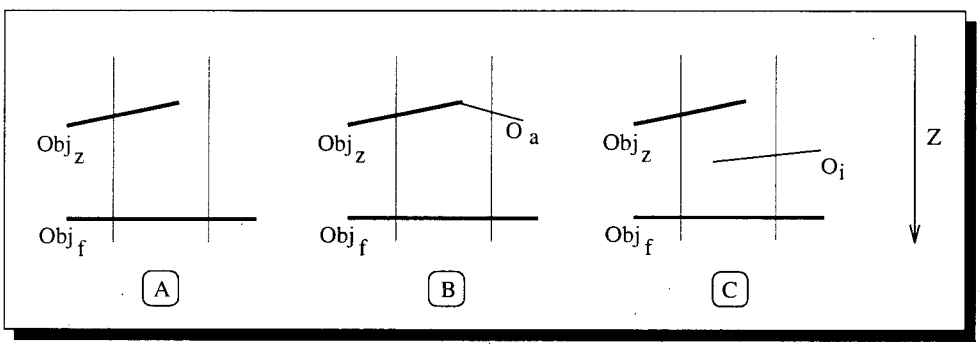


FIG. 11.1 - *Les 3 cas simples possibles*

Un bon algorithme d'antialiassage doit donc traiter les bords de polygones sur un fond, les polygones adjacents, et les arêtes implicites. Présentés comme nous venons de le faire, ces trois cas sont facilement identifiables et faciles à traiter. Un double Z-buffer par exemple, qui garderait les deux objets les plus proches, associé à un échantillonnage surfacique pourrait très facilement traiter les deux premiers. Les arêtes implicites pourraient également être prises en compte de cette façon comme nous le verrons par la suite. Malheureusement, les différents cas de figure peuvent se combiner entre eux de plusieurs façons :

- dans un même pixel, les polygones adjacents peuvent être plus de deux (cas des sommets de facettisation par exemple),
- le fond peut lui-même être constitué de polygones adjacents,
- plusieurs autres objets (adjacents entre eux ou non) couvrant partiellement le pixel, peuvent se situer entre le fond et l'objet le plus proche,

- aux sommets des arêtes de facettisation (*respectivement* implicites), les polygones adjacents (*respectivement* intersectants) peuvent ne couvrir qu'une partie du pixel et donc laisser visible un fond différent.

Ajoutons à cela que le Z-buffer stocke les caractéristiques (*i.e.* z, rvb) d'une *primitive d'affichage*, c'est-à-dire une facette.

Or, nous avons jusqu'ici simplifié nos explications en considérant que les objets que nous manipulons sont de couleurs différentes. Ce n'est bien sûr pas toujours le cas. Nous appelons *arête de facettisation* une arête qui délimite deux facettes issues de la même *primitive de modélisation*. De telles facettes sont susceptibles d'avoir la même couleur sur les pixels de l'arête qui les unit. Ceci complique donc encore davantage ce que nous avons présenté sachant que tout ce que nous avons appelé *Objet* peut être constitué de plusieurs facettes.

Toutes ces combinaisons sont autant de cas particuliers délicats à traiter, la difficulté étant de trouver une technique générale permettant

- de résoudre le maximum de cas,
- de faire des choix judicieux lors de certaines approximations de toute façon inévitables.

Notons par ailleurs que, dans notre analyse, le changement de couleur de fond n'est pas un problème en soi.

11.3 Choix et propositions

11.3.1 Les choix liés au contexte

Tout d'abord, il apparaît que les solutions à l'aliassage sont diverses et variées (du sur-échantillonnage au A-Buffer) et que chacune d'elles possède des avantages et des défauts variables selon le type d'application. Il est donc important de toujours préciser le contexte, c'est-à-dire essentiellement le type d'applications visées par l'algorithme de rendu. Les critères d'évaluation pour un simulateur de vol temps réel et un algorithme de radiosité utilisé pour simuler des éclairements en architecture d'intérieur seront très différents, même si pour les deux applications l'aliassage est un réel problème.

En résumé, deux directions peuvent être privilégiées : la rapidité du rendu au prix de quelques approximations, et la qualité irréprochable de l'image au détriment des temps de calcul. Historiquement, l'équipe Graphix du LIFL au sein de laquelle j'ai réalisé ces travaux s'est intéressée à la synthèse d'images en temps réel. Actuellement, seul ce que l'on appelle le pipeline de rendu *Gouraud/Z-Buffer* permet de calculer les images suffisamment vite pour un affichage interactif, car c'est l'unique méthode câblée. C'est donc vers un compromis entre la rapidité d'affichage et la qualité des images que nos recherches se sont dirigées.

Le sur-échantillonnage écarté

Dès le départ, nous avons écarté le sur-échantillonnage sous toutes ses formes pour deux raisons : la première est bien évidemment liée au facteur multiplicatif pour les temps de calcul des images dans le cas d'un sur-échantillonnage de la scène complète. D'autre part, un des principaux objectifs est la possibilité de câbler les algorithmes en vue de les inclure dans une machine spécialisée (au départ, feu la machine IMOGENE [Chaillou91]) ou d'en faire une carte spécialisée dans le rendu antialiassé pour des machines d'usage général (PC, station de

travail,...). Cette contrainte élimine le sur-échantillonnage adaptatif en imposant un aspect systématique (tous les pixels sont traités de la même façon) affranchi de tout cas particulier.

L'échantillonnage surfacique bien adapté

À partir de là, l'échantillonnage surfacique nous a paru une alternative intéressante tant au niveau de la qualité que du point de vue de l'adéquation avec les autres algorithmes intervenant dans le rendu. Nous avons particulièrement regardé les liens qui existent entre l'antialiassage de bords de polygones et l'antialiassage des textures plaquées.

Comme nous l'avons vu plus haut, le sur-échantillonnage peut être utilisé comme remède aux deux problèmes. Néanmoins, dans le cas des moirés sur des textures mises en perspective, il donne des résultats médiocres et seules les très hautes résolutions sont capables d'atténuer les moirés. En fait, pour ce problème précis, les solutions de pré-filtrage (de type *mip-mapping* ou *Summed Area Tables*) sont très efficaces et de faible coût, ce qui tend à les généraliser. Nous pensons dès lors qu'il serait peu judicieux d'utiliser un sur-échantillonnage pour les bords de polygones alors que les textures sont pré-filtrées. Précisons que ce choix est bien entendu lié au type de rendu qui nous concerne à savoir le rendu projectif. En lancer de rayons, les méthodes de préfiltrage de textures sont inefficaces, le sur-échantillonnage est donc la seule solution pour tous les types de défauts.

Nous sommes donc convaincus que l'échantillonnage surfacique, qui se combine très bien avec le pré-filtrage des textures, est une solution intéressante pour l'antialiassage de scènes facettisées dans le cadre d'un rendu rapide.

11.3.2 Les choix liés à l'analyse de l'existant

Dans un second temps, l'analyse des différents algorithmes d'élimination des parties cachées associés à l'échantillonnage surfacique nous a conduit au raisonnement suivant : le A-Buffer qui est sans aucun doute l'algorithme qui permet d'obtenir les meilleurs résultats en terme de qualité de l'image est beaucoup trop lourd à gérer (en particulier la gestion des listes chaînées de fragments triés en z) pour supporter une quelconque implémentation matérielle. Notre approche doit donc s'orienter vers une solution qui traiterait au vol les différentes primitives, une solution que l'on pourrait appeler *à la Z-Buffer*. La solution des algorithmes de composition d'images sont de cet ordre et correspondent en quelque sorte à ce que nous cherchions. Malheureusement, si ces techniques sont assez bien adaptées au dessin animé ou à l'affichage en représentation *fil de fer*, elles le sont beaucoup moins à la synthèse 3D pour toutes les raisons que nous avons évoquées plus haut.

Notre idée a d'abord été de rester le plus proche possible du Z-Buffer en développant une méthode à voisin [Preux et al.92]. Nous avons apporté plusieurs améliorations par rapport aux méthodes existantes de ce type. La conclusion de ces premiers travaux est la suivante : les méthodes à voisin donnent de très bons résultats visuels, en particulier grâce aux améliorations que nous avons apportées dans le choix du voisin, pour des scènes *simples* c'est-à-dire constituées de grandes facettes. Par contre, dans le cas de scènes finement facettisées et comprenant des objets fins cette méthode présente des limites intrinsèques. Nous avons donc entamé de nouveaux travaux en se fixant comme objectif de trouver un compromis entre la simplicité du Z-Buffer et l'efficacité du A-Buffer.

11.3.3 La notion de continuité

Intéressons-nous maintenant aux " *défauts*" de l'image dûs au crénelage des arêtes. Il est classique de faire l'énumération suivante :

- marches d'escalier sur les bords de polygones,
- trous dans les objets longs et fins,
- apparition et disparition des petits objets en animation.

Dans les trois cas, la notion de continuité est présente. Continuité spatiale pour les bords d'objets et les objets fins, continuité spatiale et temporelle pour le clignotement des petits objets. Il apparaît clairement qu'un bon algorithme d'antialiassage doit avant tout éviter de briser les continuités.

Lorsque sur une arête correspondant à une silhouette d'objet, les pixels directement adjacents au pixel courant représentent encore la même silhouette, alors nous disons qu'il y a continuité. Par contre, dès qu'un point se trouve *au bout* d'une arête, c'est à dire que l'un des voisins n'appartient plus à la même silhouette (à cause d'un recouvrement par exemple), nous parlons de discontinuité de la scène. Dans ce type de situation, quand un bord d'objet en recouvre un autre, il est très important de ne pas briser la continuité de l'objet qui se trouve au premier plan : à l'endroit de l'intersection, l'approximation devra porter sur les couleurs de fond (qui sont de toute façon victime d'une discontinuité) plutôt que sur la couleur de l'objet recouvrant.

11.3.4 Surface *vs* facette

D'autre part, si l'aliassage apparaît sur certains bords des polygones, il n'est pas présent sur tous les bords. En effet, il faut distinguer les bords de facette qui correspondent également à des bords d'objets de la scène (ces arêtes définissent la *silhouette* des objets), et les *arêtes de facettisation* qui séparent deux triangles de même couleur appartenant au même objet de la scène[1]. Cette remarque est particulièrement importante pour les algorithmes de préfiltrage de type A-buffer, car le fait de stocker des valeurs pour chaque facette présente dans un pixel, oblige à tenir compte parfois d'un grand nombre de primitives pour finalement s'apercevoir qu'une seule couleur couvre le pixel. Le contraire ne pouvant pas se produire, il est plus judicieux de stocker des couleurs que des objets.

11.3.5 Le calcul du taux de couverture

Le générateur de pixels que nous avons utilisé pour l'implémentation des différents algorithmes présentés par la suite, est basé sur celui décrit dans [Schilling91]. C'est un algorithme d'affichage par test d'inclusion. Précisons que nous l'avons choisi pour les raisons suivantes : la composante α est calculée avec une grande précision (1/16 de pixel), l'algorithme est identique quels que soient les cas de figure (pas de traitement particulier pour les sommets de polygones) ce qui permet une implémentation matérielle. Nous donnons le détail des choix que nous avons faits (*cf.* §*14.1*). Notons toutefois que notre algorithme d'élimination des parties

1. Notons qu'il est impossible de savoir *a priori* si une arête de facettisation correspond à un bord d'objet de la scène qu'il faut antialiasser ou non. En particulier pour les objets volumiques facettisés cela dépend du plan de projection de la scène.

cachées est compatible avec n'importe quel générateur de pixels fournissant les valeurs z,r,v,b et α .

11.4 Conclusion

Suite aux considérations exprimées ci-dessus, nous avons voulu proposer une solution à l'antialiassage de bords de polygones en utilisant un échantillonnage surfacique et un algorithme d'élimination des parties cachées à mi-chemin entre le Z-Buffer et le A-Buffer. Les idées de départ sont les suivantes :

- Ne garder qu'un nombre limité d'objets en chaque pixel,
- Stocker ces différents objets *au vol* sans tri préalable et sans gérer une liste triée,
- Toujours garder l'objet le plus proche, quel que soit son taux de couverture,
- Utiliser la notion de surface à la place des facettes, à l'aide d'un opérateur de *cumul*, comparable au *merging* du A-Buffer,
- Calculer la couleur finale des pixels, lors d'une seconde passe, en combinant les différentes couleurs stockées avec leur composante α.

À partir de ces différentes assertions, il existe de nombreuses possibilités. Nous tentons, dans les chapitres suivants, de faire le point en comparant avantages et inconvénients des diverses solutions.

Chapitre 12

Deux couleurs

Nous présentons dans ce chapitre deux algorithmes de rendu surfacique, prenant en compte deux couleurs par pixel. Tous deux conservent l'objet le plus proche avec son coefficient α comme première couleur. La seconde couleur correspond à celle de l'objet le plus proche, couvrant entièrement le pixel, dans le cas du premier algorithme. Pour le deuxième algorithme, qui est une amélioration (surtout une simplification) du précédent, la seconde couleur est celle du deuxième objet le plus proche. Les deux techniques utilisent le cumul des valeurs α lorsque plusieurs facettes adjacentes dans un pixel, correspondent à la même surface.

12.1 Deux couleurs et trois mémoires

Nous avons proposé dans [Preux et al.94] une solution qui ne considère que deux couleurs par pixel. Elle utilise deux mémoires complètes, plus une mémoire de profondeur pour des traitements intermédiaires. Après une première description au niveau fonctionnel, nous présentons l'algorithme en expliquant son mécanisme. Nous faisons ensuite quelques remarques quant à ses performances et limites.

12.1.1 Description fonctionnelle

À partir d'un échantillonnage surfacique, nous proposons un nouvel algorithme d'élimination des parties cachées qui permet un affichage direct de polygones antialiassés, tout en gardant la rapidité et la souplesse du Z-Buffer (les primitives sont traitées une seule fois et dans n'importe quel ordre).

L'algorithme repose sur deux principes de base. D'une part, un mécanisme d'accumulation des coefficients α permet, dans le cas des pixels traversés par des arêtes de facettisation, de ne stocker qu'une seule fois la couleur et la valeur de profondeur avec la couverture totale du pixel. D'autre part, dans tous les cas où l'objet qui se trouve le plus près de l'observateur ne couvre pas entièrement le pixel, une deuxième couleur est stockée. Elle correspond en principe, soit à la couleur de fond, soit à la couleur d'un objet adjacent

Lorsque plus de deux couleurs sont visibles dans un pixel, notre algorithme fonctionne en *mode dégradé* puisque nous n'en conservons que deux. Par ailleurs, notons que l'algorithme, qui fonctionne de manière identique en tous les pixels de l'écran, ne nécessite aucune opération complexe puisqu'il est, comme le Z-Buffer, exclusivement basé sur des comparateurs (plus deux additionneurs).

Afin de stocker les deux couleurs, deux mémoires d'image sont nécessaires : le *FrontBuffer* qui contient la couleur, la profondeur et le taux de couverture de l'objet se trouvant le plus près de l'observateur et le *BackBuffer* qui mémorise soit la profondeur et la couleur de l'objet le plus proche couvrant entièrement le pixel (la couleur de fond), soit la profondeur et la couleur d'un objet adjacent à celui qui se trouve dans le *FrontBuffer*.

Lorsque toutes les primitives de la scène ont été traitées, le *FrontBuffer* et le *BackBuffer* contiennent les deux couleurs à prendre en compte ainsi que leur taux de couverture. Il ne reste plus qu'à calculer la couleur finale à afficher par interpolation pondérée par le coefficient α en tous les pixels. Il faut pour cela parcourir les deux mémoires en parallèle afin de calculer la couleur de chaque pixel à afficher selon la formule classique :

$$CouleurFinale_{r,v,b} = (RVB_{front} \times \alpha_{front}) + (RVB_{back} \times (1 - \alpha_{front}))$$

Comment déterminer la couleur de fond

Chaque primitive est traitée *au vol*, à la manière du Z-Buffer, c'est-à-dire que ses attributs (*i.e.* z, rvb, α) sont soit stockés dans l'une des mémoires, soit définitivement perdus. Il se pose donc le problème de la détection de changement de couleur de fond lorsque celui-ci est constitué de la juxtaposition de plusieurs facettes. Lorsqu'une de ces facettes est traitée, si sa profondeur est supérieure à celle de l'objet mémorisé dans le *FrontBuffer*, elle n'est pas conservée dans celui-ci. Et comme elle ne couvre pas, à elle seule, la totalité du pixel, elle n'est pas conservée non plus dans le *BackBuffer*. Elle se retrouve donc définitivement perdue. Les autres facettes adjacentes subissent le même sort. Le changement de couleur de fond n'est donc pas détecté, et une erreur se produit à l'affichage.

Pour palier ce problème, nous utilisons une troisième mémoire, que nous appelons *CurrentBuffer*, qui ne conserve qu'une information de profondeur. Lorsqu'un polygone arrive, qu'il ne couvre pas entièrement le pixel, et qu'il se situe en profondeur entre les objets déjà stockés dans le *frontBuffer* et le *BackBuffer*, nous gardons sa profondeur dans le *CurrentBuffer*. La technique du cumul des coefficients α, que nous utilisons déjà dans le *FrontBuffer*, permet d'additionner les taux de couverture de toutes les facettes adjacentes issues d'une même surface. Si avec l'arrivée d'une n^{ieme} facette, la couverture atteint la valeur 1, alors la couleur et la profondeur de cette primitive remplacent les valeurs stockées dans le *BackBuffer*. Par contre, si une facette issue d'une autre surface (*i.e* pas située à la même profondeur) que celle déjà présente dans le *CurrentBuffer* alors celle-ci la remplace, il n'y a pas de changement de couleur de fond.

Bien que nous ayons utilisé le même vocabulaire, remarquons que ce problème n'a rien à voir avec le *changement de couleur de fond* décrit au §11.1.

12.1.2 Détails de l'algorithme

Nous allons maintenant expliquer dans le détail le fonctionnement de l'algorithme. Nous utiliserons les notations suivantes : z, rvb, α sont les attributs du nouveau pixel et Z_f, RVB_f, α_f (resp. Z_c, RVB_c, α_c et Z_b, RVB_b, α_b) représentent la profondeur, la couleur

et le taux de couverture du *FrontBuffer* (resp. du *CurrentBuffer* et du *BackBuffer*).

Si $(z > Z_b)$ **Alors**
  Stop ; // *objet non visible*
Si $(z < Z_f)$ **Alors**
  FrontBuffer $\leftarrow$ pixel ; // *objet le plus proche de l'observateur*
Si $((Z_f < z < Z_b)$ et $(z \neq Z_c)$ et $(\alpha < 1))$ **Alors**
  CurrentBuffer $\leftarrow$ pixel ; // *nouvel objet dans le CurrentBuffer*
Si $((z < Z_b)$ et $(\alpha = 1))$ **Alors**
  BackBuffer $\leftarrow$ pixel ; // *changement de couleur de fond*
Si $((z = Z_f))$ et $(rvb \neq RVB_f))$ **Alors**
  BackBuffer $\leftarrow$ pixel ; // *polygones adjacents de couleurs differentes*
Si $((z = Z_f)$ et $(rvb = RVB_f))$ **Alors**
  $\alpha_f = \alpha_f + \alpha$; // *polygones adjacents de même couleur*
  Si $(\alpha_f = 1)$ **Alors**
    BackBuffer $\leftarrow$ pixel ;
Si $(z = Z_c)$ **Alors**
  $\alpha_c = \alpha_c + \alpha$; // *polygones adjacents de couleurs eventuellement differentes*
  Si $(\alpha_c = 1)$ **Alors**
    BackBuffer $\leftarrow$ pixel ;

ALGO. 2 - *Notre algorithme à deux couleurs et trois mémoires*

Remarque 1 : Dans l'algorithme, certains tests ont été écrits sous la forme *si* $(z1 = z2)$ pour des raisons de clarté. Il ne faut cependant pas les prendre comme un test d'égalité stricte. Ils symbolisent le test d'intersection de deux plans dans un pixel. Des détails sont donnés sur ces tests dans le §12.4.

Remarque 2 : Les sept tests de l'algorithme couvrent l'ensemble des cas possibles, mais ne sont pas exclusifs les uns par rapport aux autres. Ce qui signifie qu'il faut voir ces différents cas comme un algorithme parallèle et non séquentiel. Ceci afin que les opérations effectuées après les premiers tests ne faussent pas les tests suivants.

12.1.3 Quelques explications

Afin de faciliter la compréhension de l'algorithme, nous allons reprendre séparément ce qui se passe pour chaque partie de la mémoire.

- **Dans le FrontBuffer :**
  Le but est de stocker la couleur de l'objet se trouvant le plus près de l'observateur quelle que soit sa couverture.
  si $(z < Z_f)$ FrontBuffer $\leftarrow$ pixel ;
  si $(z = Z_f)$ ET $(rvb = RVB_f)$ nous sommes dans un cas de polygones adjacents de même couleur, il faut donc cumuler le nouveau taux de couverture avec celui déjà stocké. De plus, le résultat du cumul doit être testé. En effet, si celui-ci atteint 1, cela signifie que le pixel est entièrement couvert, c'est à dire qu'il y a un changement de couleur de fond.
  si $(z = Z_f)$ ET $(rvb \neq RVB_f)$ cette fois c'est un polygone adjacent de couleur différente.

Nous supposons que sa couleur couvre le reste du pixel et les nouvelles valeurs sont rangées dans le BackBuffer.
si $(z > Z_f)$ la couleur n'est en aucun cas prise en compte dans le FrontBuffer.

- **Dans le BackBuffer :**
  Cette mémoire stocke la couleur de l'objet le plus proche couvrant entièrement le pixel.
  si $(z < Z_b)$ ET $(\alpha = 1)$ BackBuffer $\leftarrow$ pixel ;
  De plus, chaque fois qu'un cumul de α (dans le FrontBuffer ou dans le CurrentBuffer) atteint 1, le pixel est stocké dans le BackBuffer car c'est un changement de couleur de fond. D'autre part, le BackBuffer sert également à stocker des couleurs couvrant partiellement, dans certains cas de polygones adjacents. Le fonctionnement et les conséquences de ce mécanisme sont décrits dans le paragraphe suivant.

- **Dans le CurrentBuffer :**
  Cette mémoire va nous permettre de stocker des couleurs d'objets se trouvant à une profondeur intermédiaire entre le FrontBuffer et le BackBuffer mais ne couvrant pas entièrement le pixel. Si plusieurs facettes interviennent partiellement dans le même pixel avec la même profondeur, leur couverture est sommée afin de détecter un éventuel changement de couleur de fond.
  A chaque fois que le z du nouveau pixel est identique à celui déjà stocké (polygones adjacents) les taux de couverture sont cumulés comme dans le FrontBuffer. Si le cumul atteint 1, le pixel est rangé dans le BackBuffer, mais si un autre objet intermédiaire avec une couverture partielle et un z différent se présente, alors il remplace l'ancien :
  si $(Z_f < z < Z_b)$ ET $(z \neq Z_c)$ CurrentBuffer $\leftarrow$ pixel ;
  Il apparaît que l'algorithme impose une contrainte supplémentaire (mais naturelle), qui est le traitement séquentiel des différentes primitives de chaque objet de la scène.

12.1.4 Qualité de l'antialiassage

La qualité de l'antialiassage des algorithmes de préfiltrage dépend en premier lieu de la précision avec laquelle est calculé le coefficient α. Nous avons implémenté notre algorithme avec le générateur de pixels décrit dans [Schilling91]. L'auteur montre que la précision obtenue est au moins aussi bonne, voire meilleure dans certains cas, qu'avec un suréchantillonnage 4×4 régulier. Les détails de ces calculs sont donnés au §14.1.

Cependant, ne prenant en compte que deux couleurs par pixel, cette qualité n'est plus préservée lorsqu'au moins trois couleurs sont visibles. La couleur finale est alors une approximation calculée à partir des deux couleurs les plus significatives.

Nous faisons maintenant le point sur les différents cas de figure possibles afin d'estimer l'efficacité de notre algorithme.

Les bords d'objet sur un fond

Les marches d'escalier sur le contour des objets d'une scène est évidemment le cas de crénelage le plus répandu. Notre algorithme le traite parfaitement quel que soit l'ordre d'arrivée des primitives. Notons également que les objets (celui défini par le contour, aussi bien que l'objet représentant le fond) peuvent être découpés en facettes. Grâce au *cumul* des α, l'algorithme considère au moment de la dernière étape (*i.e* le calcul de la couleur finale) les couleurs et les taux de couverture des *surfaces* et non celles des *facettes*.

Les arêtes implicites

Une arête implicite est un segment créé par l'intersection de deux polygones. L'équation de la droite support ainsi que les extrémités ne sont donc pas connues a priori. Cette difficulté pose des problèmes à tous les algorithmes de préfiltrage, seul le suréchantillonnage traite ces cas naturellement. Dans [Schilling et al.93], l'auteur décrit une méthode de détection et d'antialiassage par préfiltrage des arêtes implicites. Cette technique est entièrement compatible avec notre algorithme. Les informations géométriques sont calculées comme pour les bords de polygones en une seule passe par l'intermédiaire de tables d'indirection.

D'autre part, la faiblesse des algorithmes de préfiltrage qui nécessitent un traitement particulier pour les arêtes implicites, face à l'aspect universel du suréchantillonnage, doit être relativisée par des considérations portant sur la modélisation. Dans le cas des scènes facettisées après modélisation, la base de données de facettes est souvent représentée à l'aide de structures de type B-Reps[1], ce qui exclut de fait la présence d'arêtes implicites.

Les polygones adjacents

Il est intéressant de noter que le *BackBuffer* peut avoir une seconde utilisation. En effet, dans le cas où le pixel est couvert par deux polygones adjacents de couleurs différentes, nous pouvons stocker une des deux couleurs (qui ne correspond pourtant pas à la couleur de fond) dans le *BackBuffer* sachant que l'autre est rangée avec sa composante α dans le *FrontBuffer*. Au moment de l'interpolation des couleurs, celle stockée dans le *BackBuffer* sera prise en compte avec une pondération de $(1 - \alpha)$, ce qui correspond exactement à son taux de couverture. Précisons maintenant ce que fait notre algorithme dans le cas des polygones adjacents.

- **Dans le FrontBuffer :**
  Quand tous les polygones sont de la même couleur, les autres mémoires ne sont pas affectées et tout se passe comme si il y avait une seule facette avec un taux de couverture plus important. Par contre, si une facette d'une autre couleur, mais de même profondeur se présente, sa couleur (ainsi que sa profondeur) est stockée dans le BackBuffer. Ce mécanisme permet de traiter parfaitement les arêtes entre deux polygones adjacents de couleurs différentes. Nous pouvons remarquer qu'aux extrémités des arêtes, l'une des deux couleurs sera surestimée (au détriment de la couleur de fond par exemple, *cf. exemple 2 Figure 2*).

- **Dans le CurrentBuffer :**
  Remarquons tout d'abord que le CurrentBuffer ne stocke que la profondeur et la composante α des objets. Lorsque tous les polygones sont de la même couleur, si le cumul des taux de couverture atteint 1, le pixel traité est rangé dans le BackBuffer. La couleur n'est donc pas perdue. Par contre, le contenu du CurrentBuffer pouvant au mieux se retrouver en couleur de fond dans le BackBuffer, il est impossible (puisque nous ne gardons que deux couleurs dont l'une est stockée dans le FrontBuffer) d'en stocker plus d'une. Cela oblige dans le cas d'une couverture complète par plusieurs couleurs à en choisir une arbitrairement. C'est la couleur du dernier polygone (celui qui finit de couvrir le pixel) qui est choisie puisque les autres n'ont pas été conservées.

1. B-Rep : Boundary Representation

12.1.5 Les approximations

Dans le cas de deux couleurs visibles

L'ensemble des cas où deux couleurs sont visibles dans le pixel (*cf.* figure 12.1) sont traités correctement, à l'exception d'un seul : celui décrit sur l'exemple 3. Dans cette situation, le polygone foncé se retrouve stocké dans le *CurrentBuffer* puisqu'il est plus profond que l'objet *A* rangé dans le *FrontBuffer* mais qu'il ne couvre pas entièrement le pixel. Il n'intervient donc pas dans le calcul de la couleur finale. C'est la couleur de l'objet *B* (stocké dans le *BackBuffer*) qui est utilisée. Notons que ce résultat ne dépend pas de l'ordre d'arrivée des polygones.

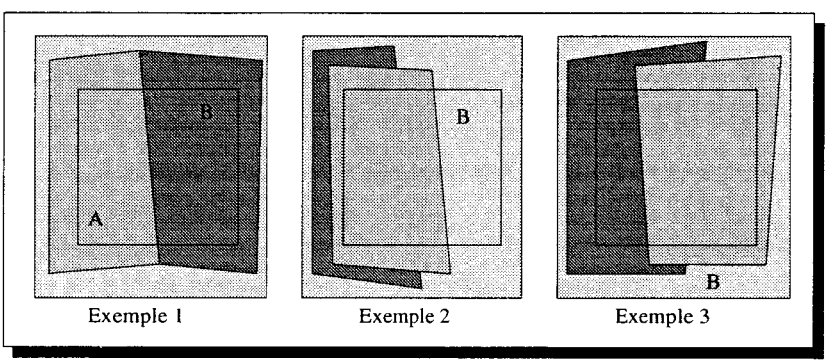


FIG. 12.1 - *Les différentes possibilités avec deux couleurs*

Dans le cas d'au moins trois couleurs visibles

Lorsque plus de deux couleurs sont visibles dans un pixel, notre algorithme ne peut pas donner le résultat exact, puisque deux couleurs seulement sont prises en compte dans le calcul de la couleur finale. Examinons les différents cas de figure avec trois couleurs visibles :

- un bord de polygone sur un objet intermédiaire de couverture partielle plus importante, sur un fond différent (*figure 12.2 exemple 1*),
- deux sommets de polygones adjacents (de couleur différente) sur un fond d'une troisième couleur (*figure 12.2 exemple 2*),
- trois facettes adjacentes de couleur différente couvrant à elles trois l'ensemble du pixel (par exemple au sommet d'un cube) (*figure 12.2 exemple 3*),
- un bord de polygone sur un fond bicolore (deux polygones adjacents) (*figure 12.2 exemple 4*).

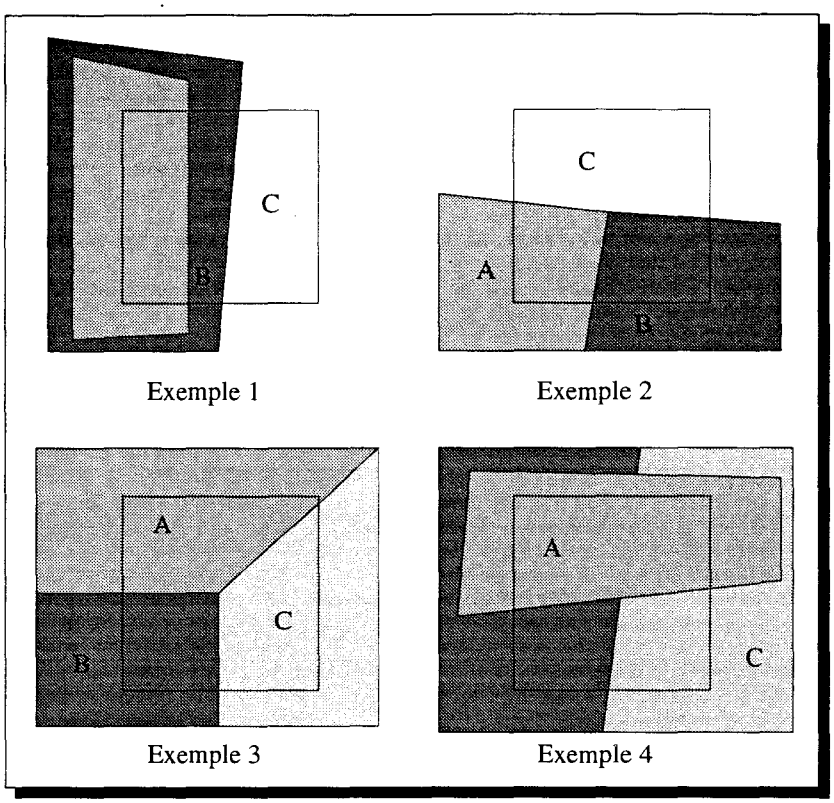


FIG. 12.2 - *Les différentes possibilités avec trois couleurs*

Dans les quatres cas, une couleur est *oubliée* au profit d'une autre qui sera *surestimée*. Dans le premier exemple, l'objet intermédiaire B n'est pas pris en compte et C intervient pour $1 - \alpha_A$. Dans le deuxième exemple, le résultat dépend de l'ordre d'arrivée des polygones : si A est traité en premier, il intervient à hauteur de sa couverture tandis que B est surestimé. Si B est le premier objet, l'inverse se produit; C est de toute façon oublié. Sur l'exemple 3, le premier objet traité est stocké dans le *FrontBuffer* et le dernier se retrouve dans le *BackBuffer* en écrasant le deuxième. Enfin, dans l'exemple 4 la première des deux couleurs de fond est surestimé et intervient pour $1 - \alpha_A$.

Les cas de pixels où plus de trois couleurs sont visibles (ces situations sont d'ailleurs assez rares) sont approximés sur les mêmes principes.

12.2 Deux couleurs et deux mémoires

Une autre version à deux couleurs par pixel peut être proposée. Au lieu de garder l'objet le plus proche et l'objet le plus proche couvrant entièrement, le choix peut être de garder les deux objets les plus proches quelle que soit leur couverture respective. Il n'est plus nécessaire d'utiliser le *CurrentBuffer*, ce qui allège considérablement le coût en mémoire de cette méthode.

De plus, l'algorithme se simplifie en une sorte de double Z-Buffer, auquel on ajoute le

mécanisme de cumul :

```
Si (z > Z_b) Alors
    Stop;          // objet non visible
Si (z < Z_f) Alors
    BackBuffer ← FrontBuffer      // le premier devient le deuxieme
    FrontBuffer ← pixel;          // objet le plus proche de l'observateur
Si ((z = Z_f) et (rvb = RVB_f)) Alors
    α_f = α_f + α;                // polygones adjacents de même couleur: cumul
Si ((Z_f ≤ z < Z_b) Alors
    BackBuffer ← pixel;           // changement de la deuxieme couleur
```

ALGO. 3 - *Version 2 : les deux plus proches*

Remarquons qu'avec cette solution, les polygones adjacents de couleurs différentes sont naturellement pris en compte. Par ailleurs, le cumul n'est effectué que sur le *FrontBuffer*, le *BackBuffer* intervenant de toute façon pour $(1 - \alpha_f)$.

12.3 Les différences avec la précédente version

Les cas de polygones adjacents de couleurs différentes et les cas simples de bords de polygone sur un fond (*i.e.* sans objet intermédiaire) sont parfaitement traités comme dans la version précédente.

La différence essentielle entre les deux versions se situe sur les cas où un ou plusieurs objets de couverture partielle se trouvent à une position intermédiaire entre l'objet le plus proche et l'objet de fond (le plus proche couvrant complétement). Ces différents cas sont résumés sur la figure 12.3.

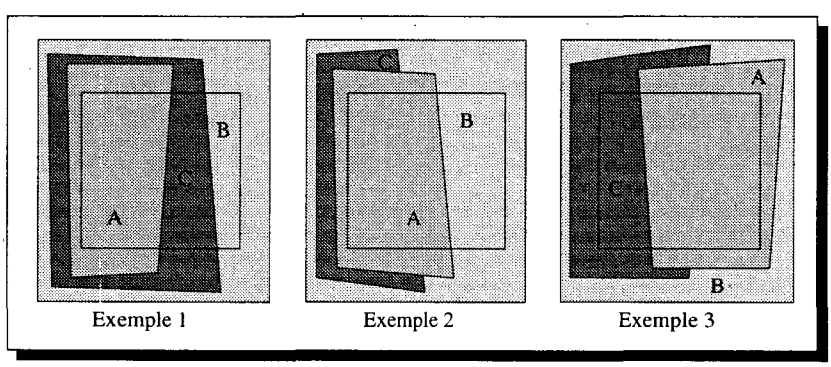


FIG. 12.3 - *les cas traités différemment*

L'exemple 1 ne peut être traité correctement ni par l'une, ni par l'autre des deux méthodes puisque trois couleurs sont visibles. L'objet A est, dans les deux cas, pris en compte avec sa composante α. La première version *oublie* l'objet C au profit de l'objet B, alors que la seconde version fait le contraire. L'erreur commise dans l'un et l'autre cas est *a priori* du même ordre.

Les exemples 2 et 3 représentent deux cas de figure qui sont également traités de manière opposée par chacune des deux versions. La première traite très bien l'exemple 2 et mal l'exemple 3, la seconde version fait exactement le contraire. Là aussi, les deux situations ont la même probabilité d'apparition, ce qui ne nous permet pas de choisir.

Les deux versions commettant le même type d'erreurs, il est tentant de préférer la moins couteuse algorithmiquement et en mémoire, c'est-à-dire la seconde. Les premiers tests que nous avons faits nous ont montré que sur les scènes que nous avons utilisées, les défauts les plus visibles étaient de deux ordres :

- Sur les sommets d'arêtes, séparant deux polygones adjacents de couleurs différentes, au moins trois couleurs sont présentes, l'approximation avec deux couleurs crée des excroissance sur ces pixels. Les deux méthodes donnent le même résultat.

- Sur les bords des objets volumiques, certains pixels *débordent* de la silhouette. Ces défauts proviennent de la combinaison de l'échantillonnage en dehors de la primitive et de notre méthode de cumul. En effet, sur les bords des objets, certaines facettes se retrouvent avec une très forte perspective. Leur largeur peut être bien inférieure au pixel et les calculs de couleur et de profondeur donnent des valeurs très éloignées de la réalité. Le cumul que nous utilisons s'effectuant à l'aide de comparaisons sur les valeurs de couleur et de profondeur, ces facettes ne sont pas *cumulées*. Elles sont donc stockées dans l'une des deux mémoires comme si elles représentaient un objet indépendant. Lorsque ce sont les deux couleurs les plus proches qui sont conservées, deux facettes adjacentes peuvent ne pas se cumuler à cause du problème que nous venons d'évoquer. Elles se retrouvent donc chacune dans une mémoire, et la couleur finale du pixel sera calculée à partir de ces deux facettes même si elles ne représentent à elles deux qu'une petite partie du pixel. La *figure 12.4* montre un exemple de ce type. Le polygone $P1$ est conservé dans le *FrontBuffer* et le polygone $P2$ devrait venir s'y ajouter. À cause du très fort gradient en Z du plan support de cette facette, la profondeur calculée peut être démesurément grande. Les tests utilisant un *seuil* pour comparer les deux valeurs sont inefficaces face à ces débordements. Dans le cas de la version à deux couleurs et deux mémoires, le polygone $P2$ passe dans la deuxième mémoire et le pixel se retrouve entièrement noir alors qu'il n'est même pas couvert à moitié. La version à trois buffers commet une erreur beaucoup moins importante en *éliminant* le polygone $P2$ qui est stocké dans le *CurrentBuffer* (et n'intervient donc pas dans la couleur finale).

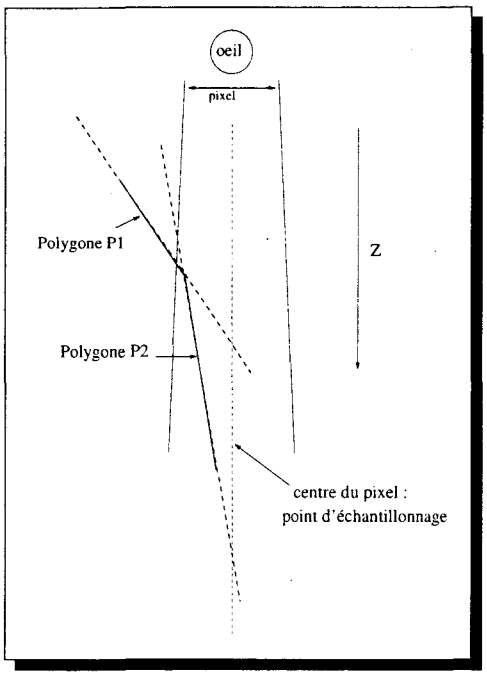


FIG. 12.4 - *Facette avec un fort gradient en Z*

En conclusion, dans les premières implémentations que nous avons effectuées, la première méthode (deux couleurs et trois mémoires) donnaient de bien meilleurs résultats que la seconde, non pas à cause de l'algorithme utilisé, mais à cause des erreurs dues à l'implémentation. Nous avons montré qu'en fait les méthodes se valent, si les détections de polygones adjacents sont correctement effectuées.

Nous dicutons maintenant des différentes méthodes permettant de détecter les arêtes communes entre deux facettes. Ce sont ces tests qui sont symbolisés dans nos algorithmes sous la forme $si(z1 = z2)$. Nous insistons sur le fait que si cette question paraît un détail d'implémentation, elle n'en est pas moins au coeur du problème. De la précision de ces tests dépend la validité des algorithmes que nous proposons. L'impossibilité à résoudre correctement ce problème rendrait caduc notre travail.

12.4 Les tests de profondeur

D'une manière générale, tous les algorithmes d'échantillonnage surfacique souffrent du manque de précision dans le calcul des valeurs de profondeur. Nous avons vu (*cf.* §*9.4.2*) que des erreurs de visibilité peuvent en résulter. Les techniques que nous proposons reposent sur la détection de deux situations particulières :

- Les cas de polygones adjacents, appartenant à la même surface, pour effectuer un cumul des coefficients α, ce sont les *arêtes de facettisation.*
- Les cas de polygones adjacents de couleurs différentes, qui se produisent fréquemment et qui doivent être correctement traités. On appelle cela des *arêtes vives.*

La même difficulté se présente dans les deux cas, à savoir repérer les facettes jointives dans un pixel. Plusieurs techniques plus ou moins efficaces peuvent être uitlisées :

12.4.1 Les seuils

La première implémentation que nous avons effectuée utilisait des *seuils* pour déterminer si deux facettes étaient adjacentes. Si la différence entre les deux valeurs de profondeur est inférieure à un seuil que nous avons fixé et que les deux facettes couvrent partiellement le pixel, alors elles sont considérées comme jointives. Si en plus les couleurs sont les mêmes, le cumul est effectué, sinon les deux objets sont adjacents mais de couleurs différentes.

Cette solution est très simple à mettre en oeuvre, mais les seuils sont déterminés empiriquement et dépendent des scènes. Lorsqu'ils sont trop petits certains polygones ne se cumulent pas, et lorsqu'ils sont trop grands, des erreurs de visibilités sont introduites (certains objets plus profonds *passent devant*). Cette solution n'est pas suffisamment précise pour traiter des scènes complexes finement facettisées.

12.4.2 Les tests d'intersection de plan

[Schilling et al.93] a proposé une méthode de détection et de traitement des arêtes implicites, cette solution a déjà été décrite au §9.4.1. On peut réutiliser cette technique dans le simple but de détecter l'intersection de deux plans. En effet, si deux facettes distinctes coupent un même pixel, et que leur plan support s'intersectent dans ce même pixel, on peut supposer que ce sont des facettes adjacentes.

En gardant les pentes du plan support de chaque objet stocké, en plus de la valeur de profondeur calculée au centre du pixel, on peut facilement déterminer si deux polygones sont adjacents dans le pixel considéré. En combinant ce test avec un autre sur les valeurs de couleur on peut aisément repérer les polygones qu'il faut cumuler et les polygones adjacents de couleurs différentes. Nous rappelons la condition à satisfaire pour que deux plans se coupent dans un pixel :

$$|z2 - z1| \leq (|dz_{2,x} - dz_{1,x}| + |dz_{2,y} - dz_{1,y}|)/2$$

Cette solution est beaucoup plus fiable que les comparaisons basées sur des valeurs *seuils*. Néanmoins, elle est sensible à certaines erreurs numériques. En effet, lorsque deux facettes jointives sont issues de la même surface plane (par exemple le dessus d'une table), les pentes et les valeurs de profondeur sont *a priori* les mêmes. Or, pour des raisons bien connues d'efficacité, les coefficients sont ramenés à une arithmétique entière, ce qui introduit une légère erreur. Il arrive donc fréquemment que certaines facettes jointives ne respectent pas le critère ci-dessus (on peut obtenir $z2 \neq z1$, alors que $dz_{2,x} = dz_{1,x}$ et $dz_{2,y} = dz_{1,y}$). Il est donc nécessaire d'ajouter également un seuil pour rattrapper ces approximations. Il faut donc utiliser :

$$|z2 - z1| \leq (|dz_{2,x} - dz_{1,x}| + |dz_{2,y} - dz_{1,y}|)/2 + \Delta z$$

où Δz est une constante qui dépend de l'intervalle sur lequel on code les valeurs de profondeur de la scène.

C'est cette technique que nous utilisons dans la dernière version de notre implémentation, elle est beaucoup plus robuste que la précédente et nous permet d'obtenir des images de très bonne qualité. Elle n'est cependant pas satisfaisante car d'une part le surcoût en mémoire est considérable (les pentes dz_x et dz_y à stocker), et d'autre part elle nécessite encore l'utilisation d'un seuil, déterminé de manière empirique.

12.4.3 Les numéros d'objet

Dans l'algorithme du A-Buffer, l'opération de *merging*, qui ressemble beaucoup à notre *cumul*, évite ce problème en utilisant des *numéros d'objets*. Deux fragments sont fusionnés en un seul, si ils ont le même numéro d'objet et qu'ils sont adjacents en profondeur dans la liste triée des fragments. Cette solution nécessite d'une part de manipuler des objets convexes, et d'autre part d'avoir une représentation hiérarchique des objets de la scène (chaque facette doit savoir à quel objet elle appartient). Un numéro différent est donné à chaque *primitive géométrique continue non auto-intersectante*.

La première version que nous avons présentée (deux couleurs et trois mémoires) différencie les polygones adjacents de même couleur et ceux de couleurs différentes. Cette distinction revient à déterminer les *arêtes vives* sur un modèle. Une arête vive est une arête séparant deux polygones qui ne doivent pas être lissés par l'algorithme d'ombrage. La solution des numéros d'objets n'est par conséquent pas suffisante pour résoudre notre problème. Il faudrait donc faire la distinction entre les *surfaces* et les *objets* en définissant une surface comme un ensemble de facettes adjacentes de même couleur, et un objet comme un ensemble de surfaces adjacentes. Mais avec cette notion d'objet, cette version a peu d'intérêt face à celle à deux mémoires (*cf.* plus haut). La simple notion d'objet est donc suffisante.

Avec ce concept, notre algorithmes s'implémente très facilement, et le test que nous avions symbolisé par :

$$si((z = Z_f)et(rvb = RVB_f))$$

devient

$$si(NumObjet = NumObjet_f)$$

Si ce test est vrai alors le cumul doit être effectué.

Cette solution est donc une solution idéale, le seul défaut est que l'algorithme de rendu n'est plus indépendant mais lié au modeleur qui fournit les scènes. Celui-ci doit en effet générer les numéros d'objets et les fournir dans le format de description de scènes.

Remarquons que peu de modeleurs fournissent les informations sur le numéro d'objet. Les scènes au format *NFF* [2] par exemple, ne contiennent pas cette information.

12.5 L'utilisation de masques de sous-pixels

Les algorithmes que nous venons de présenter n'utilisent aucune information géométrique relative à la couverture des l'objet dans le pixel.

Or, nous utilisons des masques de sous-pixels pour le calcul de la composante α. Ces masques sont donc disponibles et peuvent être stockés à la place de la valeur α. Le surcoût est de 8 bits par pixel et par mémoire (pour passer de 4 à 16).

De tels masques peuvent être utilisés de deux façons différentes :

- Au moment du calcul de couleur finale, pour ne prendre en compte que les *sous-pixels visibles*, à la manière du A-Buffer. Cette technique n'a de sens que pour les solutions avec au moins trois couleurs. En effet, lorsque deux couleurs seulement sont stockées, même si la deuxième n'a aucun sous-pixel qui *dépasse* du masque de la première, il faut l'utiliser puisqu'aucune autre n'est disponible.

2. NFF : Neutral File Format

- L'autre possibilité est d'effectuer l'élimination des parties cachées au vol, au niveau sous-pixel. Lorsqu'un objet doit être inséré dans une mémoire plus profonde que le *FrontBuffer*, il n'est pris en compte que si son masque *déborde* de ceux plus proches. Cette méthode permet de ne plus prendre en compte les objets cachés (comme sur l'*exemple 2 de la figure 12.3*), et donc de faire des choix plus judicieux quand plusieurs objets couvrent partiellement.

  Cependant, si certains cas sont mieux traités qu'avec les méthodes à α, le résultat peut être différent suivant l'ordre d'arrivée des polygones comme le montre la *figure 12.5*.

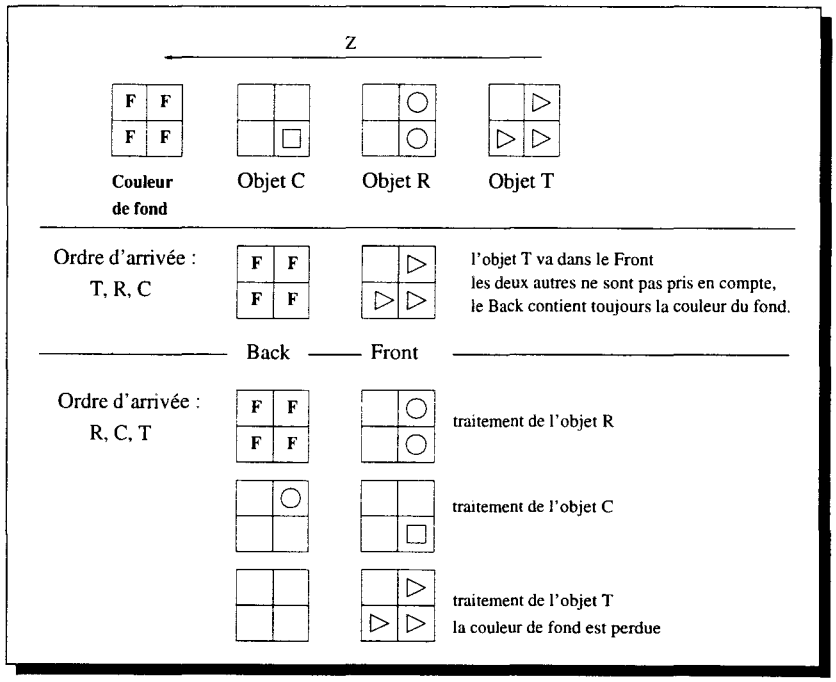


FIG. 12.5 - *Deux buffers avec masques : le résultat dépend de l'ordre d'arrivée des polygones. L'exemple est donné avec un masque* 2×2 *pour des raisons de clarté.*

Chapitre 13

Version à trois couleurs

Une des restrictions des algorithmes précédents est, nous l'avons vu plus haut, la prise en compte de seulement deux couleurs par pixel. Nous avons développé une version basée sur les mêmes principes mais qui conserverait trois couleurs au lieu de deux.

13.1 Trois couleurs et α

Dans un souci d'homogénéité, nous avons conservé les appellations *FrontBuffer*, *CurrentBuffer* et *BackBuffer* bien qu'elles ne soient plus pertinentes dans ce nouveau contexte. Ces trois mémoires représentent respectivement les trois objets les plus proches en chaque pixel.

13.1.1 Le calcul de la couleur finale

Le *CurrentBuffer* peut contenir au moment du calcul de la couleur finale la profondeur d'un objet de couverture partielle se trouvant à un distance intermédiaire entre les objets stockés dans le *Frontbuffer* et le *BackBuffer*. C'est-à-dire un objet *potentiellement* visible. En fait, il est toujours visible si sa couverture est supérieure à celle de l'objet du *FrontBuffer*, il peut être visible ou caché sinon. Dans les deux cas, la seule connaissance de la valeur α ne permet pas de dire précisément pour quelle part il doit intervenir dans la couleur finale.

Ne disposant d'aucune information géométrique sur les différents fragments (*au sens du A-Buffer*), nous devons faire un choix parmi les deux solutions suivantes :

- toujours tenir compte de l'objet stocké dans le *CurrentBuffer*, quel que soit son taux de couverture, majoré bien entendu, par la valeur $(1 - \alpha_{front})$. Dans ce cas, nous prenons le risque de faire intervenir un objet qui peut être complètement caché (par le celui du *FrontBuffer*).

- ne prendre en compte le fragment du *CurrentBuffer* que si sa composante α est plus grande que celle de l'objet stocké dans le *FrontBuffer*. Cette fois certains objets visibles n'interviendront pas dans la couleur finale. De plus, il faut encore choisir le poids qu'il faut lui accorder dans le calcul de la couleur finale.

Afin de traiter au mieux les objets de premier plan, il nous semble plus judicieux de toujours prendre en compte l'objet stocké dans le deuxième buffer, quelle que soit sa couverture. Ce choix permet de traiter correctement les polygones adjacents. Tout autre solution aurait

pour conséquence d'introduire des erreurs sur ces cas, alors qu'ils sont correctement traités par l'algorithme à deux couleurs : on aurait alors rien gagné malgré une mémoire supplémentaire.

13.1.2 L'algorithme

L'algorithme à trois couleurs se décrit donc de la manière suivante :

Si $(z > Z_b)$ **Alors**
  Stop ; *// objet non visible*
Si $((z = Z_f)$ et $(rvb = RVB_f))$ **Alors**
  $\alpha_f = \alpha_f + \alpha$; *// polygones adjacents de même couleur : cumul dans le Front*
Si $((z = Z_c)$ et $(rvb = RVB_c))$ **Alors**
  $\alpha_c = \alpha_c + \alpha$; *// polygones adjacents de même couleur : cumul dans le Current*
Si $(z < Z_f)$ **Alors**
  BackBuffer ← CurrentBuffer *// le deuxieme devient le dernier*
  CurrentBuffer ← FrontBuffer *// le premier devient le deuxieme*
  FrontBuffer ← pixel ; *// objet le plus proche de l'observateur*
Si $(Z_f < z < Z_c)$ **Alors**
  BackBuffer ← CurrentBuffer *// le deuxieme devient le dernier*
  CurrentBuffer ← pixel *// nouvelle valeur dans le Currentbuffer*
Si $((Z_c < z < Z_b)$ **Alors**
  BackBuffer ← pixel ; *// changement de la troisieme couleur*

ALGO. 4 - *Version à 3 couleurs et trois mémoires*

Nous avons réutilisé les notations **Si**$((z = Z_f) et (rvb = RVB_f))$ pour les tests de cumul, cependant ce qui a été dit plus haut (*cf.* §*12.4.3*) sur les numéros d'objets reste évidemment vrai pour cette version.

13.2 Les améliorations

Les cas à deux couleurs sont traités de la même façon que l'algorithme à deux couleurs avec α, et les cas où trois polygones adjacents de couleurs différentes sont présents dans le pixel (par exemple au sommet d'un cube), sont également correctement traités.

Un autre cas important, parce que courant et très visible lorsqu'il est traité avec seulement deux couleurs, est correctement antialiassé par l'algorithme à trois couleurs. Il s'agit du cas de deux polygones adjacents sur un fond différent (*cf. exemple 2 sur la figure 12.2*). L'algorithme à deux couleurs faisait *déborder* ces pixels en surestimant l'une des deux couleurs de premier plan. Avec trois couleurs, chacune intervient exactement pour ce qu'elle représente.

Les cas de bords d'objet sur un fond bicolore tels que celui de l'*exemple 4 de la figure 12.2* ne sont pas traités de manière exacte. Les trois couleurs peuvent intervenir dans la couleur finale, mais les proportions ne seront pas gardées. Les résultats vont dépendre de la localisation de l'objet de premier plan par rapport à la frontière qui sépare les deux autres. En moyenne, le deuxième objet sera surestimé. Nous assumons cette erreur.

13.3 Trois couleurs et masque

Nous avons envisagé l'utilisation des masques de sous-pixels pour cette version. Comme dans l'algorithme à deux couleurs, le risque est de *perdre* la couleur de fond (*cf. figure 12.5*). Cependant avec une troisième couleur il est possible de reprendre l'idée développée dans la première version (deux couleurs et trois mémoires), qui consistait à garder dans tous les cas la couleur la proche couvrant entièrement le pixel.

Ces travaux ne sont pas terminés et nous ne pouvons donner plus de résultats sur cette technique qui reste prometteuse...

13.4 Conclusion

Nous avons longtemps cru que le choix de l'algorithme serait déterminant pour la validation des méthodes que nous proposons. L'étude comparative que nous venons d'exposer, des différentes possibilités, montre notre intêret à trouver *la* méthode meilleure que les autres.

Or, aucun des algorithmes que nous proposons ne donne un résultat parfait (nous le savions depuis le début), mais les différences entre la version la plus simple et la plus sophistiquée restent mineures face aux problèmes généraux des calculs de profondeur et des débordements de couleur.

En conséquence, nous pensons qu'une version même très simple (par exemple 2 couleurs, 2 mémoires), qui donne d'excellents résultats, est une solution intéressante à condition de disposer de modèles possédant les informations nécessaires à la détection des arêtes de facettisation. En fait, les méthodes que nous présentons, qui sont à la frontière du Z-Buffer et du A-Buffer, ne peuvent réellement concurrencer le sur-échantillonnage, que si elles sont couplées à un modeleur fournissant les informations dont elles ont besoin.

Pour conclure, je répondrai à tous ceux qui me demande: *"Mais alors, laquelle de toutes ces versions est la meilleure?"*, que ce n'est pas à moi d'en décider. J'ai tenté de mettre au jour les différences de qualité entre chaque algorithme, tout en précisant les coûts mémoire et algorithmiques. Choisir une version parmi les autres, reviendrait à mettre en rapport quelques MegaOctets de RAM et des pixels plus ou moins bien antialiassés. La définition d'un tel rapport *qualité/prix* n'a de sens que dans un contexte industriel, absolument pas dans le cadre universitaire où j'ai effectué ces travaux.

Chapitre 14

Résultats-Comparaisons-Discussion

14.1 Choix d'implémentation

Nous avons choisi d'implémenter un générateur de pixels utilisant la méthode de remplissage par tests d'inclusion. L'algorithme actuellement programmé (pour des raisons de simplicité) est un parcours de boite englobante, les performances pourraient être accrues sans rien changer au fonctionnement en utilisant l'algorithme de Pineda [Pineda88]. Le choix de cette méthode est essentiellement lié au calcul de la composante α au sommet des polygones. Il reste cependant des choix à faire quant à la taille et la forme du support de filtre.

La forme du support de filtre

Pour le calcul du masque de bits dans les méthodes à test d'inclusion basées sur les équations de droite, il faut interpréter la valeur de l'expression :

$$ax_p + by_p + c$$

qui est proportionnelle à la distance séparant la droite d'équation :

$$aX + bY + c = 0$$

au point de coordonnées (x_p, y_p). Pour cela, on normalise les coefficients de l'équation en les divisant par la norme du vecteur directeur :

$$A = \frac{a}{\sqrt{a^2 + b^2}} \qquad B = \frac{b}{\sqrt{a^2 + b^2}} \qquad C = \frac{c}{\sqrt{a^2 + b^2}}$$

Après cette normalisation, le résultat de l'expression correspond à la distance euclidienne entre le point et la droite. Cette façon de faire définit implicitement les pixels comme des disques : l'ensemble des droites situées à une distance donnée d'un point délimitent un cercle.

[Schilling91] propose de définir les pixels comme des carrés. Pour cela, il divise les coefficients de la droite non plus par la norme du vecteur directeur, mais par la *distance de Manhattan*, c'est-à-dire la somme des valeurs absolues des pentes en x et y :

$$A = \frac{a}{|a| + |b|} \qquad B = \frac{b}{|a| + |b|} \qquad C = \frac{c}{|a| + |b|}$$

L'ensemble des droites situées à une distance donnée d'un point délimitent maintenant un carré. Les préoccupations de Schilling sont essentiellement d'ordre matériel (la valeur absolue évite le calcul de la racine carrée), mais il précise que les résultats sont au moins aussi bon qu'avec la distance euclidienne. Remarquons que ces appréciations sont comme d'habitude subjectives et liées à une *sensation visuelle* plus qu'à une explication théorique.

La taille du support de filtre

Il s'agit maintenant de déterminer à partir de quelle distance une droite est censée traverser le pixel, ou en d'autres termes de déterminer la taille du support de filtre. Les coefficients ayant été normalisés, il faut définir un seuil, que nous notons s, délimitant le rayon du pixel. En un point donné, toutes les droites se trouvant à une distance inférieure à s sont considéreées comme traversant le pixel.

La forme du support de filtre que nous aurons choisi a évidemment une influence considérable sur le choix de s, observons les conséquences dans le cas du pixel rond (division des coefficients par la norme), et dans le cas du pixel carré (division par la distance de Manhattan).

- **pixel rond**
  Si on prend $s = 0.5$ quelle que soit la pente de la droite (ce qui simplifie énormément les calculs), certaines droites seront *oubliées* alors qu'elles traversent certains pixels. Plus grave encore, des objets fins pourraient passer à travers les pixels sans être détectés. Il est donc nécessaire de prendre un rayon d'au moins $\frac{\sqrt{2}}{2}$ afin de couvrir l'ensemble du plan de projection. Cette solution implique des recouvrements de pixels, non seulement cela n'est pas gênant, mais cela donne même de meilleurs résultats, en particulier en animation. Les transitions lors de déplacements d'objets sont plus fluides : un objet commence à être présent dans un pixel avant qu'il ait totalement quitté le pixel voisin.

- **pixel carré**
  Cette fois une distance de 0.5, quelle que soit la pente, suffira à couvrir la totalité de la surface des pixels. Cependant, pour les raisons que nous venons de spécifier, il est également plus judicieux d'augmenter légèrement la taille du filtre pour obtenir des pixels *recouvrants*.

D'une manière générale, un filtre large ($s \geq 0.7$) élimine plus de hautes fréquences et à pour conséquence de rendre l'image plus floue. Cette solution est très utile en animation où la netteté de l'image est moins importante que la fluidité des déplacements. Par contre, pour les images fixes, on préferera un filtre un peu moins large ($s \simeq 0.6$) pour obtenir une image plus nette.

Le filtre utilisé

Il reste maintenant à définir la forme du filtre passe-bas que nous allons utilisé. Pratiquement, cela revient à définir la fonction qui permet de transformer la distance entre le centre du pixel et la droite en une valeur α. La *figure 14.1* montre comment on calcule cette valeur à partir de l'aire couverte par la primitive dans le pixel dans le cas d'un support circulaire.

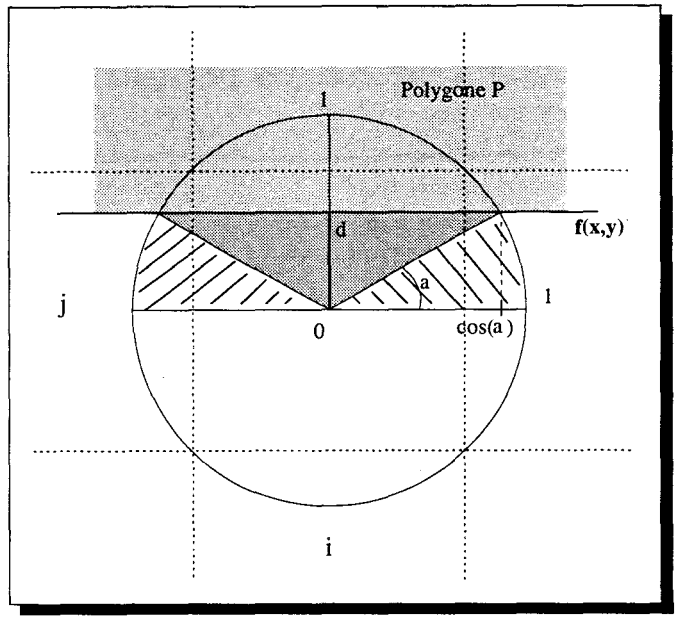


FIG. 14.1 - *Calcul de la couverture dans le cas d'un pixel rond*

Le polygone P couvre partiellement le pixel, la valeur $d = f(x_i, y_j)$ est négative car moins de la moitié du pixel est couvert. Soit $a = \arcsin(|d|)$, dans le demi-disque supérieur, la surface non couverte par P s'obtient en additionnant la surface hachurée que nous appelons S_1 :

$$S_1 = \frac{2a}{2\pi} \times \pi = a$$

et la surface coloriée que nous appelons S_2 :

$$S_2 = |d| \cos(a)\pi$$

ce qui nous donne pour ramener à une valeur entre 0 et 0.5 :

$$S = S_1 + S_2 = \frac{(|d| \cos a + a)0.5}{\pi/2} = \frac{(|d| \cos a + a)}{\pi}$$

Une table d'indirection stocke la valeur de cette fonction pour un certain nombre de valeurs discrètes de $|d|$ (16 dans notre implémentation). Il suffit donc pour obtenir la composante α d'aller lire dans la table la valeur correspondant à $|d|$ et de l'ajouter à 0.5 si $d \geq 0$ (P couvre plus de la moitié du pixel), ou de la retrancher si $d < 0$ (couverture inférieure à la moitié).

De nombreuses autres possibilités nous sont offertes pour le choix du filtre, dans le cas des pixels carrés par exemple, nous pourrions utiliser une fonction linéaire qui correspondrait à une *fonction porte*. Après de nombreux essais sur différentes scènes, nous avons opté pour la solution suivante :

- pixels carrés (division des coefficients par la distance de Manhattan),
- largeur du pixel 1.4 ($s = 0.7$),
- calcul de la composante α à partir du calcul de l'aire d'une portion de disque (comme décrit ci-dessus.

Il peut paraître étonnant de définir les pixels comme des carrés, et de calculer la composante α à partir d'un disque. Nous n'avons aucune explication théorique à ce sujet, c'est simplement la solution qui a paru *visuellement* la meilleure.

Notons que les enjeux de notre travail ne se situent pas du tout à ce niveau, en particulier les différences entre des pixels carrés ou ronds sont très peu sensibles en regard des approximations qui peuvent être faites par ailleurs.

14.2 Prise en compte des arêtes implicites?

Faut-il les prendre en compte les arêtes implicites, et à quel prix?
Les arêtes implicites sont, depuis le A-Buffer ou les premières méthodes de composition d'images, le point noir des algorithmes à échantillonnage surfacique. Alors que le sur-échantillonnage traite ces cas naturellement, les techniques de préfiltrage sont impuissantes à cause du Z unique.

Dans [Schilling et al.93], l'auteur décrit une méthode de détection et d'antialiassage par pré-filtrage des arêtes implicites. Cette technique est entièrement compatible avec notre algorithme. Les informations géométriques sont calculées comme pour les bords de polygones en une seule passe par l'intermédiaire de tables d'indirection. La contrepartie de cette amélioration n'est pas au niveau de la complexité algorithmique mais du surcoût en mémoire. Il faut garder en chaque pixel, la pente en x et y du plan support de la primitive stockée en mémoire. Cette méthode est décrite plus en détail au §9.4.1.

Notre position sur ce sujet est la suivante : nous pensons qu'un algorithme de rendu ne doit pas être indépendant du modeleur qui fournit les scènes et de l'algorithme qui facettise. De la même façon que l'antialiassage ne doit pas être vue comme une *verrue* sur un algorithme de rendu, mais comme une composante initiale impliquant des choix spécifiques, la modélisation et la facettisation ont des liens forts avec le rendu.

À cause des erreurs de calcul au moment de la transformation perspective des primitives, les algorithmes de rendu par tracé de contour et remplissage commettaient beaucoup d'erreurs au moment de l'affichage (par exemple il peut y avoir des trous entre des facettes jointives). Pour palier cet handicap, la modélisation est souvent réalisée en faisant *"déborder"* les facettes, c'est-à-dire en créant des arêtes implicites. Cette solution n'est pas du tout satisfaisante, nous pensons au contraire que les scènes doivent être modélisée avec beaucoup de précision pour obtenir un rendu de qualité.

14.3 Les architectures spécialisées

Actuellement de nombreux constructeurs de stations de travail proposent des cartes spécialisées pour l'affichage rapide de scènes 3D. Depuis longtemps, l'affichage des segments est câblé ou microprogrammé, et l'affichage *Gouraud/Z-Buffer* câblé commence lui aussi à être classique. Plus récemment, indépendemment de la compétitivité autour de la puissance d'affichage, sont apparues les machines proposant un l'antialiassage soit des segments (à l'aide des techniques de composition d'images) [Akeley et al.88], soit des bords de polygones (par sur-échantillonnage). Plus récemment encore, les interpolateurs pour le placage de texture ont également été réalisés matériellement, à la suite de quoi on trouve maintenant du préfiltrage (exclusivement par la méthode du *mip-mapping*) de texture.

14.3.1 Incompatibilité segments-facettes

Je me suis souvent demandé pourquoi les machines spécialisées proposaient soit un antialiassage de segments pour les représentations dans le modèle fil de fer, soit un antialiassage de bord de polygones, soit les deux mais pas en même temps, mais jamais la possibilité d'afficher simultanément (avec les afficheurs cablés bien sûr !) des facettes et des segments antialiassés. Il existe en fait une incompatibilité majeurs entre ces deux types de représentations. En effet, à ce jour les machines proposant un antialiassage de facette utilisent toutes le sur-échantillonnage (ponctuel) associé à un Z-Buffer. Or le Z-Buffer est incompatible, nous l'avons expliqué plus haut, avec la composante α des segments, par ailleurs sauf à les représenter comme des polygones à quatre cotés (ce qui rendrait leur affichage beaucoup plus lent) le sur-échantillonnage ne sait pas afficher de segments antialiassés. Nous assistons donc à un fractionnement des applications en deux mondes distincts : les scènes de C.A.O. en représentation fil de fer, et le rendu géométrique 3D avec éclairage de Gouraud.

14.3.2 Étude d'une machine haut de gamme: la *Reality Engine* 2

La *Reality Engine* 2 de chez *Silicon Graphics* est aujourd'hui ce qu'il existe de mieux sur le marché des stations de travail. Il nous a paru intéressant de faire le point sur ce que propose cette machine en terme d'antialiassage.

Remarquons pour l'anecdote que dans l'article [Akeley93] qui est une présentation de la machine, près du tiers du papier concerne l'antialiassage. En comparaison avec des publications plus anciennes sur les architectures spécialisées où le sujet (quand il était abordé) tenait sur une demi page, c'est dire si ce thème est devenu un enjeu prépondérant.

En fait, la *RE*2 implémente deux techniques d'antialiassage fondamentalement différentes. Une première approche basée sur un échantillonnage surfacique avec masque permet l'antialiassage de bords de polygones et de segments à condition que ceux-ci soient initialement triés en profondeur. C'est en fait une amélioration par les masques de sous-pixels de l'algorithme de composition d'images. Cette possibilité est bien adaptée aux applications 2D.

Pour les applications 3D, les bords de facettes sont antialiassés par un sur-échantillonnage ce qui rend l'affichage indépendant de l'ordre d'arrivée des polygones. Dans l'article, l'auteur prétend que ce sur-échantillonnage traite également les lignes mais nous ne voyons pas comment cela est possible.

Par ailleurs, le placage de texture est effectué à l'aide d'un *mip-mapping* trilinéaire. Or, nous avons vu ce que coutait cette technique en calculant une couleur de texture par pixel. Afin de ne pas multiplier encore ces accès à cause du sur-échantillonnage, le calcul de couleur (ou des coordonnées de texture) n'est effectué qu'une seule fois par pixel. Le mécanisme utilisé est intéressant :

- les objets sont convertis en pixels à la résolution de l'écran, avec un algorithme d'échantillonnage surfacique à masque (à partir d'un remplissage par test d'inclusion de type *Pineda*),
- une seule couleur (ou coordonnée de texture) est calculée pour le pixel,
- à l'aide de la méthode décrite dans [Schilling91] un masque de sous-pixels est généré, et grâce aux pentes dz_x et dz_y les vraies valeurs de profondeur sont recalculées en chaque sous-pixel,

- un Z-Buffer est appliqué sur chacun d'eux afin d'effectuer l'élimination des parties cachées au niveau sous-pixel comme un sur-échantillonnage classique.

On a donc ici une solution hybride où la mémoire de couleur et de profondeur est dupliquée : le Z-Buffer est exécuté au niveau sous-pixel sans que les performances soient ralenties, mais les interpolations de couleurs ou de coordonnées de texture se font à la résolution d'affichage.

Chapitre 15

Conclusion

L'aliassage est un phénomène intrinsèque à la synthèse d'images, inhérent à son processus de génération, qui nuit au réalisme des images s'il n'est pas traité. Présent dans de nombreux autres domaines, il est lié au processus discrétisation/reconstruction d'un signal continu. La théorie du signal nous permet d'analyser le phénomène et d'en comprendre les mécanismes. En ramenant ces résultats à la synthèse d'images, on s'aperçoit que parmi les différents *artefacts*, certains d'entre eux ne sont pas véritablement dûs à l'aliassage, c'est-à-dire à un repliement de spectre, mais à une mauvaise reconstruction qui entraîne un crénelage du signal reconstruit.

Dans tous les cas, le suréchantillonnage permet, sans résoudre le problème, de diminuer les effets d'aliassage. Cependant, pour que les défauts disparaissent, la résolution doit être considérablement augmentée, ce qui multiplie le temps de calcul des images. Sauf à utiliser des machines extrêmement puissantes, le suréchantillonnage reste aujourd'hui inaccessible au rendu rapide.

À cette technique de force brute, on préfère les méthodes de préfiltrage qui, s'inspirant des résultats établis en traitement de signal, éliminent les hautes fréquences du signal avant son échantillonnage. Ainsi, les moirés qui apparaissent sur les objets texturés peuvent être évités à l'aide de techniques de préfiltrage, plus ou moins sophistiquées.

Le problème des marches d'escalier sur les bords de polygones peut lui aussi se résoudre autrement que par le suréchantillonnage. Les techniques d'échantillonnage surfacique permettent de prendre en compte la participation d'un objet dans un pixel de manière analytique. Malheureusement, l'échantillonnage surfacique complique considérablement l'élimination des parties cachées. La prise en compte de plusieurs couleurs par pixel nécessite d'une part de connaître l'ordre des primitives en profondeur pour chaque pixel, d'autre part de disposer de quelque information géométrique sur la localisation dans le pixel, des fragments d'objets. En utilisant un tri en profondeur des primitives en chaque pixel, ainsi qu'un système de masque de sous-pixels, les algorithmes de type A-Buffer génèrent des images de très bonne qualité. Cependant, ce système basé sur des listes chainées d'objets, est lourd et assez complexe à gérer.

À l'opposé, les méthodes à voisins sont très simples à mettre en oeuvre et ne demandent que peu de mémoire, mais leur efficacité est beaucoup plus limitée. Nous avons proposé une méthode de ce type en cherchant à améliorer les techniques existantes. Néanmoins, nous pensons que ces méthodes ne sont pas assez robutes et qu'elles sont trop approximatives pour

être concurrentes face au suréchantillonnage sur des scènes complexes.

Par ailleurs, nous avons montré que de telles techniques pouvaient être réutilisées très efficacement pour traiter les problèmes d'aliassage en radiosité. Ces défauts, spécifiques aux méthodes de calcul des facteurs de forme par projection de la scène sur une surface discrétisée, présentent une certaine similitude avec le crénelage des bords de polygones. Nous avons donc adapté une méthode à voisin pour l'antialiassage des intensités de facettes dans le cadre des algorithmes de radiosité.

Dans le cadre du rendu rapide (rendu projectif, éclairage de Gouraud et Z-Buffer), nous avons voulu proposer une méthode à base d'échantillonnage surfacique, qui soit suffisamment simple pour une implémentation matérielle, afin d'en faire une alternative au suréchantillonnage. Nous avons donc proposé une approche mixte, qui garde la simplicité du Z-Buffer combinée à la précision d'un échantillonnage surfacique ; les primitives d'affichage sont traitées au vol sans tri préalable. L'idée centrale est de conserver un nombre limité de couleur par pixel. Pour cela, la notion de surface remplace celle de facette dans la mémoire d'images.

En stockant deux couleurs par pixel, les cas de polygones adjacents et de bords d'objets sur fond différent sont parfaitement traités. Les pixels plus complexes (mais aussi plus rares) dans lesquels plus de deux couleurs sont visibles sont approximés. Une autre version à trois couleurs a également été présentée, plus coûteuse (une mémoire d'image en plus), elle permet de traiter plus de cas.

Nous avons testé ces algorithmes sur des scènes complexes et très finement facettisées. Les résultats sont de très bonne qualité et peuvent être comparés à un suréchantillonnage 4×4. Cependant, notre méthode nécessite (comme le A-Buffer) de connaître, pour chaque facette de la scène, le numéro de surface à laquelle elle appartient afin de pouvoir effectuer l'opération de cumul. Sans cette information, les algorithmes deviennent fragiles et dépendants de paramètres très difficiles à évaluer. Nous pensons que cette contrainte est minime en regard des résultats obtenus.

À notre avis, une telle méthode mériterait d'être implémentée matériellement sous forme d'une carte accélératrice 3D. Elle permettrait l'affichage en temps réel de scènes antialiassées pour un coût qui serait de l'ordre de deux à trois fois celui des cartes actuelles. Celles-ci ne proposant l'antialiassage que sous la forme du suréchantillonnage c'est-à-dire en faisant chuter les performances.

Le choix entre la version à deux et celle à trois couleurs reste une question difficile. Cela revient à définir un rapport qualité/prix, à ce jour il manque une évaluation précise de la différence de coût (en Dollars) pour y répondre de manière pertinente.

Bibliographie

[Abram et al.85] Abram (Greg), Westover (Lee) et Whitted (Turner). – Efficient alias-free rendering using bit-masks and look-up tables. *Computer Graphics (SIGGRAPH '85 Proceedings)*, éd. par Barsky (B. A.), pp. 53–59. – July 1985.

[Akeley et al.88] Akeley (Kurt) et Jermoluk (Tom). – High-performance polygon rendering. *Computer Graphics (SIGGRAPH '88 Proceedings)*, pp. 239–246. – august 1988.

[Akeley93] Akeley (Kurt). – Realityengine graphics. *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 109–116. – 1993.

[Amanatides et al.90] Amanatides (John) et Mitchell (Don P.). – Antialiasing of interlaced video animation. *Computer Graphics (SIGGRAPH '90 Proceedings)*, pp. 77–85. – August 1990.

[Amanatides92] Amanatides (John). – Algorithms for the detection and elimination of specular aliasing. *Proceedings of Graphics Interface '92*, pp. 86–93. – May 1992.

[Beigbeder et al.86] Beigbeder (M.), Ghazanfarpour (D.) et Péroche (B.). – The "gz-buffer" method for antialiasing. *Deuxième colloque international de l'image électronique, CESTA*, pp. 817–822. – avril 1986.

[Carpenter84] Carpenter (Loren). – The A-buffer, an antialiased hidden surface method. *Computer Graphics (SIGGRAPH '84 Proceedings)*, éd. par Christiansen (Hank), pp. 103–108. – July 1984.

[Chaillou91] Chaillou (Christophe). – *Étude d'un processeur de visualisation d'images de synthèse en temps réel exploitant un parallélisme massif objet: le projet I.M.O.G.E.N.E.* – Thèse de PhD, Université des Sciences et Technologies de Lille, 1991.

[Cook et al.84] Cook (Robert L.), Porter (Thomas) et Carpenter (Loren). – Distributed ray tracing. *Computer Graphics (SIGGRAPH '84 Proceedings)*, pp. 137–145. – july 1984.

[Cook86] Cook (Robert L.). – Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, pp. 51–72. – january 1986.

[Crow82] Crow (Franklin C.). – Computational issues in rendering anti-aliased detail. *IEEE 1982 Spring COMPCON*, pp. 238–244. – 1982.

[Crow84] Crow (Franklin C.). – Summed-area tables for texture mapping. *Computer Graphics (SIGGRAPH '84 Proceedings)*, pp. 207–212. – july 1984.

[Deering et al.88] Deering (Michael), Winner (Stephanie), Schediwy (Bic), Duffy (Chris) et Hunt (Neil). – The triangle processor and normal vector shader : A vlsi system for high performance graphics. *Computer Graphics (SIGGRAPH '88 Proceedings)*, pp. 21–30. – August 1988.

[Dippe et al.85] Dippé (Mark A. Z.) et Wold (Erling Henry). – Antialiasing through stochastic sampling. *Computer Graphics (SIGGRAPH '85 Proceedings)*, éd. par Barsky (B. A.), pp. 69–78. – July 1985.

[Duff85] Duff (Tom). – Compositing 3-D rendered images. *Computer Graphics (SIGGRAPH '85 Proceedings)*, éd. par Barsky (B. A.), pp. 41–44. – July 1985.

[Dumas94] Dumas (C.). – *Antialiassage de traits et de surfaces en deux dimensions.* – Thèse, LIFL, Université des Sciences et Technologies de Lille, 1994.

[Fiume et al.83] Fiume (E.), Fournier (A.) et Rudolph (L.). – A parallel scan conversion algorithm with anti-aliasing for a general purpose ultracomputer. *Computer Graphics (SIGGRAPH '83 Proceedings)*, pp. 141–150. – July 1983.

[Foley et al.90] Foley (J.), Van Dam (A.), Feiner (S.) et Hughes (J.). – *Computer Graphics: Principes and Practice (second edition).* – The system programming series, Addison Wesley, 1990.

[Fuchs et al.85] Fuchs (Henry), Goldfeather (Jack), Hultquist (Jeff P.), Spach (Susan), Austin (John D.), brooks (Frederick P.), Eyles (John G.) et Poulton (John). – Fast spheres, shadows, textures, transparencies, and image enhancement in pixel-planes. *Computer Graphics (SIGGRAPH '85 Proceedings)*, pp. 111–120. – july 1985.

[Fujimoto et al.83] Fujimoto (A.) et Iwata (K.). – Jag-free images on raster displays. *IEEE Computer Graphics And Applications*, vol. 3, December 1983, pp. 26–34.

[Gangnet et al.84] Gangnet (Michel) et Ghazanfarpour (Djamchid). – Comparaison de techniques de mise en perspective de textures planes. *Actes du Premier Colloque International d'images électroniques, Biarritz, CESTA*, pp. 29–35. – mai 1984.

[Ghazanfarpour et al.87] Ghazanfarpour (Djamchid) et Péroche (Bernard). – A fast antialiasing method with a z-buffer. *Eurographics*, pp. 503–513. – 1987.

[Ghazanfarpour et al.89] Ghazanfarpour (Djamchid) et Péroche (Bernard). – Antialiasing by successive steps with a z-buffer. *Eurographics*, pp. 235–244. – 1989.

[Ghazanfarpour85] Ghazanfarpour (Djamchid). – *Synthèse d'Images et Antialiassage.* – Thèse de PhD, Ecole des Mines de Saint-Etienne, Novembre 1985.

[Ghazanfarpour92] Ghazanfarpour (Djamchid). – Visualisation réaliste par lancer de pyramides et subdivision adaptative. *Actes du Micad92*, pp. 167–180. – 1992.

[Glassner86] Glassner (Andrew). – Adaptive precision in texture mapping. *Computer Graphics (SIGGRAPH '86 Proceedings)*, pp. 297–306. – August 1986.

[Glassner89] Glassner (Andew S.). – *An introduction to RAY TRACING.* – Academic Press, 1989.

[Glassner95] Glassner (Andrew S.). – *Principles of Digital Image Synthesis.* – Morgan kaufmann Publishers, 1995.

[Gros91] Gros (Pascal). – *Etude et mise en oeuvre d'une architecture multiprocesseurs pour la synthèse d'images.* – Thèse de PhD, Université des Technologies de Compiègne, octobre 1991.

[Gupta et al.81] Gupta (S.) et Sproull (R. F.). – Filtering edges for gray-scale displays. *Computer Graphics (SIGGRAPH '81 Proceedings)*, pp. 1–5. – August 1981.

[Haeberli et al.90] Haeberli (Paul) et Akeley (Kurt). – The accumulation buffer : Hardware support for hight-quality rendering. *Computer Graphics (SIGGRAPH '90 Proceedings)*, pp. 309–318. – August 1990.

[Heckbert et al.91] Heckbert (Paul S.) et Moreton (Henry P.). – *Interpolation for Polygon Texture Mapping and Shading.* – Springer-Verlag, 1991volume in State of the art in computer graphics: Visualization and Modeling.

[Kajiya86] Kajiya (James T.). – The rendering equation. *Computer Graphics (SIGGRAPH '86 Proceedings)*, pp. 143–150. – August 1986.

[Lau et al.95] Lau ("Wing Hung) et neil Wiseman". – "the compositing buffer : a flexible method for image generation and image editing". *"Computer Graphics Forum"*, vol. 14, n° 4, 1995, pp. 229–238.

[Lee et al.85] Lee (Mark E.), Redner (Richard A.) et Uselton (Samuel P.). – Statistically optimized sampling for distributed ray tracing. *Computer Graphics (SIGGRAPH '85 Proceedings)*, éd. par Barsky (B. A.), pp. 61–67. – July 1985.

[Lindgren et al.93] Lindgren (Terence) et Weber (John). – Measuring the quality of anti-aliased line drawing algorithms. *Fourth Eurographics Workshop*

on Rendering, éd. par Cohen (Michael F.), Puech (Claude) et Sillion (Francois). Eurographics, pp. 157–174. – June 1993. held in Paris, France, 14–16 June 1993.

[Lindgren90] Lindgren (Terence). – Anti-aliased curves. *Ausgraph'90.* – 1990.

[Mammen89] Mammen (Abraham). – Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Computer Graphics and Applications*, pp. 43–55. – july 1989.

[Mccool et al.92] McCool (Michael) et Fiume (Eugene). – Hierarchical poisson disk sampling distributions. *Proceedings of Graphics Interface '92*, pp. 94–105. – May 1992.

[Meyer90] Meyer" ("Urs). – "hemicube ray tracing: a method for generating soft shadows". *Eurographics '90.* pp. "365–376". – Elsevier Science Publisher, 1990.

[Mitchell et al.88] Mitchell (Don P.) et Netravali (Arun N.). – Reconstruction filters in computer graphics. *Computer Graphics (SIGGRAPH '88 Proceedings)*, pp. 221–228. – august 1988.

[Mitchell87] Mitchell (Don P.). – Generating antialiased images at low sampling densities. *Computer Graphics (SIGGRAPH '87 Proceedings)*, éd. par Stone (Maureen C.), pp. 65–72. – July 1987.

[Molnar91] Molnar (Steven Edward). – *Image-composition Architectures for real-Time Image Generation.* – Thèse de PhD, University of North Carolina at Chapel Hill, 1991.

[Pineda88] Pineda (Juan). – A parallel algorithm for polygon rasterization. *Computer Graphics (SIGGRAPH '88 Proceedings)*, pp. 17–20. – August 1988.

[Pitteway et al.80] Pitteway (M.L.V.) et Watkinson (D.J.). – Bresenham's algorithm with grey scale. *ACM*, vol. 23, n° 11, November 1980, pp. 625–626.

[Platel95] Platel (Fabrice). – *Antialiassage de bords de polygones pour le rendu et la radiosité sur architecture SIMD.* – Thèse, Université des Sciences et Technologie de Lille, 1995.

[Porter et al.84] Porter (Thomas) et Duff (Tom). – Compositing digital images. *Computer Graphics (SIGGRAPH '84 Proceedings)*, pp. 253–259. – july 1984.

[Potmesil et al.83] Potmesil (Michael) et Chakravarty (Indranil). – Modelling motion blur in computer generated images. *Computer Graphics (SIGGRAPH '83 Proceedings)*, pp. 389–399. – july 1983.

[Preux et al.92] Preux (Alain) et Chaillou (Christophe). – Un algorithme économique pour l'antialiassage des bords de polygones. *Actes des journées GROPLAN 1992*, pp. 23–30. – Nantes, Novembre 1992.

[Preux et al.94] Preux (Alain) et Chaillou (Christophe). – Une extension du z-buffer pour l'antialiassage des bords de polygones. *Revue internationale de CFAO et d'infographie, Actes de MICAD 1994*, pp. 235–249. – 1994.

[Renaud93] Renaud (Christophe). – *Approches parallèles pour la radiosité.* – Thèse de PhD, Université des Sciences et Technologies de Lille, 1993.

[Ris95] Ris (Philippe). – *Lancer de rayon et parallélisme.* – Thèse, Université de Franche Conté Besançon, 1995.

[Schilling et al.93] Schilling (Andreas) et Straßer (Wolfgang). – Exact : Algorithm and hardware architecture for an improved a-buffer. *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 85–91. – August 1993.

[Schilling91] Schilling (Andreas). – A new simple and efficient anti-aliasing with subpixel masks. *Computer Graphics (SIGGRAPH '91 Proceedings)*, éd. par Sederberg (Thomas W.), pp. 133–141. – July 1991.

[Shinya93] Shinya (Mikio). – Spatial anti-aliasing for animation sequences with spatio-temporal filtering. *Computer Graphics (SIGGRAPH '93 Proceedings)*, éd. par Kajiya (James T.), pp. 289–296. – August 1993.

[Swanson et al.86] Swanson (Roger W.) et Thayer (Larry J.). – A fast shaded-polygon renderer. *Computer Graphics (SIGGRAPH '86 Proceedings)*, pp. 95–101. – August 1986.

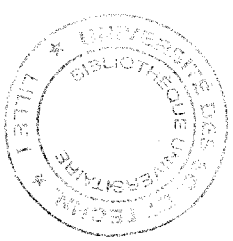
[Weinberg81] Weinberg (Richard). – Parallel processing image synthesis and antialiasing. *Computer Graphics (SIGGRAPH '81 Proceedings)*, pp. 55–62. – August 1981.

[Whitted80] Whitted" ("T.). – "an improved illumination model for shaded display ". *Communication of the ACM*, vol. 23, n° 6, "1980", pp. 342–349.

[Williams83] Williams (Lance). – Pyramidal parametrics. *Computer Graphics (SIGGRAPH '83 Proceedings)*, pp. 1–11. – July 1983.

[Wu91] Wu (Xiaolin). – An efficient anti-aliasing technique. *Computer Graphics (SIGGRAPH '91 Proceedings)*, éd. par Sederberg (Thomas W.), pp. 143–152. – July 1991.

[Yellot83] Yellot (J. I.). – Spectral consequences of photoreceptor sampling in the rhesus retina. *Science*, n° 221, 1983, pp. 382–385.


UNIVERSITE DE LILLE I
SCIENCES ET TECHNIQUES
BIBLIOTHEQUE UNIVERSITAIRE