ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre : 14 ID

THESE

présentée par

Michel BEIGBEDER

pour obtenir le titre de

DOCTEUR

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

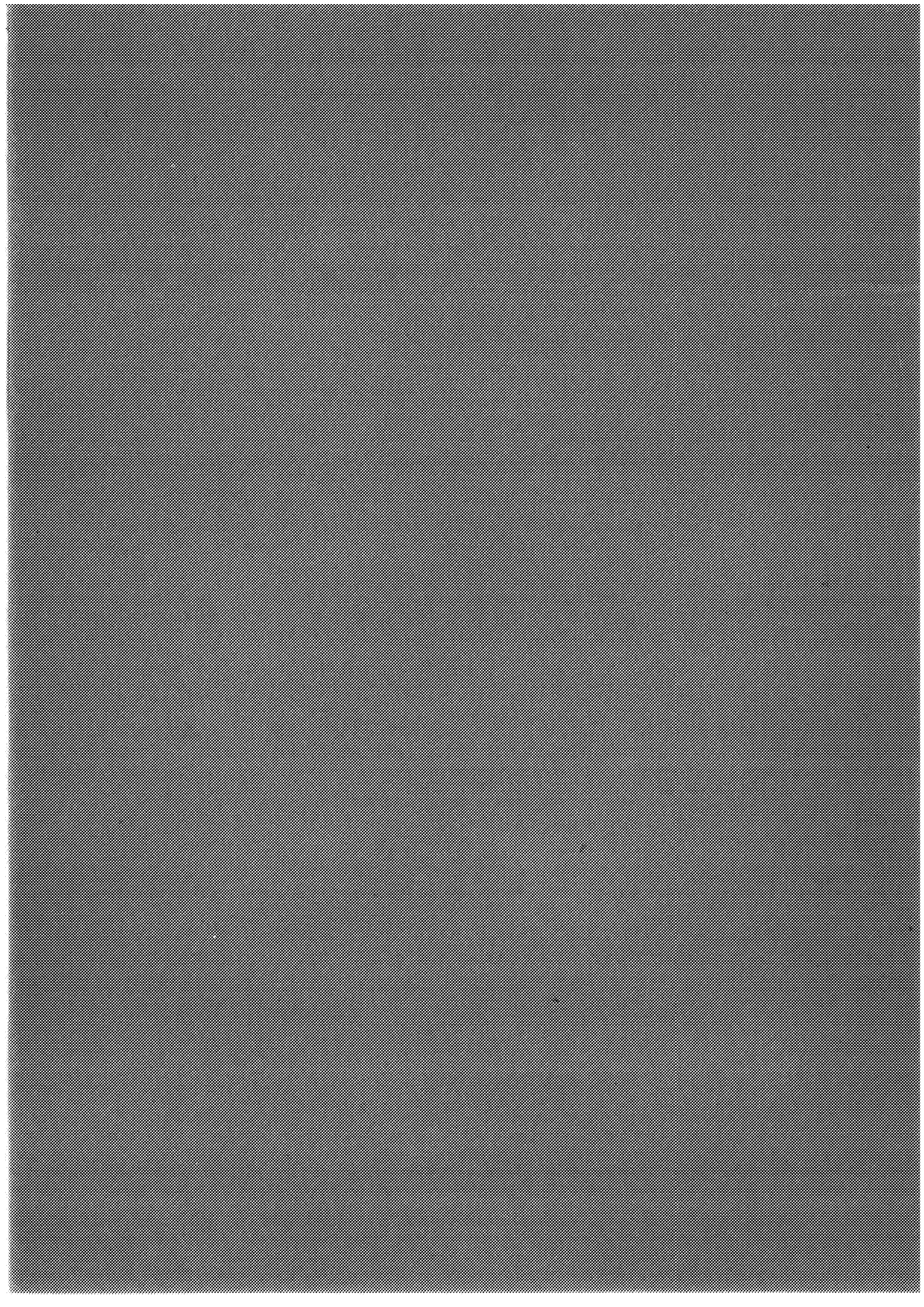
(Spécialité : Informatique, Image, Intelligence Artificielle et Algorithmique)

UN DEVELOPPEMENT POUR LA MODELISATION ET LA VISUALISATION EN SYNTHESE D'IMAGES : CASTOR

Soutenue à Saint-Etienne le 25 Avril 1988

COMPOSITION DU JURY

Monsieur Y. GARDAN	Président et rapporteur
Monsieur D. LAURENT	Rapporteur
Messieurs J. AZEMA M. GANGNET A. NICOLAS B. PEROCHE D. PLAY	Examinateurs



ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre : 14 ID

THESE

présentée par

Michel BEIGBEDER

pour obtenir le titre de

DOCTEUR

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Informatique, Image, Intelligence Artificielle et Algorithmique)

UN DEVELOPPEMENT POUR LA MODELISATION ET LA VISUALISATION EN SYNTHESE D'IMAGES : CASTOR

Soutenue à Saint-Etienne le 25 Avril 1988

COMPOSITION DU JURY

Monsieur Y. GARDAN	Président et rapporteur
Monsieur D. LAURENT	Rapporteur
Messieurs J. AZEMA M. GANGNET A. NICOLAS B. PEROCHE D. PLAY	Examinateurs

REMERCIEMENTS

J'exprime mes plus vifs remerciements à tous les membres du jury :

Monsieur Yvon GARDAN, professeur à l'Université de METZ, rapporteur, qui me fait l'honneur de présider le jury ;

Monsieur Daniel LAURENT, professeur à l'Université de Paris VII, rapporteur ;

Monsieur Jean AZEMA, maître de conférences à l'Université de SAINT-ETIENNE ;

Monsieur Michel GANGNET, chercheur au Laboratoire de Recherche Parisien de Digital;

Monsieur Alain NICOLAS, directeur du département *Recherche et Développements* de la société Thomson Digital Image,

Monsieur Bernard PEROCHE, directeur du Département Informatique Appliquée de l'Ecole des Mines de SAINT-ETIENNE, qui m'a toujours fait confiance et qui a consacré beaucoup de temps à encadrer mes travaux ;

Monsieur PLAY, professeur à l'Institut National des Sciences Appliquées de LYON.

Je remercie les utilisateurs des logiciels que j'ai produits pour leur obstination et tout particulièrement Monsieur Frank CHOPIN (architecte DPLG) sans qui je n'aurai pu réaliser les photographies présentées dans ce rapport.

Je remercie les membres du Département Informatique, et tout particulièrement :

Monsieur Jean-Pierre TAVERNIER, qui assure la maintenance du matériel informatique que j'ai utilisé, et qui réussit à trouver quelques octets sur un disque qui déborde périodiquement ;

Monsieur Jean-Jacques GIRARDOT pour son logiciel de formatage de texte : *groff*, qui a assuré la mise page de ce rapport ;

Monsieur Paul-André Pays pour quelques aides très précieuses dans la manipulation de *groff*,

REMERCIEMENTS

tous les membres, passés ou présents, de l'équipe *Communications Visuelles* : Jacqueline ARGENCE, Sabine COQUILLART, Gilles FERTEY, Djamchid GHAZANFARPOUR, Ghassan JAHMI, Dominique MICHELUCCI, Jean-Michel MOREAU, Zhigang NIE, Didier TALLOT.

Je remercie enfin les membres du service de reprographie de l'Ecole des Mines de Saint-Etienne.

INTRODUCTION 5

MODELISATION SOLIDE TRIDIMENSIONNELLE 7

Introduction 7
Définitions 8
Schéma de représentation des solides 17
Instanciation de primitives 17
Enumération spatiale 17
Géométrie solide constructive (Constructive Solid Geometry : CSG) 18
Principe de la représentation CSG 19
Définition de la représentation CSG 19
Variantes et propriétés de la représentation CSG 20
Représentation par extrusion 22
Représentation par frontières. 24
Les croisements de représentation 27
Conclusion 28
Conversion entre les représentations 29
Facettisation 35
Les informations non géométriques 37
Les informations indépendantes de la géométrie 37
Les informations dépendant de la géométrie 39
Les langages de saisie 39
PICTUREBALM, GRAMPS, QUADRIL, PADL 39
PICTUREBALM 40
GRAMPS 40
QUADRIL 41
PADL-1 et PADL-2 41
Conclusion 42
CASTOR 42
Les raisons d'un choix 42
La représentation interne 43
La syntaxe 45
La sémantique 49
L'implantation et l'environnement 49
Exemples d'applications 51
Limitations et extensions possibles du langage 52

MODELISATION DEDIEE *53*

Introduction 53
Terrains 53
Représentation au moyen d'une grille régulière 53
Représentation par mosaïque de triangles 55
Végétation 56
Modèles basés sur les textures 56
Définition des textures 56
Définition des zones d'applications 57
Arbres définis grâce à des textures 57
Modèles procéduraux 58
Les méthodes existantes 58
Notre implantation 62

VISUALISATION *71*

Introduction 71
Les morphologies 72
L'élimination des parties cachées 73
Le cas des polygones 74
Les différents algorithmes 75
Essai de classification 82
Conclusion 87
L'approche inverse : le lancer de rayons 88
Généralisation des algorithmes aux surfaces 88
Généralisation des algorithmes aux mélanges de surfaces 94
Composition par approximation 94
Composition par liste de priorité 97
Visualisation d'arbres CSG 97
Evaluation en dimension un 98
Evaluation en dimension deux 100
Evaluation en dimension trois 100
Le rendu 100
Les éléments nécessaires aux calculs du rendu 100
Le processus de rendu par rapport aux processus de composition 101
L'antialiassage 102
L'algorithme du "gz-buffer" 110
Présentation 110
L'algorithme 111
Le calcul de gzp[.,.] et de gp[.,.] 120
Les défauts de la méthode du gz-buffer 123
Comparaison avec les autres méthodes 126

Le cas des terrains 127
Implantation actuelle : UNIX 128
Présentation générale 128
Le format de communication de polygones 129
Composition : l'extension de l'implantation, développements futurs 130
La composition au niveau du pixel 130
Le rendu 131

PHOTOGRAPHIES *133*

REFERENCES BIBLIOGRAPHIQUES *147*

Chapitre 1

INTRODUCTION

La synthèse d'images doit sa naissance à la première de ses applications : les simulateurs de conduite. Si les premiers de ces systèmes ne permettaient que d'obtenir des représentations de scènes nocturnes où n'intervenaient que des sources lumineuses ponctuelles, l'accroissement des performances des matériels informatiques et l'évolution des algorithmes ont considérablement amélioré les possibilités des systèmes récents. Toutefois l'ampleur des marchés concernés par ces systèmes est telle que les détails d'implantation et les algorithmes utilisés ne sont pas publiés.

Les travaux présentés dans ce rapport sont issus de problèmes posés par la représentation de paysages, rejoignant donc ceux qui se sont posés aux concepteurs des systèmes de simulation de conduite. La plupart des travaux précédemment publiés ne prennent en compte que le terrain lui-même [FISH 80], [MAX 81], [ANDE 82], [COQU 84a] ; c'est-à-dire qu'ils ne considèrent pas les problèmes posés par la définition et la visualisation du *sur-sol*. Ce dernier terme recouvre à la fois les éléments naturels tels que la végétation, les cours d'eau,... et les éléments construits par l'activité humaine tels que les maisons, les routes,...

Nous avons donc été conduits à étudier les moyens de représenter toute cette diversité d'éléments. Nous nous sommes particulièrement attachés aux techniques de modélisation de solides tridimensionnels pour la représentation des maisons ou autres constructions volumiques. Cette étude est présentée dans le deuxième chapitre de ce rapport. Nous nous sommes inspirés des travaux sur la modélisation de solides en CAO (conception assistée par ordinateur) mais nous avons pris en compte les besoins spécifiques de la synthèse d'images et particuliers par rapport à ceux de la CAO. Cela nous a conduit à définir un langage de modélisation basé sur la représentation par arbre de construction et à réaliser un interpréteur de ce langage : *castor*.

Pour ce qui est des éléments naturels, nous avons fait un tour d'horizon des techniques de modélisation et/ou de visualisation (ces deux phases étant parfois étroitement imbriquées) pour les terrains et la végétation. Ce tour d'horizon est présenté dans le chapitre trois.

Une grande diversité de modélisation s'avère donc indispensable pour représenter les différents éléments apparaissant dans un paysage. De ce fait, un problème se pose pour la visualisation : cette grande diversité de modélisation entraîne-t-elle une aussi grande diversité de visualisation ? Plus précisément, chaque modélisation va-t-elle nécessiter son propre algorithme de visualisation ou bien un seul sera-t'il suffisant ? Lorsque l'on sait qu'un tour d'horizon récent [WILL 85] ne cite pas moins de 60 références d'articles décrivant des méthodes d'élimination des parties

cachées, on comprend qu'il faut disposer d'un moyen pour classer ces algorithmes, bien sûr selon les éléments qu'ils traitent, mais aussi selon la façon dont ils procèdent pour évaluer la compatibilité qu'ils peuvent présenter entre eux. Après une étude rapide des différents algorithmes, nous proposons, dans le chapitre quatre, une notation permettant de décrire certaines caractéristiques de ceux-ci, en particulier celles qui sont utiles pour déterminer la compatibilité de plusieurs algorithmes. Nous évoquons aussi dans ce chapitre les liens entre les algorithmes d'élimination des parties cachées et les problèmes - plus récents que certaines des publications originales ou qui ne sont pas toujours traités explicitement ni d'une façon homogène - que la synthèse d'images s'efforce de résoudre : le rendu et l'antialiassage. En particulier une méthode d'antialiassage adaptée à l'algorithme du tampon de profondeur est présentée et comparée aux autres méthodes traitant du même problème.

Les questions liées à la notion de *composition* [MART 84] posent le problème de la réalisation et de la maintenance d'un système de synthèse. Ce problème déjà abordé dans [WHIT 81] et [CROW 82] est tout à fait d'actualité et a fait l'objet de trois publications très récentes [NADA 87], [POTM 87] et [COOK 87].

Pour notre part, nous proposons une approche fortement inspirée et associée au système d'exploitation UNIX. Cette approche privilégie donc une structure très modulaire avec des données qui circulent dans des tubes ou qui sont stockées sur des fichiers. Dans cette idée, nous avons implanté trois programmes qui permettent d'avoir un ensemble cohérent de modélisation et de représentation :

- l'interpréteur *castor* cité *supra* transforme les arbres de constructions qu'il reçoit sur son entrée standard en listes de polygones ;
- un autre filtre *wn* assure la transformation perspective d'une liste de polygones et réalise le fenêtrage de ces polygones par rapport au cône de vision ;
- enfin un dernier programme *zb* transforme une liste de polygones dans un format utilisable par notre matériel d'affichage (cf. annexe) qui utilise la méthode du tampon de profondeur pour déterminer les parties visibles, et il permet donc d'obtenir une image.

Ces réalisations ont suscités d'autres développements cités dans le quatrième chapitre. D'autre part, le choix de se baser sur UNIX a permis d'autres réalisations non prévues d'origine ; ces réalisations prouvent donc la souplesse du système. Ce sont :

- l'adaptation d'une technique de modélisation d'arbre, cette technique est présentée dans le troisième chapitre et le même principe serait utilisable pour toute modélisation procédurale ;
- la réalisation d'animation, dont le principe est présenté au chapitre deux.

Chapitre 2

MODELISATION SOLIDE TRIDIMENSIONNELLE

2.1 INTRODUCTION

Nous allons étudier dans ce chapitre les problèmes (et quelques solutions) que posent la description et la visualisation des objets du sur-sol. Par ces derniers termes, nous désignons tout ce qui est fabriqué par l'activité humaine et qui est visible dans un paysage :

- maison d'habitation
- mur de clôture
- pont
- poteau
- ...

Il s'agit donc d'objets à décrire géométriquement en trois dimensions. Cette activité de description n'est pas nouvelle en informatique puisqu'elle est apparue avec les logiciels de CAO en mécanique, logiciels dont le but était et reste toujours de fournir à un ingénieur un ensemble d'outils devant l'aider à concevoir des pièces mécaniques.

Nous insistons ici sur le caractère tridimensionnel des objets modélisés et représentés. Cela signifie que ces objets possèdent un intérieur non vide au sens de la topologie usuelle de l'espace euclidien de dimension 3. Cela ne signifie pas que ces objets ne peuvent pas être représentés par un ensemble cohérent de surfaces ; par contre ces objets ne peuvent être réduits à l'une de ces surfaces seulement.

Nous allons donc reprendre ici certaines des idées développées pour les applications de CAO ; toutefois, nous les compléterons à deux points de vue : l'un étroitement lié à la synthèse d'images puisqu'il s'agit de la visualisation et l'autre plus général : la saisie de la modélisation par un opérateur. Ce dernier point nous conduira à étudier plusieurs langages de description et en particulier nous décrirons la solution que nous avons apportée au problème de représentation et de description de données tridimensionnelles : le langage **CASTOR**.

2.2 DEFINITIONS

Nous nous plaçons dans ce paragraphe à un niveau formel et nous écartons toute particularité issue de l'implantation sur une machine plutôt que sur une autre.

Nous allons définir les mots que nous emploierons tout au long de ce chapitre en précisant le sens que nous leur donnerons :

- modélisation
- représentation
- description
- schéma de représentation

Le langage que nous utilisons tous les jours (en ce qui nous concerne : le français) nous permet d'associer un sens à certains assemblages (des mots) de symboles (des lettres) respectant certaines règles syntaxiques et sémantiques. Prenons un exemple en nous plaçant à un niveau très bas où nous ne considérerons que le sens que l'on peut attribuer à un mot isolé.

L'ensemble **E** de tous les assemblages possibles de lettres peut être construit par les règles syntaxiques suivantes :

1. x
2. $\mathbf{M}x$

pour toute lettre x de l'alphabet et pour tout assemblage **M** de **E**. Parmi les assemblages éléments de **E**, seuls certains possèdent un sens. Appelons **V** l'ensemble des assemblages de **E** qui ont une signification ; on définit ainsi une surjection de l'espace réel dans l'espace **V**.

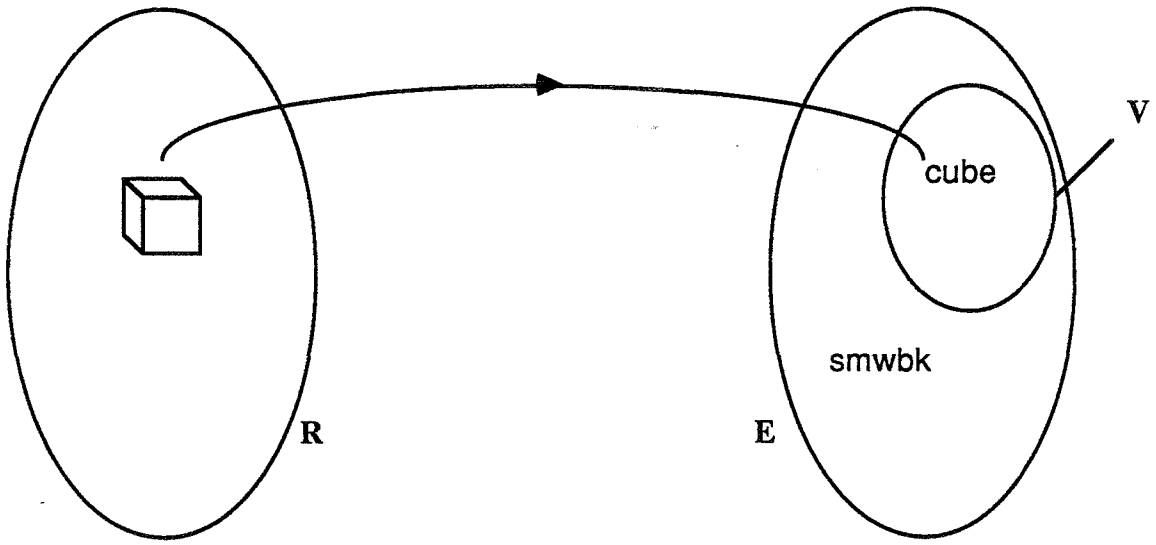


figure 2.1

Ce même schéma de relation va se retrouver pour les termes de modélisation et de représentation.

Pour obtenir des définitions plus précises que celles suggérées par l'exemple précédent, nous allons devoir nous placer dans un espace abstrait et écarter l'espace réel : plutôt que de parler d'un cube dont chaque arête mesure deux unités, nous parlerons de :

$\{(x,y,z) \in \mathbf{R}^3 ; \sup(|x| , |y| , |z|) \leq 1\}$.

Nous nous plaçons donc dans un *espace de modélisation* qui est l'espace affine euclidien de dimension trois. Comme nous voulons modéliser dans cet espace les solides du monde réel, nous devons traduire les propriétés de ces solides en propriétés mathématiques. Nous appelerons *solide abstrait* tout sous-ensemble de $\mathbf{R}^3$ susceptible de modéliser un solide réel ; plus précisemment nous reprenons ici la liste donnée par [REQU 80] des propriétés que doit posséder une partie de $\mathbf{R}^3$ pour être un *solide abstrait* :

Rigidité	La forme d'un *solide abstrait* doit être indépendante de sa position et de son orientation.
Tridimensionnalité	Un *solide abstrait* doit avoir un intérieur et sa frontière ne doit pas contenir d'arêtes ou de faces pendantes.
Finitude	Un *solide abstrait* ne doit occuper qu'une partie bornée de l'espace euclidien $\mathbf{R}^3$.

Fermeture pour certaines opérations booléennes

L'application d'opérations qui ajoutent ou retirent du matériau doivent être des lois de composition internes dans l'ensemble des *solides abstraits*.

Finitude d'une description

Les *solides abstraits* doivent avoir d'un certain point de vue un aspect fini (par exemple un nombre fini de *faces*) de façon à pouvoir être représentés en machine.

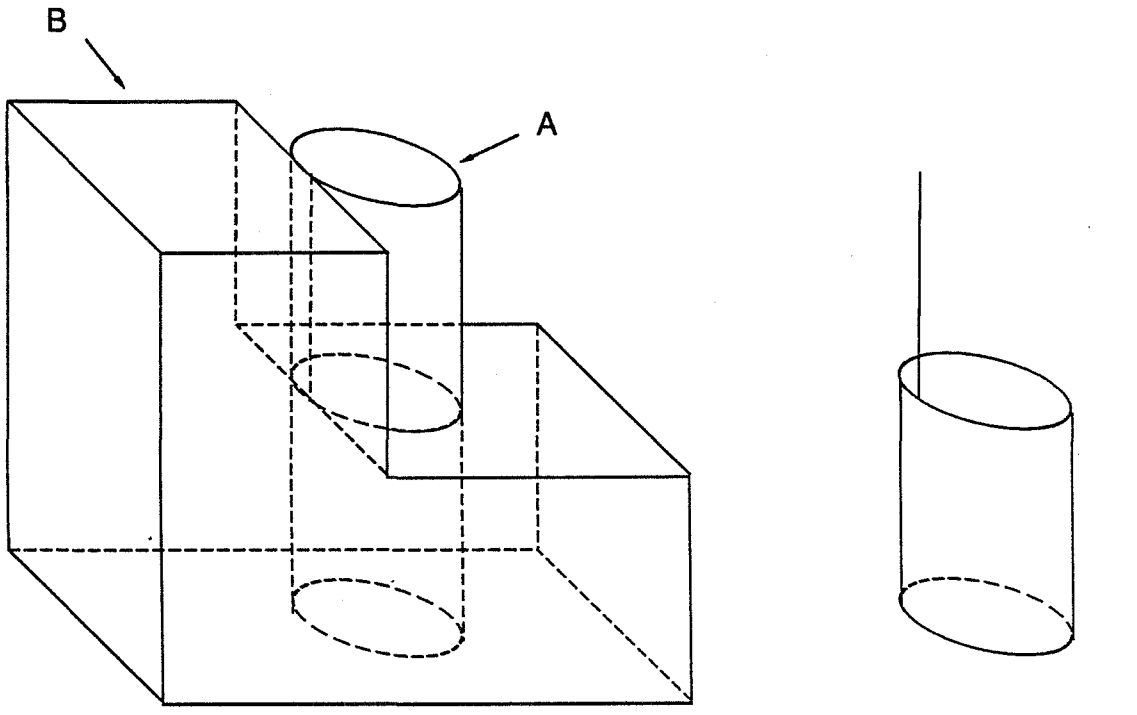
Déterminisme des frontières

La frontière d'un *solide abstrait* doit définir de façon non ambiguë ce qui est à l'*intérieur* du solide.

Ces conditions impliquent [REQU 77] que les modèles convenables de solides sont les sous-ensembles de l'espace euclidien de dimension trois qui sont bornés, fermés, réguliers et semi-analytiques. C'est désormais le sens que nous attribuerons à l'expression *solide abstrait*, et nous appelerons **S** l'ensemble des *solides abstraits* ; c'est un sous-ensemble de $\mathbf{P}(\mathbf{R}^3)$.

Remarque

Les opérations booléennes usuelles ne sont pas des lois de composition internes dans **S** comme on peut s'en rendre compte avec l'exemple suivant :



Objets A et B intersection de A et B

figure 2.2

Cela conduit à introduire de nouvelles opérations booléennes, les opérations booléennes régulières définies dans [TILO 80] de la façon suivante :

- $X \cap^* Y = r(X \cap Y)$
- $X \cup^* Y = r(X \cup Y)$
- $X \setminus^* Y = r(X \setminus Y)$
- $c^*(X) = r(c(X))$

où r désigne l'application qui à un sous-ensemble de $\mathbf{R}^3$ associe l'adhérence de son intérieur, et c désigne l'application qui à un sous-ensemble associe son complémentaire. $\cap^*$, $\cup^*$ et $\setminus^*$ sont des lois de composition internes dans S. Par contre, c^* n'est pas une application dans S : en effet, si X est borné son complémentaire ne l'est pas.

La représentation a pour objectif de pouvoir associer des solides abstraits à des assemblages de symboles en machine. Les assemblages de symboles sont construits par l'usage de règles syntaxiques. L'ensemble de tous les assemblages syntaxiquement corrects est l'*espace de représentation* **R**. Cet espace est un langage engendré par une grammaire. Toutefois, nous ne limitons pas ces assemblages à des chaînes de caractères ; ce peuvent être des arbres ou des graphes : nous verrons d'ailleurs des exemples où c'est le cas.

La sémantique de ce langage est assurée par une relation entre l'*espace de modélisation* et l'*espace de représentation.* C'est cette relation que nous appelons *schéma de représentation* **s**.

Ayant défini cette relation **s**, nous pouvons utiliser les notions mathématiques et définir **D** comme le domaine de **s** et **V** comme l'image de **s**.

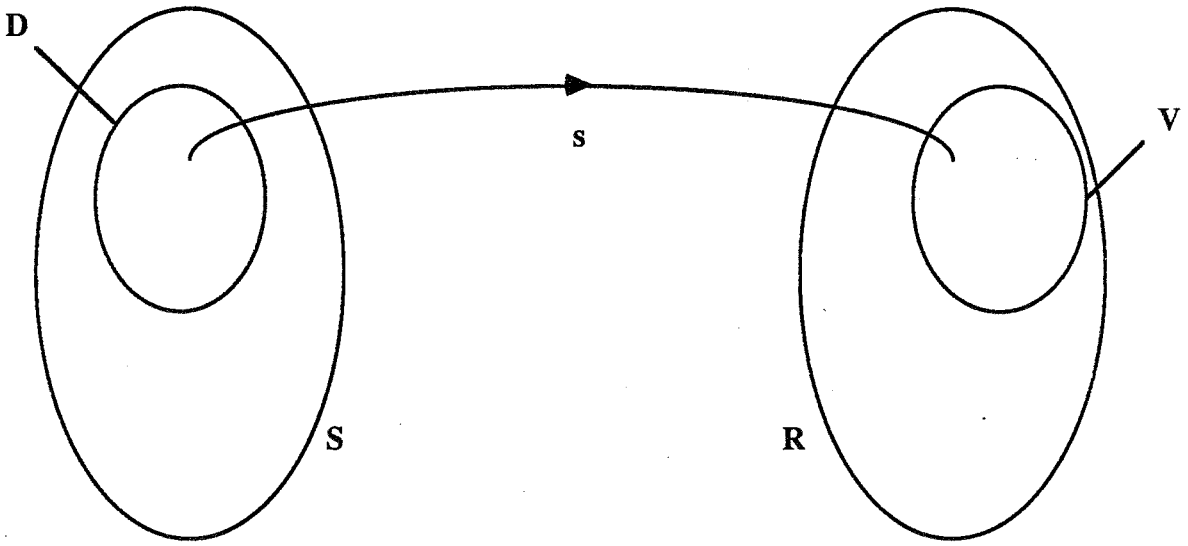


figure 2.3

V est l'ensemble des représentations syntaxiquement correctes qui sont images d'éléments de **D**. Le plus souvent, **V** est égal à **R** pour des grammaires raisonnables (par exemple : indépendantes du contexte). Ceci est d'ailleurs fortement souhaitable : en effet, il est peu recommandé d'utiliser un algorithme, géométrique par exemple, sur une représentation qui n'a pas de sens : on ne pourra obtenir que des résultats sans valeur et, dans le pire des cas, des résultats sans valeur et à l'apparence correcte ! Dans le cas où **V** est égal à **R**, l'existence du sens d'une représentation peut être assurée au niveau syntaxique. L'implantation de la vérification de la cohérence sémantique se limite à écrire un analyseur syntaxique, problème bien résolu.

D est l'ensemble des solides qui sont représentables par le schéma de représentation **s**. Plus l'ensemble **D** est grand, plus le pouvoir de représentation du schéma est grand.

Une représentation **r** élément de **V** est *non ambiguë* si elle est en relation avec un seul élément de **D**. Le schéma de représentation **s** est *non ambigu* si tout élément de **V** est non ambigu ; ce qui revient à dire que **s** est non ambigu si et seulement si la relation réciproque s^{-1} est une fonction.

Une représentation **r** élément de **V** est *unique* si l'objet correspondant de **D** n'admet pas d'autre représentation que **r** (ie : $s(s^{-1}(r)) = \{r\}$). Le schéma de représentation **s** est *unique* si toute représentation syntaxiquement correcte est unique ; ce qui revient à dire que **s** est unique si et seulement si **s** est une fonction.

Un schéma de représentation unique et non ambigu réalise une bijection entre **V** et **D**.

Les propriétés que nous venons de citer correspondent à des définitions mathématiques bien précises. Il y a une autre catégorie de propriétés qui se révèlent toutes aussi importantes au niveau d'une application mais qui sont beaucoup plus difficiles à formaliser. Nous allons les citer ici :

Concision	Il s'agit ici d'une mesure de la place qu'occupe une représentation en machine. Une grande concision s'accompagne souvent d'une non-redondance des informations et donc d'une facilité pour le contrôle de la validité. Par contre, davantage de redondance peut permettre d'accélérer des algorithmes en conservant en mémoire certaines données plutôt qu'en les recalculant.
Facilité de création	Point très important pour l'utilisateur de l'application, la facilité de création va souvent de pair avec la concision : celle-ci implique qu'il y a moins d'informations à spécifier et que les informations à fournir sont dans une large mesure indépendantes. Les schémas qui nécessitent des représentations redondantes et verbeuses nécessitent pour le remplissage des structures de représentation des systèmes d'entrée puissants. Cette façon de procéder consiste en fait à effectuer un changement de représentation, problème que nous évoquerons en 2.4.
Efficacité	Cette qualité est complètement dépendante de l'application et plus précisemment dans une même application, dépendante des algorithmes à utiliser. Pour ce qui concerne la synthèse d'images, ce ne sont pas les mêmes structures que l'on va pouvoir utiliser pour l'algorithme du *lancer de rayon* ou pour celui du *z-buffer* par exemple.

Les problèmes de la qualité des algorithmes (exactitude, efficacité, stabilité (ie : résistance aux erreurs de calculs),...) viennent donc interférer avec les problèmes liés à la représentation elle-même.

En conclusion, un système de modélisation qui doit avoir une gamme d'utilisation raisonnable devra le plus souvent avoir des représentations multiples ; cela pose le problème de la cohérence entre les différentes représentations lors de l'implantation d'une part, et les problèmes de *consistance* et d'*équivalence* de deux schémas de représentation au niveau formel d'autre part.

Considérons deux schémas de représentation **s** et **s'** sur le même espace de modélisation **M** (la figure 2.4 illustre notre propos). On dit qu'une représentation **r** de **R** et une représentation **r'** de **R'** sont consistantes si elles ont un antécédent commun.

Cependant, dans un cas général, **s**, s^{-1}, **s'** et s'^{-1} ne sont ni des bijections ni même des fonctions. Ceci ne permet donc pas de définir une bijection entre les représentations valides de **s** et celles de **s'**. Par exemple, si les schémas ne sont pas non ambigus, on peut avoir les éléments $\mathbf{m}_1$ et $\mathbf{m}_2$ de **S** en relation respectivement avec **r** et **r'** :

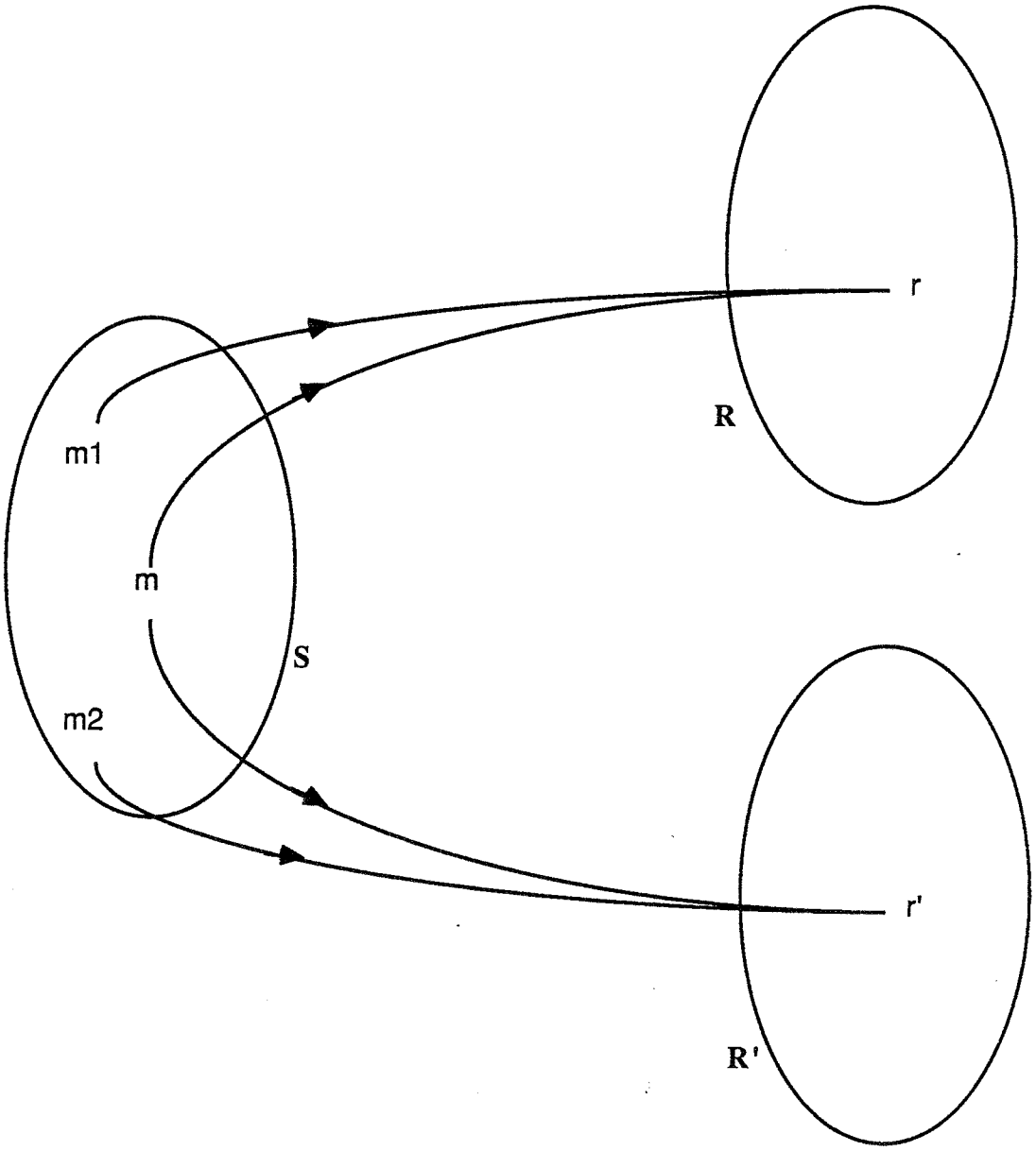


figure 2.4

et si les schémas ne sont pas uniques, d'autres représentations $\mathbf{r}_1$ et $\mathbf{r}_1$' peuvent représenter **m** respectivement par **s** et **s'** :

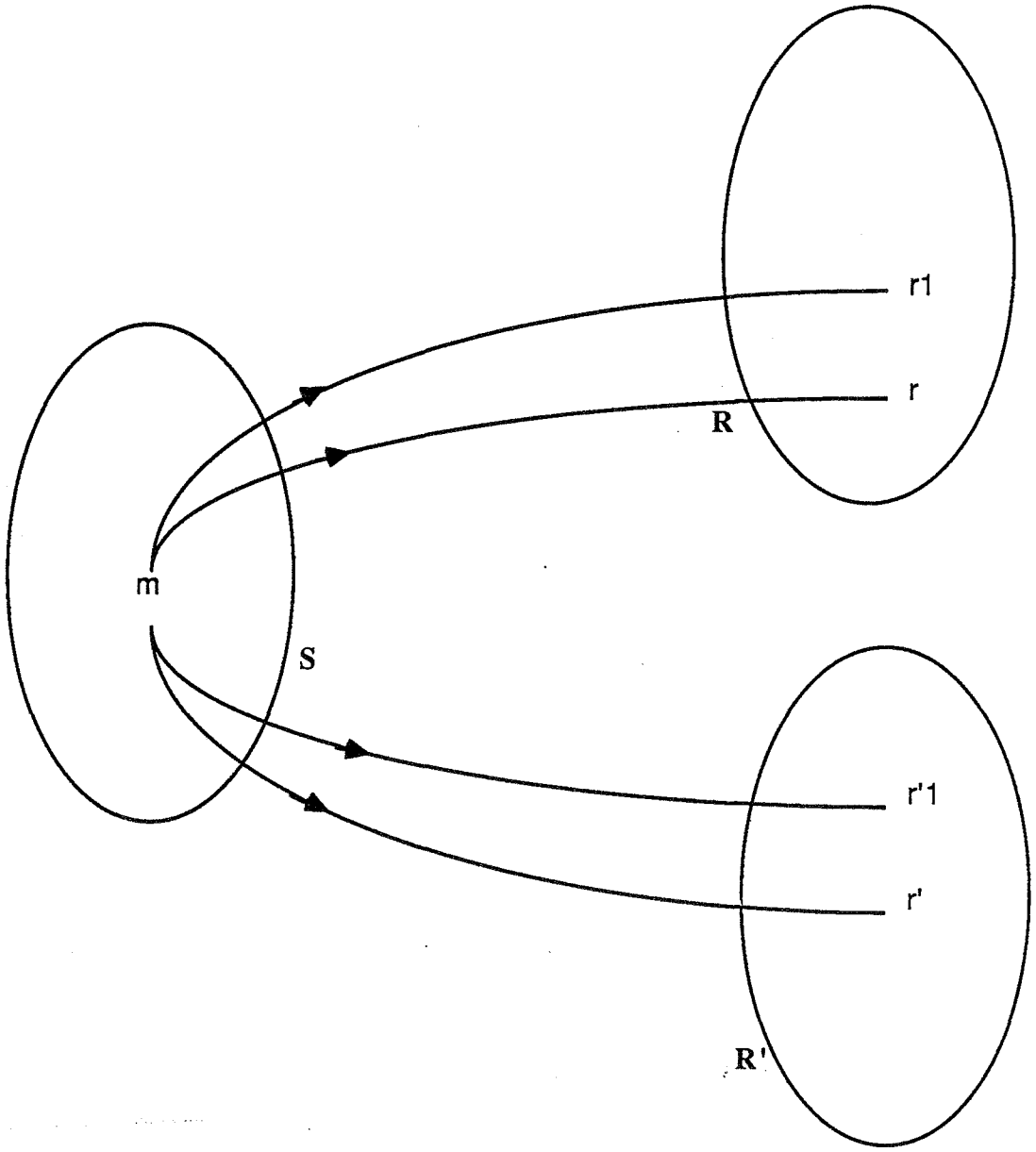


figure 2.5

Dans le deuxième cas, $\mathbf{r}_1$ et **r** sont consistants. Mais dans le premier cas cette notion de consistance semble insuffisante ; aussi on dira que les deux représentations **r** et **r'** sont équivalentes si et seulement si elles représentent le même ensemble d'objets :

$$s^{-1}(r) = s'^{-1}(r')$$

et on étendra la définition aux schémas de représentation : les schémas de représentation **s** et **s'** sont équivalents si et seulement si toute représentation **r** de **s** possède un équivalent **r'** dans **s'** et réciproquement.

De cette définition il résulte que deux schémas équivalents ont le même domaine. Réciproquement si deux schémas non ambigus ont le même domaine, alors ils sont équivalents.

La cohérence dont nous avons parlée pour l'implantation lorsque plusieurs représentations doivent cohabiter nécessite donc l'équivalence des schémas de représentations, et cette cohérence signifie qu'à tout instant sont présentes en machine les différentes représentations consistantes de tous les objets.

2.3 SCHEMA DE REPRESENTATION DES SOLIDES

Dans ce paragraphe, nous allons étudier différents schémas de représentation du point de vue formel que nous avons défini en 2.2, du point de vue algorithmique pour la synthèse d'images et enfin les interfaces utilisateur directes que l'on peut envisager pour ces schémas.

2.3.1 Instanciation de primitives

Le domaine des schémas de ce type est limité à quelques familles d'objets, les objets étant indexés à l'intérieur d'une famille par un petit nombre de paramètres. Un objet est donc identifié par un n-uplet de taille fixe qui d'une part indique à quelle famille appartient l'objet et d'autre part donne la liste des paramètres permettant de le sélectionner. Par exemple, ("CUBE",3) permet de désigner l'élément de taille trois unités dans la famille des cubes.

Par leur principe, ces schémas sont uniques, non ambigus et triviaux à valider. Pour la CAO, ces schémas posent des problèmes pour les algorithmes de calcul sur les solides représentés. En effet, une grande partie des connaissances spécifiques à une famille doit être intégrée dans le codage des algorithmes, ce qui restreint l'extensibilité de ces schémas. Le deuxième inconvénient de ces schémas est leur incapacité à structurer un ensemble d'objets élémentaires en un objet complexe de plus haut niveau. De ce fait, l'utilisation d'un tel schéma est très limitée pour l'interaction. Ce schéma peut par contre se justifier comme le dérivé d'un autre schéma et utilisé au moment de la visualisation : la plupart des machines de synthèse d'images sont actuellement capables d'afficher des listes de polygones bi- ou tridimensionnels ; on peut supposer que la génération suivante sera capable de traiter des listes d'objets primitifs.

2.3.2 Enumération spatiale

Les images que nous manipulons découpent l'espace bidimensionnel de l'écran en un pavage de carrés élémentaires : les pixels. D'une façon analogue, en trois dimensions, on peut considérer qu'un cube est constitué d'une multitude de cubes élémentaires : des voxels (*volume element*) ou obels (*object element*). Une représentation d'un solide sous forme d'énumération spatiale consiste à avoir la liste des cellules qui sont occupées par le solide. Pour déterminer si une cellule est occupée par le solide ou non, on se base en fait sur un point particulier de cette dernière (par

exemple son centre) qui sert aussi à repérer et à identifier la cellule.

Variante de ce schéma, on peut garder une information plus riche que celle de savoir si le voxel est occupé ou non en gardant l'information du matériau qui occupe le voxel. Cette information supplémentaire peut aussi être une information de couleur.

A la taille du voxel près un schéma de représentation par énumération spatiale est non ambigu et a la propriété d'unicité. Son domaine a une richesse inversement proportionnelle à la taille des voxels manipulés.

La façon la plus simple de représenter un solide par cette méthode est de conserver une matrice à trois dimensions. Cette structure est nommé PEARY (*Picture Element ARraY*) [OHAS 85]. La taille de la mémoire nécessaire pour le stockage d'une scène est très grand et ceci d'autant plus que l'on veut conserver beaucoup d'informations pour chaque voxel. Une taille aussi grande a conduit certains chercheurs à évaluer des solutions matérielles aux problèmes de visualisation : [KAUF 85], [OHAS 85]. A titre d'exemple, [KAUF 85] envisage une matrice 512x512x512 avec huit bits de profondeur ce qui représente 128 méga-octets !

Lorsque l'objet représenté occupe une grande région d'espace, de nombreux éléments adjacents de la matrice ont la même valeur : ceci suggère qu'une forte compression des données est possible [DOCT 81]. Le codage de la matrice sous forme d'octree réalise une compression du fait que des éléments voisins de la matrice ayant la même valeur vont pouvoir être regroupés. Ce codage est l'analogue en trois dimensions du quadtree en deux dimensions. L'information contenue dans la matrice est codée dans un arbre. Si un cube est homogène, l'information décrivant son contenu est conservée et on est sur une feuille de l'arbre ; dans le cas contraire, ce cube est découpé en huit octants - eux-mêmes des cubes - et on est sur un nœud de l'arbre qui a alors huit fils pour lesquels on recommence.

[MEAG 82] présente en détail ce mode de représentation et propose des algorithmes pour les opérations booléennes, les transformations géométriques ainsi que pour la visualisation avec élimination des parties cachées.

Au niveau interaction, il est évidemment hors de question de saisir les données directement dans ce schéma de représentation. L'absence de structure sous la forme tableau à trois dimensions ou son inconsistance avec la signification (ie : l'objet) sous la forme octree ne permettent que de donner une liste de codes à l'aspect rebutant pour un être humain.

2.3.3 Géométrie solide constructive (Constructive Solid Geometry : CSG)

Nous arrivons ici à une des représentations les plus utilisées dans les systèmes commerciaux pour la modélisation d'objets tridimensionnels. C'est ce modèle que nous avons également choisi d'implanter ; aussi, pour ces deux raisons, nous allons le détailler d'avantage.

Nous avons traduit mot à mot les termes de l'expression anglaise *Constructive Solid Geometry* pour le titre de ce paragraphe. On trouve aussi ce schéma de représentation sous le nom d'*arbre de représentation* ou encore d'*arbre*

de contruction dans la littérature française. mais ce sont les initiales de l'expression anglaise qui sont le plus souvent employées : CSG. Nous utiliserons désormais ce sigle.

2.3.3.1 Principe de la représentation CSG

Le principe du CSG consiste à combiner des solides au moyen d'opérations ensemblistes ; ce principe est récursif, c'est-à-dire que de nouvelles opérations peuvent intervenir sur des objets précédemment créés. La structure naturellement sous-jacente est donc un arbre dont les nœuds sont des opérations booléennes et les feuilles sont des objets primitifs. Enfin des transformations géométriques peuvent intervenir pour modifier les éléments du sous-arbre auquel elles sont associées. De façon à ne pas faillir à la tradition, nous donnerons l'exemple maintenant célèbre de la figure 2.6.

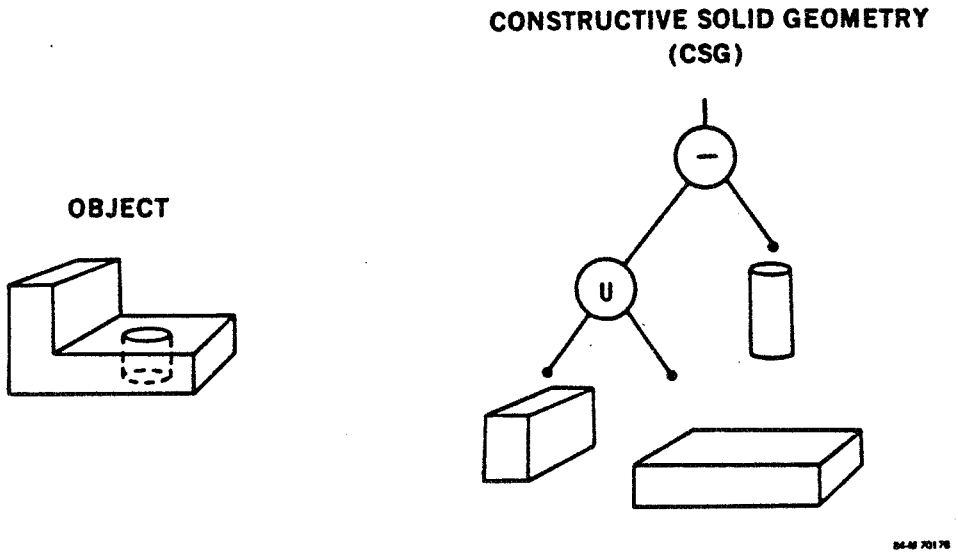


figure 2.6

2.3.3.2 Définition de la représentation CSG

Le modèle CSG repose sur des bases mathématiques qui sont claires, bien connues et aisées à formaliser :

- les opérations booléennes régulières que nous avons évoquées en 2.2 : $\cap^*, \cup^*, \setminus^*$,
- l'ensemble **T** des transformations géométriques affines,
- l'ensemble **P** des primitives qui se trouvent sur les feuilles de l'arbre.

Avec un arbre de profondeur 1, c'est-à-dire qui ne comporte qu'une feuille l'ensemble des objets représentables est :

$$R_1 = \bigcup_{f \in T, S \in P} \{ f(S) \}$$

Dans un arbre de profondeur 2 on peut combiner avec une des trois opérations $\cap^*$, $\cup^*$ ou $\setminus^*$ des éléments de $\mathbf{R}_1$, ce qui permet donc de représenter :

$$R_2 = \bigcup_{S \in R(1), S' \in R(1)} \{ S \cap^* S' \} \cup \bigcup_{S \in R(1), S' \in R(1)} \{ S \cup^* S' \} \cup \bigcup_{S \in R(1), S' \in R(1)} \{ S \setminus^* S' \}$$

et récursivement avec un arbre de profondeur n, on peut représenter tous les éléments de $\mathbf{R}_n$ avec la définition récurrente suivante :

$$R_n = \bigcup_{S \in R(n-1), S' \in R(n-1)} \{ S \cap^* S' \} \cup \bigcup_{S \in R(n-1), S' \in R(n-1)} \{ S \cup^* S' \} \cup \bigcup_{S \in R(n-1), S' \in R(n-1)} \{ S \setminus^* S' \}$$

En définitive, l'ensemble des objets représentables est

$$R = \bigcup_{n \in \mathbf{N}} R_n$$

Ces définitions mettent en évidence le fait que la validité d'une représentation peut être assurée au niveau syntaxique, ceci à condition que l'on utilise comme primitives un sous-ensemble de l'ensemble des solides abstraits que nous avons définis en 2.2.

2.3.3.3 Variantes et propriétés de la représentation CSG

Suivant les implantations de ce modèle, les sous-arbres peuvent être partagés ou non. Dans le deuxième cas, la structure est véritablement un arbre alors que dans le premier cas nous avons affaire à un graphe sans circuit.

Les définitions que nous avons données ne font apparaître les transformations géométriques qu'au niveau de $\mathbf{R}_1$. Ceci ne restreint aucunement le pouvoir de représentation ; en effet, pour toute transformation affine f, on a :

- $f(A \cap B) = f(A) \cap f(B)$
- $f(A \cup B) = f(A) \cup f(B)$
- $f(A \setminus B) = f(A) \setminus f(B)$

Cependant, ramener toutes les transformations au niveau des primitives n'est compatible qu'avec la structure d'arbre et pas avec la structure de graphe que nous venons de citer lorsque des sous-arbres peuvent être partagés.

Un autre facteur de différence notable entre les systèmes est le type des objets primitifs connus. D'une part, plus le nombre d'objets primitifs connus est important, et plus ceux-ci sont variés, plus le pouvoir de représentation sera important. Par exemple, un premier niveau d'un schéma de représentation CSG pourrait ne contenir qu'un objet primitif : le cube. Grâce par ailleurs à l'usage de transformations géométriques et d'opérations booléennes l'ensemble des objets représentables serait l'ensemble des polyèdres bornés de $\mathbf{R}^3$. Un deuxième niveau pourrait intégrer en plus la sphère comme objet primitif,...

La différence fondamentale, du point de vue théorique, est la présence ou non de primitives non bornées. Dans le cas où des primitives non bornées sont utilisées, la validation d'une représentation n'est pas nécessairement assurée par la validation syntaxique puisque des arbres représentants des objets non bornés peuvent être générés. D'après [REQU 80], la seule méthode pour vérifier cette validité est de transformer la représentation CSG en une représentation par frontières puis de tester si les frontières sont bornées ; ce qui est algorithmiquement très coûteux. Une autre façon d'assurer la validité des représentations CSG avec primitives non bornées est de les obtenir par conversion d'après des représentations avec primitives bornées. Cela peut se faire par exemple en remplaçant dans les feuilles de l'arbre les primitives par l'intersection des demi-espaces qui les définissent (eux-mêmes étant représentés par des inéquations).

Nous avons donné les définitions des ensembles des objets représentables dans le cas où l'arbre CSG est un arbre binaire et où les opérations booléennes sont également binaires. L'associativité des opérations booléennes $\cup$ et $\cap$ permet aussi bien d'autoriser que les arbres que l'on construit ne soient pas binaires que de transformer ces derniers en arbres binaires. L'ensemble des objets représentables globalement est donc le même suivant qu'on utilise des arbres binaires ou non. Cependant pour un niveau de profondeur dans les arbres fixé, l'ensemble des objets représentables par un arbre binaire est plus petit que l'ensemble des objets représentables par un arbre quelconque : la représentation par un arbre binaire est moins concise que l'autre.

Pour que les définitions des ensembles $\mathbf{R}_n$ précédentes collent à la réalité de l'implantation il faut que dans cette dernière les opérations booléennes soient bien les opérations booléennes régularisées et qu'elles s'appliquent d'une façon générale et sans restriction à tous les éléments de ces ensembles $\mathbf{R}_n$. Dans le cas contraire les propriétés de validité des arbres CSG sont analogues à celles des schémas de représentation par décomposition en cellules. Cette dernière remarque de [REQU 80] nous incite à faire une différence entre le niveau de *représentation des objets* et le niveau de l'*interprétation de la représentation*. Si, formellement, on peut très bien considérer deux représentations dont la première n'utilise que des opérations booléennes classiques et la deuxième les opérations booléennes régularisées, à quel niveau se situe la distinction entre les deux ? La première représentation (en reprenant l'exemple de la figure 2.2) notera :

$A \cap B$

et la deuxième :

$A \cap^* B$.

Ce n'est que dans l'interprétation de ces deux expressions que la différence va apparaître, puisque la représentation de $\cap$ et de $\cap^*$ peut être la même. Par exemple si l'interprétation de cette expression est une visualisation, va-t'on ou ne va-t'on pas afficher le segment

$$(A \cap B) \setminus (A \cap^* B) ?$$

On voit une deuxième fois que les problèmes algorithmiques sont étroitement imbriqués avec les problèmes de représentation.

Certaines formes dégradées de représentations CSG ne possèdent qu'un seul opérateur : le *collage*. C'est la forme restreinte de l'opérateur *union* qui ne s'applique qu'aux objets d'intérieurs disjoints.

Le pouvoir de représentation du schéma CSG est directement lié au nombre et à la qualité des primitives admises. C'est un schéma non ambigu, par contre il y a plusieurs façons de représenter le même objet : ce schéma n'est pas unique.

Ce type de représentation se prête bien à une interaction avec l'utilisateur au moyen d'un langage. Nous en verrons des exemples dans un prochain paragraphe. De plus, un ensemble de primitives adaptées aux objets à représenter permet d'avoir des représentations concises et de ce fait faciles à manipuler par un utilisateur.

Pour ce qui est de la visualisation, de nombreuses méthodes ont récemment été étudiées. La méthode la plus classique de visualisation pour les arbres CSG est le *lancer de rayons* [ROTH 82]. [ATHE 83] propose, lui, une extension des algorithmes à *balayage par ligne* pour ce schéma de représentation. [JANS 85] combine une méthode de visualisation par listes de priorité avec une structuration (EXCELL [MANT 83]) permettant d'effectuer un tri indépendant du point de vue. Cette même structure lui permet d'optimiser un algorithme de lancer de rayons. Enfin [OKIN 84], [ROSS 86], et [HOOK 86] sont des extensions de la méthode du *z-buffer* adaptées à ce schéma.

2.3.4 Représentation par extrusion

Il est très facile de comprendre le principe de cette représentation. Un ensemble de points en se déplaçant balaye un volume que l'on peut représenter par "l'objet" qui se déplace et par sa "trajectoire". Plus formellement, si

$$\begin{aligned} \phi : [0,1] &\rightarrow \mathrm{Is}(\mathbf{R}^3) \\ \lambda &\rightarrow \phi(\lambda) \end{aligned}$$

est une application continue du segment [0,1] dans l'ensemble des isométries de $\mathbf{R}^3$, on définit le solide extrudé par ϕ à partir de l'ensemble S par :

$$\bigcup_{\lambda \in [0,1]} \phi(\lambda)(S)$$

Cependant, on ne connaît pas de conditions nécessaires et suffisantes sur S et ϕ pour pouvoir assurer la validité de ce schéma.

Deux types particuliers de fonctions ϕ sont plus particulièrement utilisées. Premièrement si nous appelons $tr_{\vec{u}}$ la translation de vecteur **u**, et **k** le vecteur (0,0,1) de $\mathbf{R}^3$, nous pouvons définir la fonction ϕ_h indexée par le réel h, (ce réel h ayant une signification de hauteur), par :

$$\begin{aligned}\phi_h : [0,1] &\rightarrow Is(\mathbf{R}^3)\\ \lambda &\rightarrow tr_{\lambda.h\mathbf{k}}\end{aligned}$$

et on choisit un ensemble S inclus dans un plan orthogonal à **k** et d'intérieur non vide pour la topologie du plan.

Par exemple, l'extrusion du carré [0,1]x[0,1] par le vecteur **k** donne le cube [0,1]x[0,1]x[0,1] pour une hauteur h d'une unité.

Deuxièmement en appelant r_θ la rotation d'axe **k** et d'angle θ, nous pouvons définir la fonction ψ_θ indexée par l'angle θ, par :

$$\begin{aligned}\psi_\theta : [0,1] &\rightarrow Is(\mathbf{R}^3)\\ \lambda &\rightarrow r_{\lambda.\theta}\end{aligned}$$

et on choisit un ensemble S inclus dans un plan contenant **k** et d'intérieur non vide pour la topologie du plan.

Par exemple, l'extrusion du disque D de centre (0.5,0,0) et de rayon 0.5 par la rotation d'un tour complet donne un tore.

Ces deux schémas sont non ambigus mais non uniques. Leur domaine de représentation est très réduit : si leur utilisation peut être suffisante pour un système à usage restreint comme par exemple un logiciel de CFAO pour une machine de tournage, leur utilisation dans un système général doit plutôt être vue comme un complément destiné à être transformé dans un autre mode de représentation.

Citons un avantage important de ce schéma : les éléments à représenter sont d'une part un nombre réel (hauteur ou angle) et d'autre part une surface plane. De ce fait une bonne interaction peut être obtenue avec l'utilisateur puisqu'on peut cette fois entrer la donnée géométrique (la surface plane) sans difficulté avec un écran et une souris ou une tablette à numériser.

La visualisation d'objets représentés par ce schéma se fait en convertissant d'abord cette représentation en représentation par frontières, ce qui ne pose en général pas de difficultés au niveau formel puisque les courbes servant à la génération de la surface sont disponibles ; cependant, pratiquement, il faut encore que les équations obtenues à partir de ces courbes soient utilisables par un algorithme de visualisation.

2.3.5 Représentation par frontières

Dans ce schéma un solide n'est plus représenté que par sa frontière. Celle-ci est partitionnée en un nombre fini de sous-ensembles appelés "faces" ou "patchs". Cependant même dans le cas simple de représentation de polyèdres il existe des schémas où la notion intuitive de *face* ne correspond pas aux faces qui vont être représentées : pour s'en convaincre il suffit de considérer un schéma où ne sont admis que des faces triangulaires et de représenter par ce schéma une pyramide à base carrée... Cette base qui est intuitivement une face de la pyramide devra être décomposée en triangles qui seront les *faces* de la représentation.

Pour les objets courbes, la notion intuitive de *face* disparaît (cf. figure 2.7).

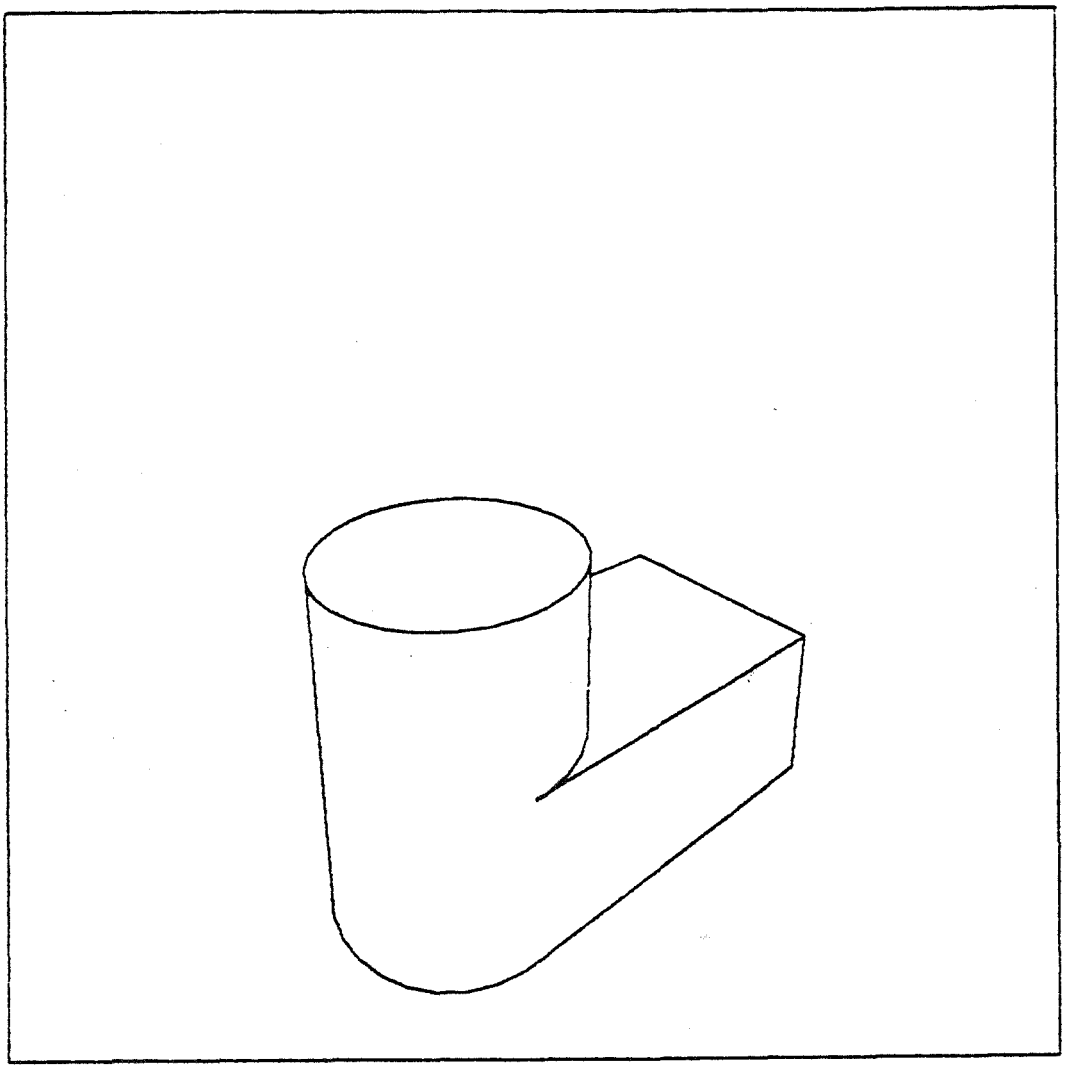


figure 2.7

De la même façon que nous avons donné les caractéristiques qui nous ont permis de définir la notion de *solide abstrait*, la notion de *face* doit être clarifiée. [REQU 80] donne des détails et des références sur ce sujet. Nous devrons pour notre part retenir que pour avoir un *schéma de représentation* non ambigu, la représentation des surfaces doit elle-même être non ambiguë.

Sans entrer dans des détails sur la représentation des surfaces, on peut cependant donner quelques indications. Trois types semblent pouvoir être dégagés, suivant le *degré* des surfaces supportant les frontières. *Degré* signifie ici le degré d'un polynome équation de la surface dans un repère cartésien. Nous avons donc :

premier degré

Les surfaces support sont donc contenues dans des plans, de ce fait ce sont des polygones. Plusieurs méthodes existent pour représenter un volume par un ensemble de polygones et pour représenter les polygones eux-mêmes. Pour ce qui est des polygones, on peut soit donner les listes des coordonnées des sommets de chaque arête de chaque polygone, soit donner des listes de pointeur sur les sommets précédemment définis pour chaque arête ; de toutes façons, dans ce type de schéma, les polygones sont représentés sous forme explicite. De même, pour le solide on peut avoir plus ou moins de structuration dans la liste de polygones.

Le domaine de représentation d'un schéma de représentation par frontières dont les frontières sont contenues dans des plans est l'ensemble des polyèdres bornés.

[BAUM 75] décrit très précisemment une implantation de ce schéma. Le sujet est très classique et on peut consulter [FOLE 82] pour des détails et d'autres références.

second degré

Dans ce schéma de représentation, les surfaces sont contenues dans des quadriques et limitées par des demi-espaces dont la frontière est également une quadrique. Le sujet a aussi été plusieurs fois abordé, citons [MAHL 72], [LEVI 76].

La méthode usuelle pour représenter un morceau de surface supporté par une quadrique est de conserver les coefficients de l'équation de la quadrique supportant la surface et ceux des inéquations des frontières des demi-espaces limitant le morceau. De cette façon, la surface est définie de façon implicite. Toutefois, [LEVI 76] présente une méthode permettant d'obtenir sous forme paramétrique la courbe intersection de deux surfaces quadriques. On peut ainsi obtenir une représentation explicite de cette courbe, frontière du

morceau de surface défini sous forme implicite.

La représentation citée ci-dessus définit les solides comme des intersections de demi-espaces limités par des quadriques. Elle apparaît donc comme une représentation CSG où les primitives (non bornées!) sont ces demi-espaces. Aussi dans la suite, et en particulier dans le tableau de synthèse des propriétés des représentations nous considérerons que ces deux représentations n'en font qu'une.

troisième degré

C'est en général à partir de ce degré que l'on commence à parler de *patch*s. Typiquement, la représentation des surfaces frontières consiste à conserver les coordonnées des points de contrôle permettant d'obtenir les équations paramétriques de la surface support. Dans ces schémas de représentation, un morceau est défini par restriction de l'ensemble source de la fonction paramétrique à un rectangle [a,b]x[c,d].

Les systèmes qui utilisent ce schéma de représentation possèdent aussi pour la définition des points de contrôle d'interfaces utilisateur très sophistiquées permettant une bonne interaction graphique au travers de l'écran et de périphériques de désignation (stylo optique, souris ou table à numériser). Cependant ces systèmes ne sont disponibles que sur de très grosses machines et ils font partie de logiciels de CAO.

Il faut noter ici que ces systèmes n'entrent pas strictement dans le cadre de modélisation tridimensionnelle que nous avons fixé au début de ce chapitre ; en effet, ce schéma ne représente en fait que des *surfaces*, les volumes sont créés par des ensembles de ces morceaux de surfaces, mais la validité de ces représentations de volumes ne sont faites que par l'utilisateur, c'est en particulier celui-ci qui a la charge de raccorder les différents morceaux.

La visualisation et le rendu des surfaces bicubiques ont également été très étudiés en synthèse d'images du fait qu'elles permettent de représenter des objets à l'allure moins "géométrique" que les autres schémas de représentation. Plusieurs méthodes utilisent des découpes récursives des patchs : [CATM 74] a introduit cette méthode en la combinant avec celle du *z-buffer*. Dans [LANE 80], Lane et Carpenter, Whitted et Blinn proposent des méthodes de visualisation de surfaces bicubiques avec des techniques de *balayage par ligne*. Enfin, pour la technique du *lancer de rayons*, on peut citer [KAJI 82].

Les schémas de représentation par frontières ne sont pas du tout *uniques* et leur pouvoir de représentation englobe tous les précédents à condition que les surfaces limitant les primitives des autres schémas soient disponibles.

La validation d'une représentation par frontières pose des problèmes algorithmique et théorique intéressants, du fait qu'il faut pouvoir assurer qu'un ensemble de surfaces définit bien un volume qui a un intérieur et un extérieur. En pratique, cette validation, lorqu'elle est effectuée, est faite de façon indirecte en construisant la représentation par frontières à partir d'une autre représentation. Nous pouvons déjà citer ici la méthode employée par [MANT 83] pour la conversion CSG en représentation (approchée) de polyèdres par frontières : l'utilisation des opérations d'Euler assure la validité de la représentation par frontières. De même, [LEVI 80] construit de nouveaux objets à partir de primitives limitées par des quadriques. La représentation de ces primitives étant elles-mêmes valides, il conserve cette validité dans ses constructions.

Le pouvoir de représentation du schéma de représentation par frontières est évidemment très étroitement lié au pouvoir de représentation du schéma de description des surfaces sous-jacent.

Nous terminerons ce paragraphe par des propriétes informelles de ce schéma. Il est assez verbeux bien que n'étant pas redondant et la difficulté pour assurer la validité d'une représentation nécessite l'assistance de l'ordinateur dans cette tâche. Cette dernière fonction peut être réalisée par un langage de description, ce qui conduit en fait à effectuer un changement de mode de représentation.

2.3.6 Les croisements de représentation

On a vu que le pouvoir de représentation des schémas CSG est directement lié au nombre et à la qualité des primitives. Par ailleurs, le schéma de représentation par frontières permet grâce à l'utilisation de morceaux de surfaces B-splines d'avoir un pouvoir de représentation très étendu. Il semble donc intéressant de croiser les deux schémas pour essayer de combiner leurs avantages respectifs. Ceci peut se faire en utilisant comme ensemble de primitives d'un schéma de représentation CSG l'ensemble des solides représentables par un schéma de représentation par frontières. Il va de soi qu'on peut utiliser ainsi le domaine de représentation de tout autre schéma comme ensemble de primitives pour le schéma de représentation CSG.

Cette méthode, simple dans son principe, pose des problèmes puisque les algorithmes devant utiliser une telle représentation doivent être capables d'utiliser chacune d'elle séparément (ce qui n'est déjà pas nécessairement simple) ainsi que la combinaison des deux. Les algorithmes peuvent aussi utiliser une représentation basée sur un schéma unique à condition d'avoir un algorithme de transformation dans celui-ci. Se pose donc dans ce cas particulier le problème de la conversion d'un schéma dans un autre.

2.3.7 Conclusion

[REQU 80] présente un tableau comparatif des propriétés de ces différents schémas de représentation. Nous l'avons utilisé comme point de départ pour construire le notre en y apportant quelques mises à jour. Nous y trouvons successivement les qualités ou possibilités suivantes :

- non ambiguïté, unicité, domaine, validabilité, concision, facilité de création : ces termes ont été expliqués dans la section 2.2 ;
- ensemble vide : est-ce facile de reconnaître qu'un objet est en fait l'ensemble vide ?
- égalité : peut-on tester l'égalité de deux objets ?
- dessin au trait : peut-on obtenir des tracés des contours de l'objet ?
- image : peut-on obtenir des images ombrés de l'objet ?
- interaction : le schéma de représentation se prête-t'il au développement d'une interface utilisateur *conviviale* ?

	instanciation de primitives	énumération spatiale	arbre de construction	représentation par frontières explicites	représentation par frontières implicites	représentation par morceaux de surface	extrusion simple	extrusion générale
non ambiguité	X	X	X	X	X		X	X
unicité	X	X		X			?	
domaine		X	X	X	X		?	X
validabilité	X	X	X				X	
concision	X		X			?	X	X
facilité de création	X		X			?	X	X
ensemble vide	X	X		X			X	X
égalité	X	X					?	
dessin au trait	X			X			X	
image	X		X	X	?		X	
interaction			X	X			X	

figure 2.8

Ce tableau nous permet de constater que tous les schémas de représentation sont non ambigus, à l'exception toutefois du schéma de représentation par ensembles de morceaux de surfaces (nous avons cependant déjà indiqué que cette exception ne devait pas être réellement considérée comme un schéma de représentation tridimensionnel). Cette qualité ne représente donc pas un critère de choix d'un schéma de représentation.

Nous avons aussi dit que la validité des représentations était déterminante, ceci quels que soient les algorithmes utilisant la représentation solide tridimensionnelle, et donc quelles que soient les applications. Nous pouvons constater que tous les schémas de représentation ne sont pas égaux devant cette qualité. En pratique, les représentations dans des schémas difficilement validables sont obtenus par conversion depuis un schéma dans lequel on est sûr de n'avoir que des représentations valides ; c'est le cas pour :

- le schéma de représentation par frontières,
- le schéma de représentation CSG avec des primitives non bornées.

Ces deux schémas, s'ils peuvent présenter des avantages pour certains algorithmes, ne sont donc pas ceux que l'utilisateur doit manipuler directement. Par contre, rien n'empêche de les créer simultanément à la donnée par l'utilisateur d'une représentation dans un autre schéma. Du point de vue de la validité des représentations, les schémas à retenir sont donc l'instanciation de primitives, l'énumération spatiale, la représentation CSG avec des primitives bornées, et les extrusions par rotation et translation.

Si l'on se place du point de vue de l'interaction, le schéma de représentation par frontières doit retenir notre attention, en effet cette structure de données contient les informations sur les éléments géométriques caractéristiques de l'objet (sommets, arêtes, faces,...) et c'est ceux-ci que l'utilisateur va vouloir désigner. C'est donc une représentation qu'il peut être intéressant de conserver simultanément à une autre pour bénéficier de ses avantages pour l'interaction.

Le point de vue facilité de création n'est pas très éloigné du point de vue interaction. Cependant, trois schémas se détachent des autres pour cette qualité. La représentation CSG permet d'obtenir rapidement de nouvelles formes par des combinaisons de primitives. La représentation par extrusion permet d'avoir une bonne interaction graphique tout en permettant d'obtenir des objets qui seraient longs ou difficiles d'obtenir avec une autre méthode (on s'aperçoit ainsi que l'évaluation de la facilité de création ressemble à celle du domaine). Enfin les assemblages de morceaux de surfaces bicubiques eux-aussi permettent une bonne interaction tout en possédant un domaine qu'ils ne partagent pratiquement avec aucun autre schéma de représentation. Rappelons toutefois le problème de la validité des représentations dans ce schéma.

2.4 CONVERSION ENTRE LES REPRESENTATIONS

La conclusion à tirer du tour d'horizon précédent est qu'il n'existe pas de représentation qui soit meilleure que les autres sur tous les points. Le besoin de conversion se fait sentir par exemple pour passer d'une représentation commode pour l'interaction à une représentation commode pour l'affichage.

Comme nous l'avons indiqué en 2.2 deux schémas équivalents ont le même domaine. Des restrictions (au sens mathématique : restriction de l'ensemble de départ et/ou de l'ensemble d'arrivée) doivent être apportées aux schémas si on veut pouvoir parler d'équivalence.

Le but d'une conversion est de trouver *une* représentation consistante à une représentation de départ dans un autre schéma. Considérons l'exemple suivant, où d'une part la non-unicité des représentations CSG et par frontières fait apparaître plusieurs images pour le même objet, et d'autre part où la relative ambiguité de la représentation par voxel nous fait représenter l'ensemble A des antécédents de l'image par ce schéma de l'objet qui nous intéresse (*Ouf...!*) (cf. figure 2.9)

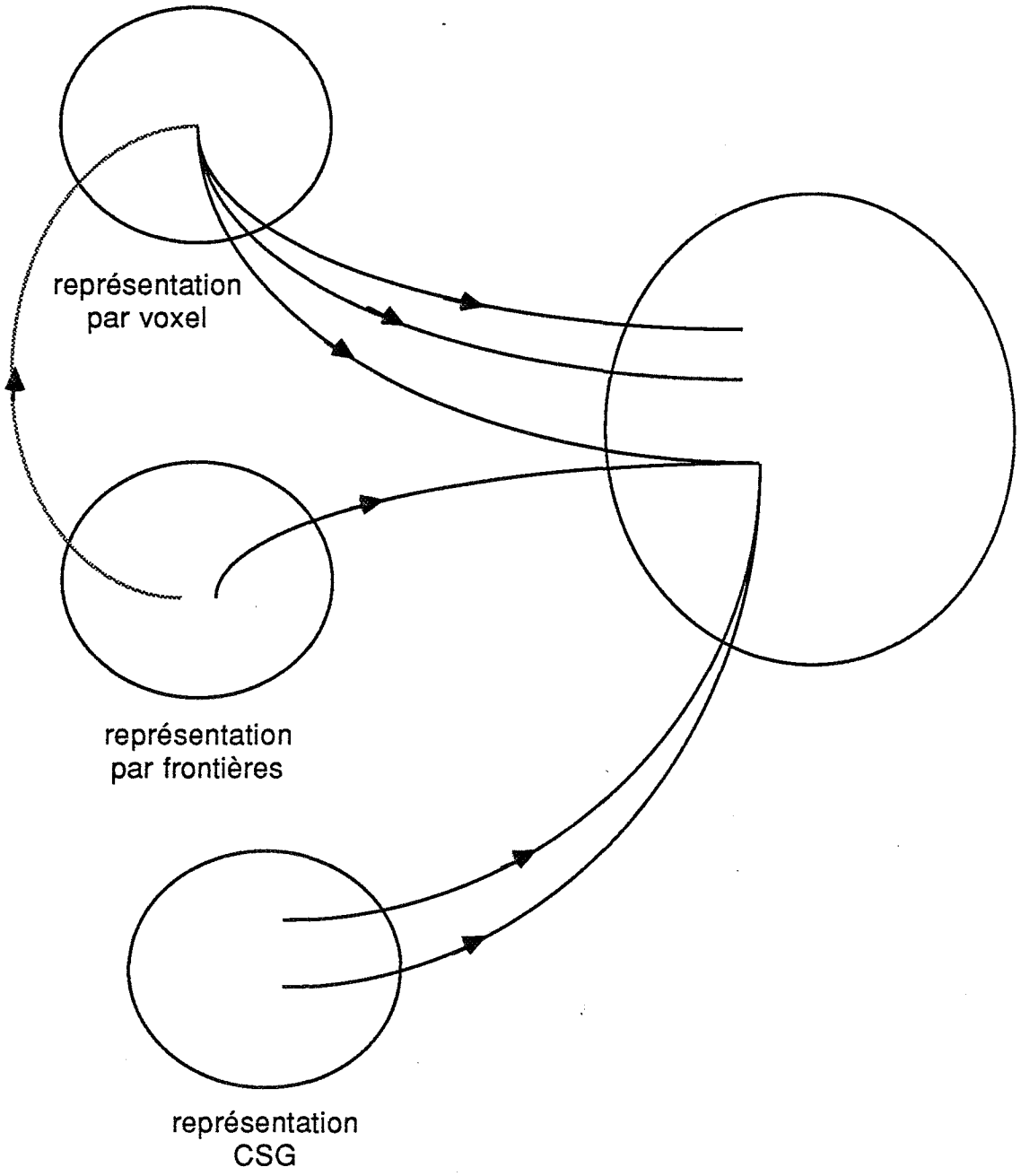


figure 2.9

Comme le schéma de représentation par voxel est le seul qui soit ambigu, étudions-le d'abord à part pour les problèmes de conversion. Pour lui, on ne pourra pas chercher de représentations *équivalentes* mais seulement des représentations *consistantes*.

Premièrement, si on veut passer d'une autre représentation à la représentation par voxel cela ne pose pas de problèmes théoriques. En effet, les autres schémas sont non ambigus et de ce fait une représentation dans un de ces schémas représente un *solide abstrait* unique. Or nous avons supposé (cf. 2.2) que l'on savait dire de tout point s'il était à l'intérieur ou à l'extérieur du solide : c'est en particulier le cas pour les points centres des voxels. Pratiquement, on ne peut toutefois remplir la matrice tridimensionnelle de voxels par trois boucles parcourant chaque voxel puis testant le centre de chaque voxel par rapport à l'objet ; ce serait beaucoup trop long. Deuxième problème pratique dans la mise en œuvre de cette conversion : quel cube englobant va-t'on utiliser et jusqu'où va-t'on le découper ?

[TAMM 84] présente une méthode pour la conversion représentation par frontières en énumération spatiale. [LEE 82] propose une conversion CSG en énumération spatiale.

Deuxièmement, si on veut passer d'une représentation par voxel à une autre... cela pose d'énormes problèmes qui semblent pour l'instant difficiles à résoudre surtout dans un contexte purement algorithmique. Nous donnons ici une idée de l'ampleur des difficultés :

Nous avons un cylindre représenté par une matrice de voxels. Cette représentation subissant quelques traitements de synthèse d'images produit une image (matrice de pixels) regardée et analysée par un observateur. Si le processus représentation-visualisation ne s'est pas trop mal passé, celui-ci doit être capable de reconnaître l'image d'un cylindre et ainsi de réussir à déterminer (connaissant le point de vue) une approximation de la mesure de son rayon et de sa hauteur. Cet observateur est donc capable de reconstruire des représentations CSG, par frontières,... depuis l'image finale : l'information *cylindre* est donc bien contenue quelque part et dans l'image finale et dans la représentation par voxel qui a permis de l'obtenir. Le même observateur regardant une matrice de cubes pourrait aussi y reconnaître le cylindre et effectuer la mesure de sa hauteur et de son rayon. Mais dans les deux cas l'extraction de cette information a fait appel à l'intelligence de l'observateur. Pour arriver au même résultat au moyen d'une machine, il semble donc nécessaire de faire appel à des techniques d'intelligence artificielle.

En résumé, le schéma de représentation par voxels ne peut être utilisé que comme un utilitaire pour un autre schéma de représentation dans les applications de visualisation, de calculs de masse, de calculs de moments d'inertie,... mais il est inadapté à une représentation universelle étant donnée l'impossibilité d'obtenir une autre représentation à partir de lui.

Nous nous plaçons maintenant dans le contexte des autres schémas de représentation qui sont tous non ambigus. Le problème de conversion en devient aussi plus simple dans sa formulation : on veut obtenir une représentation équivalente dans un autre schéma à une représentation donnée.

Ce problème plus contraignant a aussi plus volontiers une solution algorithmique. Nous allons étudier les conversions cas par cas en les classant par schéma de représentation but.

Instanciation de primitives

Tout dépend ici des primitives connues. Dans les cas *normaux*, c'est à dire pour des primitives du genre cube, quadriques diverses... on ne pourra convertir dans ce schéma que des représentations CSG restreintes à l'opération *collage*.

Conversion peu intéressante : si ce schéma est suffisant pour représenter tous les objets nécessaires à une application, autant n'utiliser que lui et l'utiliser tout de suite, il n'apporte aucune facilité par rapport aux autres. Une exception toutefois si l'on dispose de matériel ou de logiciel utilisant directement cette représentation.

CSG

Cette représentation semble difficile à obtenir de façon *réfléchie* par algorithme. En effet, la structuration voulue par l'utilisateur et éventuellement indispensable ne peut être obtenue que par de l'intelligence. Par ailleurs existe-t-il une structuration meilleure qu'une autre pour représenter, par exemple, une maison en CSG ?

Pour une modélisation grossière qui servira à une visualisation de loin, on va préférer représenter la maison par un cube percé par les ouvertures, alors que de près, on préfèrera une union des murs de façon à pouvoir pénétrer à l'intérieur de la maison.

Cependant, en restreignant le domaine d'application, on peut trouver des algorithmes et des techniques intéressants. Citons [VOSS 85] qui construit un arbre CSG pour des pièces de mécanique représentées par ailleurs par des extrusions par translation. Qu'une telle restriction permette d'obtenir des résultats n'est pas très étonnant ; en effet en ne considérant que des objets obtenus par extrusion en translation, [VOSS 85] se ramène à un problème en deux dimensions pas très éloigné de la reconnaissance des formes ou de l'analyse d'images, domaines dans lesquels on commence à avoir de bons résultats pratiques.

Représentation par extrusion

Le domaine de ce schéma est très petit et ne contient que d'infimes parties des domaines de représentation des autres schémas, la conversion dans ce schéma n'est ainsi possible que pour des cas sans intérêt : cylindre, cube,...

Représentation par frontières

Ce schéma de représentation est le plus intéressant à obtenir : la synthèse d'images foisonne de méthodes d'affichage de surfaces.

La conversion depuis le schéma de représentation par instanciation de primitives est triviale : il suffit d'intégrer dans l'algorithme des équations des faces limitant les primitives ; ces dernières étant souvent *simples*, ces équations le sont aussi.

La conversion d'un schéma CSG où n'intervient que l'opération d'assemblage n'est pas beaucoup plus compliquée dans la mesure où l'on doit éliminer les faces communes à deux solides [REQU 80].

Trouver les frontières d'un solide représenté par CSG est déjà une activité plus complexe, mais cette complexité dépend très fortement du niveau d'explicitation que l'on désire obtenir dans la représentation par frontières. En effet, d'une certaine façon, cette conversion consiste à transformer une représentation implicite en une représentation explicite. On va donc avoir à résoudre des systèmes d'équations. Pour une intersection par exemple, on doit pouvoir calculer explicitement la courbe intersection de deux surfaces... On ne sait résoudre ces systèmes d'équations que dans des cas relativement simples.

Nous avons distingué trois schémas de représentation par frontières suivant le degré des polynomes représentant les supports des frontières. Nous allons reprendre cette classification.

premier degré — C'est curieusement la conversion la plus compliquée parce que c'est celle où l'on va demander la représentation la plus explicite. En effet, on veut obtenir la liste des sommets des polygones et non des équations et des inéquations du premier degré. Nous l'étudierons plus loin, en effet on ne peut convertir de façon exacte dans ce schéma que des solides polyédriques, or rares sont les représentations CSG limitées à de tels solides.

second degré — C'est ici que l'on trouve la majorité de conversions effectivement implantées car les primitives manipulées dans les modeleurs CSG sont typiquement des solides dont les frontières sont des quadriques : cylindres de révolution, cônes, sphères, ellipsoïdes,... Ces implantations font partie de systèmes commerciaux de modélisation pour la CAO. Ces derniers se contentent d'une représentation qui reste implicite, comme nous l'avons indiqué en citant ce schéma en 2.3.5.

troisième degré — Nous ne connaissons rien qui puisse se classer dans cette catégorie.

Enfin la conversion depuis un schéma de représentation par extrusion (que ce soit par translation ou par rotation) en une représentation par frontière est, elle, beaucoup plus simple car les surfaces de base sont elles-mêmes représentées par des assemblages de courbes. Il s'agit en fait à peine d'une conversion : on peut très bien voir le schéma de représentation par extrusion comme un schéma de représentation par frontières où les frontières sont représentées par des courbes ; ce qui revient à dire que, sémantiquement, on considère la représentation de courbe comme une représentation de surface.

2.5 FACETTISATION

L'ensemble des polyèdres est un sous-ensemble remarquable de l'ensemble des solides abstraits :

- d'une part, il se prête bien au schéma de représentation par frontières qui, on l'a vu, est très important en synthèse d'images,
- d'autre part, il est *intuitivement* dense dans l'ensemble des solides abstraits.

Cette deuxième caractéristique signifie que l'on peut approcher autant que voulu un solide abstrait par un polyèdre. Cette notion d'approximation nécessite en toute rigueur de définir une topologie. Par exemple, on peut choisir la topologie métrique engendrée par la distance qui à deux solides abstraits associe la mesure au sens de Lebesgue de la différence symétrique des deux solides abstraits. Pour cette topologie l'ensemble des polyèdres est effectivement dense dans l'ensemble des solides abstraits.

Supposons, pour l'instant, qu'une topologie, celle que nous avons citée ou une autre, est effectivement définie sur l'ensemble des solides abstraits. Nous pouvons alors définir ce qu'est une conversion approchée de représentation dans un schéma de représentation n'acceptant que des polyèdres :

- la densité nous permet de dire que tout voisinage d'un solide abstrait contient un polyèdre,
- ce polyèdre peut être représenté dans le nouveau schéma.

La notion de conversion approchée nécessite la définition d'un voisinage pour tous les solides abstraits à convertir. La façon la plus simple est d'avoir une topologie métrique et de définir l'ensemble des voisinages du solide abstrait comme l'ensemble des boules ouvertes centrées sur le solide.

Pratiquement, la définition du voisinage utilisé n'est pas toujours explicite : lorsque [CATM 74] subdivise des bicubiques pour leur affichage, il les subdivise jusqu'à ce que chaque morceau ne recouvre plus qu'un pixel après projection sur l'écran. Le morceau de patch est alors visualisé comme s'il s'agissait d'un polygone : on calcule l'intensité en utilisant *une* normale. La décomposition en polygones d'un patch n'est réalisée ainsi que d'une façon implicite et ceci seulement au moment de l'affichage.

[LANE 79] et [LANE 80] présentent une méthode un peu plus explicite puisqu'un test de *planéité* des patchs est présent dans l'algorithme (on peut ainsi stopper la subdivision avant d'arriver à la taille du pixel). Par ailleurs l'idée de la subdivision reste la même que dans [CATM 74].

Ces deux exemples montrent l'utilisation d'une conversion *représentation par frontières* en une *représentation par frontières polygonales* ; cependant la conversion n'est pas explicite, n'étant obtenue que lors de l'affichage.

L'autre possibilité est de définir d'abord le voisinage puis, ensuite, de fabriquer le polyèdre approché appartenant à ce voisinage. L'inconvénient de cette méthode par rapport à la première lorsqu'elle est utilisée pour l'affichage est la difficulté du *bon choix* du voisinage :

- si on le choisit trop grand, on risque lors de la visualisation de laisser apparaître les polygones,
- si on le choisit trop petit, on risque d'engendrer un polyèdre comportant de trop nombreuses faces trop petites, d'où un coût de calcul inutilement plus grand et des risques d'aliassage plus difficile ou plus long à traiter à cause de nombreux petits objets.

Les trois méthodes citées ci-dessus [CATM 74], [LANE 79] et [LANE 80] s'adaptaient, elles, à la définition de l'écran.

Cependant, seule la dernière méthode peut être honnêtement qualifiée de *conversion* au sens où nous l'avons utilisé dans le paragraphe précédent. De plus très souvent ni la topologie ni le voisinage ne sont précisés : les objets sont découpés en un nombre de morceaux fixé par l'utilisateur.

Remarques

1. On pourrait parler de *conversion simultanée* à l'affichage ou de *conversion a priori.*
2. La visualisation par lancer de rayon peut de même que [CATM 74] être considérée comme une facettisation simultanée à l'affichage : pour chaque pixel de l'écran on regarde quels sont les objets intersectés, pour le plus proche on calcule *une* normale.
3. Cette deuxième remarque nous invite à penser que si la facettisation (associée à un algorithme de visualisation rapide tel que le *z-buffer* lorsqu'il est câblé), doit permettre d'obtenir une image plus rapidement qu'un *simple* lancer de rayon, la subdivision ne doit pas se faire simultanément à l'affichage.
4. Pour atténuer la remarque 3, il est évident qu'il y a toujours des solutions intermédiaires.

Dans ce paragraphe, nous avons évoqué jusqu'ici les problèmes de conversions approchées de primitives ou de patchs en facettes. Il nous reste à parler du problème de conversion du schéma de représentation CSG en facettes.

Deux possibilités se dégagent :

- soit la conversion se fait par l'intermédiaire du schéma de représentation par frontières ; et alors les problèmes sont ceux cités en 2.4 pour la première conversion puis ensuite ceux de la conversion frontières-facettes.
- soit on effectue d'abord la conversion primitives-facettes et on évalue les opérations booléennes.

Pour cette deuxième méthode, il faut donc savoir effectuer les opérations booléennes sur les polyèdres. Il s'agit déjà d'une opération beaucoup plus simple que celle d'évaluer explicitement les résultats des mêmes opérations sur des primitives plus générales, par exemple des quadriques. C'est cependant une opération délicate sur laquelle on ne trouve qu'assez peu de publications. Celles-ci sont par ailleurs assez récentes.

[MANT 83] se base sur les opérateurs d'Euler (ajout et suppression de sommets et d'arêtes) pour assurer la validité des représentations par frontières qu'il crée. Cependant, dans son article, il présente essentiellement une amélioration de la complexité de la recherche des intersections de deux solides grâce à la structure de données *EXCELL*. Il évite en particulier tous les problèmes de faces coplanaires.

[YAMA 84] utilise la structure de données *Winged Edge* proposée par [BAUM 75] pour décrire la représentation par frontières. Cette structure de données permet, elle aussi, d'utiliser les opérateurs d'Euler. Il triangule les surfaces susceptibles de s'intersecter. Ainsi il simplifie le problème d'intersections des faces en n'ayant à étudier que le cas d'intersections de triangles.

[REQU 85] présente une méthode assez formelle basée sur la *classification* (cf. [TILO 80]). Il ne traite pas en particulier des cas (souvent fréquents et embêtants) des faces coplanaires de polygones.

[LAID 86] travaille lui sur des faces convexes. Les primitives usuelles se facettisant de façon naturelle avec des faces convexes et cet algorithme ne générant à partir de ces données que des faces elles aussi convexes, on ne perd pas en généralité avec cette contrainte. Par contre, il détaille le cas des faces coplanaires.

[MICH 87b] aborde lui le problème des faces coplanaires et montre qu'il est en fait un problème de précision (ou de représentation) des nombres.

2.6 LES INFORMATIONS NON GEOMETRIQUES

Nous avons jusqu'ici décrit le mode de représentation des informations de type géométrique. D'autres informations sont nécessaires à la synthèse d'images, et ceci d'autant plus que l'on voudra obtenir un grand degré de réalisme à la visualisation. Nous classerons ces autres informations en deux catégories suivant qu'elles sont liées ou non à l'information géométrique.

2.6.1 Les informations indépendantes de la géométrie

Ces informations concernent les propriétés physiques associées aux objets solides. Nous pouvons citer le matériau, la couleur, divers coefficients de réflexion ou de transmission de la lumière, ceci si nous restons dans le type d'informations qui vont être utiles pour la synthèse d'images. Dans un cadre plus large, on trouve aussi dans ce genre de données des informations de masse volumique, de conductivité ou toutes autres grandeurs qui peuvent être utiles dans un cadre de CAO pour des calculs sur les objets modélisés.

Ces informations sont typiquement associées à un objet solide homogène pour le type d'information donnée. Par rapport aux représentations d'objets solides que nous avons décrites dans les paragraphes précédents, ces informations doivent donc être associées à *un* objet. Ceci nous conduit à faire maintenant une différence entre modélisation (et représentation) d'un objet solide d'une part, et d'une scène composée d'objets solides d'autre part. Nous allons reprendre de ce point de vue les différents schémas de représentation précédemment cités ; ils peuvent se classer en deux catégories :

- ceux qui modélisent les objets un par un et qui nécessitent un niveau supérieur de modélisation et de représentation pour positionner les différents objets les uns par rapport aux autres,
- et ceux qui permettent aussi bien de représenter un objet qu'une scène complète.

Dans la première catégorie, on trouve les schémas de représentation par instanciation de primitives, par extrusion et par frontières. Pour ceux-ci les informations de type caractéristiques physiques sont homogènes pour les objets et sont associées à chacun d'eux.

Les schémas de représentation par énumération spatiale et CSG font eux partie de la deuxième catégorie. Etudions-les séparément.

Enumération spatiale

Deux cas peuvent être envisagés : soit une matrice de voxels décrit un objet solide, soit une matrice de voxels décrit une scène. Du point de vue CAO, c'est plutôt la première option qui sera utilisée puisque dans ce domaine d'application, l'ingénieur représente une pièce et veut connaître des caractéristiques physiques de cette pièce. Dans ce cas, les informations physiques sont associées à la matrice entière.

Du point de vue synthèse d'images, c'est plutôt l'ensemble de la scène que l'on va représenter dans une matrice de voxels. En effet, de telles matrices occupent beaucoup de place mémoire et il est peu raisonnable de conserver une telle matrice pour chacun des objets, d'autant que ceux-ci risquent d'être très nombreux.

En résumé, si *un* objet est représenté dans la matrice de voxels, les informations physiques sont globales pour la matrice, dans le cas contraire, on va trouver ces informations pour chaque voxel.

CSG

Du fait de la structuration, un arbre CSG peut aussi bien représenter, du point de vue de l'utilisateur, un objet qu'une scène. Nous devons ici faire attention à la sémantique des opérations. Prenons l'exemple d'une scène complexe où sont modélisés entre autres un verre et une table. La table est construite comme la réunion d'un plateau et des quatre pieds, c'est un objet cohérent auquel on va associer une matière (le bois), une

couleur,... Par ailleurs, l'ensemble verre plus table constitue lui aussi un tout par rapport au reste de la scène ; on va donc naturellement réunir ces deux objets. Par contre, il n'y a pas unité de matière ou de couleur entre eux. Les informations physiques dont nous parlons dans ce paragraphe ne sont naturellement associées ni aux feuilles de l'arbre (on veut considérer l'unité de la table), ni aux nœuds, ni aux racines. Elles doivent pouvoir s'inscrire à n'importe quel niveau de la hiérarchie arborescente suivant les ensembles logiques qu'elles concernent.

2.6.2 Les informations dépendant de la géométrie

Nous donnerons simplement un exemple dans ce paragraphe. Si nous voulons modéliser dans un schéma de représentation CSG un bouchon de champagne en liège, nous pouvons du point de vue géométrique faire la réunion de deux cylindres. Pour donner l'effet du liège, nous pouvons utiliser les textures tridimensionnelles telles qu'elles ont été décrites dans [PEAC 85] ou [PERL 85]. Ces textures sont en fait des applications dont la source est l'espace à trois dimensions de modélisation. Le problème est de positionner chacun des deux cylindres qui représentent géométriquement le bouchon d'une façon cohérente dans l'espace des textures ; il ne suffit pas de dire simplement que chacun des deux cylindres est constitué de liège, il doit y avoir une cohérence entre le liège qui constitue chacunes des deux parties puisqu'en fait ce bouchon est taillé dans un unique morceau de liège. De même, changer la longueur du bouchon ne doit pas pour autant allonger la texture. D'un autre côté, on peut avoir besoin d'agir sur la texture sans agir sur l'objet pour changer par exemple le grain du liège. Le même genre de problème existe pour des textures planes à plaquer sur des surfaces. Il faut donc disposer de transformations géométriques qui n'agissent que sur la texture, et de transformations géométriques qui n'agissent que sur les objets.

2.7 LES LANGAGES DE SAISIE

On peut classer en deux catégories les modes de saisie des données par l'utilisateur. Soit un mode dit interactif est employé où l'utilisateur désigne des points sur son écran, ce qui lui permet de choisir des options dans des menus, de désigner des points dans un plan ou sur un objet déjà visualisé. Notons, dans ce dernier cas, que la détermination du point dans la structure de données tridimensionnelles n'est pas triviale puisqu'on doit retrouver une information tridimensionnelle à partir de l'information à deux dimensions obtenue par le positionnement d'un curseur sur l'écran.

L'autre façon de décrire les informations géométriques consiste à utiliser un langage de description. Nous allons étudier quatre langages déjà existants puis nous présenterons le nôtre.

2.7.1 PICTUREBALM, GRAMPS, QUADRIL, PADL

Nous présentons dans ce paragraphe quatre langages de description. Leur étude va permettre de préciser leurs points communs et leur différences. Ceci nous donnera des indications sur ce que peut être un langage destiné à la synthèse d'images et un peu universel.

2.7.1.1 PICTUREBALM

Nous commençons par le langage PICTUREBALM décrit dans [GOAT 80]. Bien que basé sur le langage LISP, l'utilisateur voit une syntaxe et des structures de contrôle de type PASCAL. Ce langage est interactif dans le sens où il permet d'obtenir une visualisation aussitôt après avoir décrit un objet. Les objets manipulés sont des ensembles de points définis par leurs coordonnées dans un repère tridimensionnel. Le sens donné à cet ensemble de points repose sur un pilote extérieur au programme. Cette signification est éventuellement dépendante du périphérique : par exemple, à la date d'écriture de l'article les pilotes qui existaient pour les périphériques à dessin au trait interprétaient ces ensembles de points comme un ensemble connexe de segments de droite ; par contre un interprète pour représentation de surfaces ne pouvait accepter que certains ensembles de points décrivant des surfaces de types déterminées.

Une hiérarchie de représentation peut être obtenue grâce à l'emploi d'un opérateur de groupement logique '&'. Toutefois, on ne sait pas à la lecture de l'article si cette représentation hiérarchique est conservée par l'interpréteur ou bien si ce dernier la transforme immédiatemment en un ensemble de lignes brisées. Cependant, dans les deux cas, la conservation d'un fichier texte *source* des objets représentés permet à l'utilisateur de conserver cette hiérarchie.

Des transformations géométriques peuvent être appliquées aux objets. L'utilisateur peut en définir de nouvelles dont la syntaxe d'utilisation sera la même que celles qui sont déjà définies, ceci grâce à un mode procédural accessible au niveau de l'interpréteur.

Ce langage n'a pas de notion d'objets tridimensionnels, et c'est plutôt un langage interprété, complété par des primitives graphiques (essentiellement le tracé de segment de droite), une structure de données : l'ensemble de points, et quelques fonctions pour travailler sur cette structure. Cependant il est intéressant de retenir l'idée de *procédure* qui permet d'étendre le langage. De plus, l'implantation du langage fait que les procédures créées par l'utilisateur peuvent être utilisées avec les mêmes règles syntaxiques (d'ailleurs assez souples) que les procédures qui sont prédéfinies.

2.7.1.2 GRAMPS

La structure de données interne de ce langage est beaucoup mieux décrite dans l'article [ODON 81] que celle de PICTUREBALM. GRAMPS connaît cinq types de données :

objets primitifs — Ce sont des ensembles de points décrivant des segments de droite associés à une matrice de transformation permettant translation, rotation et changement d'échelle.

mondes — Les deux *mondes* sont en fait des matrices de transformation globales. Il y en a deux pour pouvoir gérer des vues stéréoscopiques.

groupes — La structure hiérarchique est obtenue grâce à la notion de *groupe*. Un groupe peut en effet regrouper des groupes déjà définis. De plus, ils contiennent une matrice de transformation qui s'applique

à tous les objets du groupe. L'utilisateur ne manipule pas directement la structure de données *groupe* mais il peut créer des groupes et ajouter ou supprimer des objets dans un groupe déjà existant grâce aux mots-clés : *INSERT* et *REMOVE*.

frame object — Cette structure est en fait la même que celles d'un objet sauf qu'ici les coordonnées sont stockées dans un tableau et un nombre permet de choisir celles que l'on va utiliser. Cette notion est utile car elle permet d'effectuer certaines animations en temps réel, en permettant de garder plusieurs positions successives du même objet.

synonyme — Ce type contient comme tous les autres une matrice de transformation, mais au lieu de contenir comme un objet un ensemble de coordonnées, il contient un pointeur sur les coordonnées d'un autre objet. Cette structuration permet d'éviter une grande partie de verbiage dans la description et par la même occasion d'économiser de la place mémoire.

On voit ici un langage permettant de décrire des structures hiérarchiques où les transformations sur les objets peuvent intervenir à tous les niveaux. Comme PICTUREBALM il n'a toutefois aucune connaissance d'objets solides, il ne manipule que des ensembles de segments.

2.7.1.3 QUADRIL

Le nom QUADRIL signifie *QUADRIc-surface body Language*. Ce langage [LEVI 80] permet de manipuler des quadriques qui sont des surfaces définies par des équations du second degré en fonction de coordonnées dans un repère cartésien. Comme dans PICTUREBALM et dans GRAMPS, des transformations géométriques peuvent être appliquées aux objets. Pour la structuration hiérarchique, QUADRIL dispose de deux niveaux de langage. Le premier, la notation Algébrique, permet de réunir des objets grâce à l'opérateur '+', d'obtenir l'intersection de deux objets grâce à l'opérateur '*' ou encore la différence grâce à '-'. Le deuxième niveau, la notation Booléenne, permet de décrire complètement l'arbre de représentation.

Encore une fois, ce langage ne possède pas de notion de volume. Il manipule des surfaces et son unique moyen de sortie est la génération d'images au trait du contour et des intersections des surfaces décrites. Comme GRAMPS, la structure interne est une collection d'arbres (accessibles par leurs identificateurs) où les transformations géométriques peuvent intervenir à tous les niveaux. En plus QUADRIL permet d'utiliser les opérations booléennes sur les objets décrits. Signalons enfin que sa syntaxe est assez verbeuse et que les paramètres ne peuvent être munis d'identificateurs, ils doivent être donnés sous forme numérique.

2.7.1.4 PADL-1 et PADL-2

Nous avons utilisé deux sources de renseignements : [VOEL 78] pour PADL-1 et [BROW 82] pour PADL-2. Ces deux langages ne diffèrent au niveau où nous nous plaçons, que par une syntaxe différente, aussi dans la suite nous parlerons globalement de PADL.

Ce langage a une connaissance volumique ; c'est une véritable implantation du schéma de représentation CSG. On ne décrit pas directement l'arbre mais le langage permet de définir au moyen d'une notation algébrique des actions qui construiront cet arbre. Dans PADL, beaucoup d'entités peuvent être utilisées au moyen d'identificateurs : en plus des objets bien sûr, il y a les paramètres numériques et les transformations (par contre, on ne sait pas si un ensemble de transformations géométriques peut lui aussi être muni d'un identificateur).

2.7.2 Conclusion

Tous ces langages ont donc, à un niveau explicite ou implicite, une représentation hiérarchique analogue à celle des arbres de représentation CSG. Ce schéma de représentation est donc ainsi reconnu comme celui le plus adapté à une description par langage.

Outre ses nombreuses qualités déjà citées dans le cadre de modélisation solide, il en présente d'autres plus spécifiques à la description de scènes de synthèse d'images :

- la structuration arborescente permet d'attribuer des qualités à tout un sous-arbre d'objet (par exemple, il est facile d'indiquer la matière de tout une sous-arborescence) ;
- cette même structuration arborescente permet de n'appliquer des transformations géométriques qu'à des sous-arbres, cela revient à pouvoir considérer que chaque objet possède un repère local ;
- possibilité de description de scènes complètes et pas uniquement d'objets isolés ;
- possibilité de description d'animation par la seule modification de paramètres de certaines transformations géométriques.

2.7.3 CASTOR

2.7.3.1 Les raisons d'un choix

Nous commençons cette section en indiquant les raisons qui nous ont guidés dans le choix d'une représentation des données géométriques tridimensionnelles.

Le premier choix à effectuer réside dans l'alternative entre un système de modélisation interactif et une description de scènes par langage. En ce qui nous concerne, la question ne se posait pas autrement que formellement à la date des travaux présentés dans ce rapport. En effet, nous ne disposions que d'un seul équipement permettant de visualiser des images (cf. annexe). De plus, cet équipement n'est pas du tout adapté à la création de programmes interactifs :

- pas de gestion locale d'une souris ou d'un autre dispositif de désignation sur l'écran,
- pas de gestion locale de menus,
- pas de gestion de déplacements de blocs de pixels sur l'écran.

Toutes ces fonctions auraient dûes être simulées sur l'ordinateur hôte avec comme résultats de piètres performances.

Par ailleurs, nous ne disposions d'aucune bibliothèque spécialisée *multifenêtrage* ou *interaction graphique* sur l'ordinateur hôte en couplage avec ce matériel de visualisation. De toutes façons, il nous semble difficile de réaliser une application interactive *portable*, même en utilisant ces bibliothèques. En effet, il nous paraît indispensable de tenir compte des spécificités du matériel graphique connecté, tout comme un éditeur de texte comme *vi* tient compte des caractéristiques des terminaux alphanumériques qu'il contrôle... et les variations entre les terminaux alphanumériques sont très faibles par rapport aux variations entre les périphériques graphiques, malgré tout les paramètres descriptifs d'un terminal alphanumérique sont déjà fort nombreux.

C'est donc les questions de portabilité et de performances qui nous ont fait éliminer les solutions interactives. C'est aussi pourquoi dans ce qui précède, nous n'avons qu'assez peu évoqués les systèmes interactifs.

Au niveau du choix de la représentation de base que l'utilisateur manipule, il ne restait alors que peu de choix, en effet :

- les représentations par instanciations de primitives et par extrusion simple ont des domaines trop limités ;
- les extrusions et les représentations par assemblages de morceaux de surface sont surtout intéressantes dans une utilisation interactive que nous avons écartée ;
- la représentation par énumération spatiale n'est pas du tout adaptée à la description de scènes : comme nous l'avons vu il s'agit d'une représentation à obtenir par conversion depuis une autre représentation.

Il nous restait donc à choisir entre la représentation par arbre de construction et la représentation par frontières, et comme nous l'avons indiqué en 2.7.2, la première est beaucoup plus adaptée à une description par langage.

Nous avons donc choisi la représentation par arbre de construction comme base de notre système de synthèse d'images. Nous allons maintenant décrire le langage de modélisation CASTOR. Ce langage a été décrit dans [BEIG 86b].

2.7.3.2 La représentation interne

Tout objet porte un nom ; celui-ci est un identificateur alphanumérique commençant par une lettre minuscule. Lors de l'analyse sémantique, une table des identificateurs gérée par *hash-coding* est tenue à jour. Au moyen de cette table, on peut retrouver le pointeur sur le graphe qui définit un objet. Un objet, une fois qu'il a été défini, peut être utilisé pour la définition d'un ou plusieurs autres objets.

Par exemple, si les objets *roue* et *carrosserie* ont déjà été définis, nous pouvons écrire :

```
voiture = $U(
   carrosserie,
   @t(X1,Y1,0) roue,
   @t(X2,Y1,0) roue,
   @t(X2,Y2,0) roue,
   @t(X1,Y2,0) roue
   );
```

Ces lignes dans le fichier d'entrée vont définir *voiture* comme la réunion (*$U*) de la *carrosserie* et de quatre *roue*s positionnées grâce aux translations (*@t(...)*) qui utilisent les paramètres numériques *X1*, *Y1*, *X2* et *Y2*.

La représentation interne sera :

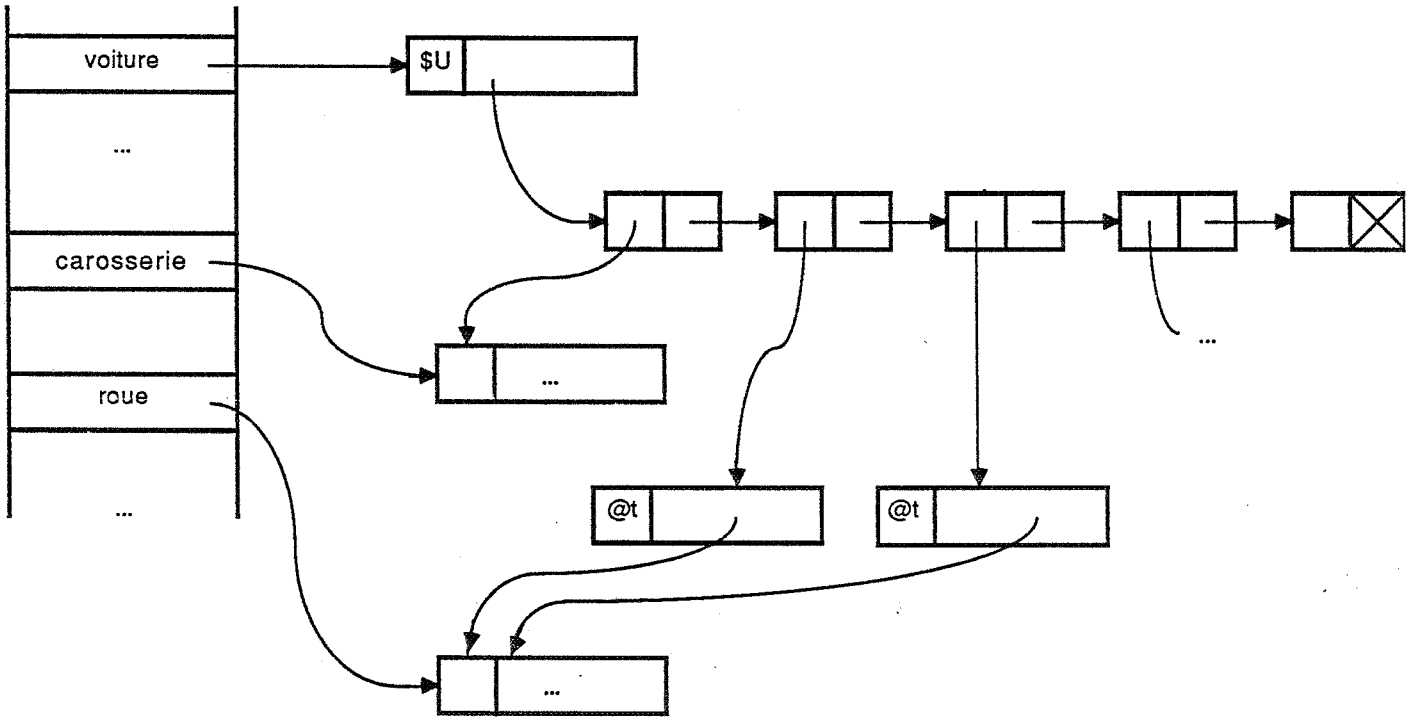


figure 2.10

Les graphes obtenus sont cependant des graphes sans circuits, et qui de plus possèdent un sommet privilégié : leur racine, elle-même accessible à travers un identificateur. Pour toute procédure qui a besoin de travailler sur toutes les primitives qui constituent un objet, un parcours en profondeur permet d'atteindre tous les objets primitifs.

Les nœuds du graphe sont de trois types :

- primitives géométriques,
- transformations géométriques,
- opérateurs booléens.

Enfin, chaque nœud sait s'il est directement pointé par l'intermédiaire d'un identificateur ; cela permet de documenter pour l'utilisateur le chemin dans le graphe qui aboutit à une primitive géométrique, par exemple :

```
(voiture).U2(roue)
```

désigne dans l'objet *voiture* le deuxième élément de la réunion (*U2*) qui est une *roue*.

2.7.3.3 La syntaxe

La syntaxe est très proche de la description interne. Un caractère préfixe les identificateurs de nœuds :

! introduit un objet primitif, par exemple *!cu* va définir un cube, *!sp* une sphère,...

@ introduit une transformation géométrique, *@t* désigne une translation, *@r* une rotation,...

$ introduit un opérateur booléen, *$U* indique une réunion, *$I* une intersection,...

La structure décrite au paragraphe précédent permet de gérer un nombre quelconque de nœuds avant d'arriver aux objets primitifs, on peut donc utiliser autant de transformations, de combinaisons et d'appels à des objets déjà définis que nécessaire. Cette structure arborescente est reproduite dans le fichier texte grâce à l'utilisation de parenthèses qui permettent de regrouper un niveau de l'arbre. Les différents composants d'un même niveau sont séparés par des virgules. On peut voir ces possibilités dans les définitions récursives qui suivent :

```
<définition> ::= <identificateur d'objet> = <objet complexe> ;

<objet complexe> ::= <identificateur d'objet> |
                     <objet primitif> |
                     <transformation> <objet complexe> |
                     <union> |
                     <inter> |
                     <diff>

<union> ::= $U ( <objet complexe> { , <objet complexe> }* )
<inter> ::= $I ( <objet complexe> { , <objet complexe> }* )
<diff> ::= $D ( <objet complexe> , <objet complexe> )
```

Les objets nécessitent trois paramètres de couleur et les transformations géométriques ont besoin de paramètres numériques pour être complètement définies. Tous ces paramètres peuvent être soit des nombres réels au format *%lf* du langage C, soit des identificateurs, soit des expressions combinant les éléments précédents. *Castor* possède en effet une table d'identificateurs de paramètres numériques ainsi qu'un évaluateur d'expressions algébriques. Notons que lexicalement un idenficateur de paramètre numérique commence par une lettre majuscule alors qu'un identificateur d'objet commence par une lettre minuscule. Cette distinction des identificateurs au niveau du premier caractère permet de simplifier l'analyse sémantique sans induire de grosses charges pour l'utilisateur.

Voici la syntaxe des objets primitifs tridimensionnels :

```
<objet primitif> ::= !co <aspect> |
                     !cu <aspect> |
                     !cy <aspect> |
                     !sp <aspect> |
                     !pc <ensemble 2d> <aspect> |
                     !pr <ensemble 2d> <aspect> |
```

Sémantiquement ces chaînes de caractères représentent respectivement les objets suivants :

- le cône de révolution
  $\{(x,y,z) \in R^3 ; x^2 + y^2 \leq 1 - z \text{ et } z \geq 0 \}$,
- le cube
  $\{(x,y,z) \in R^3 ; 0 \leq x \text{ et } x \leq 1 \text{ et } 0 \leq y \text{ et } y \leq 1 \text{ et } 0 \leq z \text{ et } z \leq 1 \}$,
- le cylindre de révolution
  $\{(x,y,z) \in R^3 ; x^2 + y^2 \leq 1 \text{ et } 0 \leq z \text{ et } z \leq 1 \}$,
- la sphère $\{(x,y,z) \in R^3 ; x^2 + y^2 + z^2 \leq 1\}$.
- le cône à base polygonale, d'origine le point de coordonnées (0,0,1) ; l'ensemble de points bidimensionnels étant la liste des sommets d'un polygone convexe dans le plan Oxy, polygone qui définit la base du cône,
- le cylindre à base polygonale ou prisme, de hauteur 1, l'ensemble de points bidimensionnels donnant la liste des sommets d'un polygone convexe dans le plan Oxy, polygone qui définit la base du prisme,

Voici maintenant la syntaxe des transformations :

```
<transformation> ::= <transformation affine> | <déformation>
<transformation affine> ::=
    @h ( <expression> ) |
    @a ( <expression> , <expression> , <expression> ) |
    @r ( <expression> , <expression> , <expression> , <expression> ) |
    @s ( <expression> , <expression> , <expression> , <expression> ) |
    @t ( <expression> , <expression> , <expression> )

<déformation> ::=
    @d ( <contrôle départ> <contrôle arrivée> )
<contrôle départ> ::= <ensemble 3d>
<contrôle arrivée> ::= <ensemble 3d>
```

Ces chaînes représentent respectivement les transformations suivantes :

- l'homothétie, et l'expression donne son rapport,
- l'affinité, et les expressions donnent ses coefficients,
- la rotation, et les trois premières expressions donnent les composantes d'un vecteur qui définit un axe passant par l'origine autour duquel s'effectue la rotation, la quatrième expression définit une mesure en degrés de l'angle de la rotation,
- la symétrie plane, les quatre expressions représentent les coefficients a, b, c, d, d'une équation (sous la forme ax+by+cz+d=0) du plan par rapport auquel s'effectue la symétrie,
- la translation, les trois expressions définissent les composantes du vecteur de translation,
- la *déformation* au sens donné dans [SEDE 86], (voir [DUCH 87]).

Il nous reste à donner la syntaxe des éléments de bas niveau. Hormis l'aspect, ils sont classiques dans les langages de programmation pour la définition des identificateurs, des constantes numériques et des expressions algébriques :

```
<aspect> ::= ( <rouge> , <vert> , <bleu> , <numéro> )
<rouge>  ::= <expression>
<vert>   ::= <expression>
<bleu>   ::= <expression>
<numéro> ::= <expression>
```

```
<identificateur d'objet> ::=
                    <lettre majuscule> { <caractère alphanumérique> }*
<identificateur de paramètre> ::=
                    <lettre minuscule> { <caractère alphanumérique> }*
<caractère alphanumérique> ::= <lettre> | _ | <chiffre>
<lettre> ::= <lettre majuscule> | <lettre minuscule>
<lettre majuscule> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<lettre minuscule> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<chiffre> ::= 0|1|2|3|4|5|6|7|8|9


<séquence de chiffres> ::= <chiffre> { <chiffre> }*
<entier non signé> ::= <séquence de chiffres>
<réel non signé> ::= <entier non signé> . <séquence de chiffres> |
    <entier non signé> . <séquence de chiffres> E <exposant> |
    <entier non signé> E <exposant>
<exposant> ::= <entier non signé> | <signe> <entier non signé>
<signe> ::= + | -
<nombre non signé> ::= <entier non signé> | <réel non signé>


<facteur> ::= <identificateur de paramètre> |
    ( <expression> ) |
    <fonction> ( <expression> ) |
    <nombre non signé>
<terme> ::= <facteur> | <terme> <opérateur multiplicatif> <terme>
<expression> ::= <terme> | <expression> <opérateur additif> <terme> |
    <signe> <terme>
<opérateur multiplicatif> ::= * | /
<opérateur additif> ::= + | -
<fonction> ::= sin | cos | tan | asin | acos | atan | log | ln | exp |
    floor | ceil | abs
```

Nous terminerons avec la syntaxe générale. Un fichier doit contenir outre la définition des paramètres de "caméra" des définitions de paramètres numériques, d'objets et des commandes de facettisation :

```
<fichier> ::= <définition de paramètre caméra>
            | <définition d'objet>
            | <définition de paramètre>
            | <facettisation>

<définition d'objet> ::= <identificateur d'objet> = <objet complexe> ;
<définition de paramètre> ::= <identificateur de paramètre> = <expression> ;
<facettisation> ::= '>' <identificateur d'objet> ;
<définition de paramètre caméra> ::=
            % <identificateur> ( [ <expression> { , <expression> }* ] ) ;
```

2.7.3.4 La sémantique
Il reste quelques points de sémantique à préciser, ce sont ceux qui ne sont pas évidemment liés à la syntaxe. La définition de tout élément, que ce soit un paramètre ou un objet, ne peut être effectuée qu'une seule fois. Ainsi la séquence suivante est illicite :

```
X = 3;
c = @h(X) !cu(1,0,0,0);
X = 4;
```

Cette convention évite les questions qu'on pourrait se poser quant à la valeur de l'objet *c*.

De plus, seule des références aux objets ou aux paramètres précédemment définis sont autorisées. Cela signifie que supprimer la ligne :

```
X = 3;
```

dans la séquence précédente ne la rend pas valide. Ce choix permet d'analyser le fichier de description en une seule passe ; ce qui est en accord avec le choix que nous avons fait pour l'usage dans l'environnement UNIX de ces fichiers de description : ils doivent pouvoir circuler dans des tubes.

Le *numéro* qui apparaît en quatrième paramètre d'aspect n'est pas utilisé pour l'instant ; il devrait permettre dans le futur d'associer à chaque objet primitif une formule de calcul d'éclairement et donc de définir l'interaction des objets avec la lumière.

Les paramètres définissant la caméra ne sont pas interprétés par *castor*, les expressions sont simplement évaluées et leurs valeurs écrites sur la sortie standard. Ces paramètres doivent être donnés avant la première commande de facettisation ; ainsi, les programmes de visualisation recevant le flot de sortie de *castor* connaissent toutes les informations définissant la caméra avant de recevoir le premier polygone.

2.7.3.5 L'implantation et l'environnement
Nous emploierons dans cette section le vocabulaire du système d'exploitation UNIX : filtre, entrée standard, sortie standard, tube,... Pour les définitions de ces termes on pourra se reporter à [BOUR 83].

Le programme *castor* fonctionne comme un filtre UNIX. Cela signifie qu'il reçoit sur son entrée standard un flot de caractères et qu'il génère sur sa sortie standard un autre flot de caractères. Le flot d'entrée doit contenir une description des différents objets constituants la scène. La forme de cette description doit respecter la syntaxe que nous avons décrite. Ce flot d'entrée doit aussi contenir une ou plusieurs commandes de facettisations signifiées grâce au caractère '>'. Cette commande appelle la fonction de facettisation. Cette dernière fonction assure la facettisation des primitives en tenant compte des transformations affines. *castor* écrit donc sur sa sortie standard le développement de la structure arborescente associé à chaque objet ainsi que la liste des polygones approximant chaque primitive.

L'utilisation normale de *castor* est donc de créer un fichier de description grâce à un éditeur de texte puis de rediriger l'entrée standard de *castor* depuis ce fichier. Il est aussi possible de passer ce fichier à travers d'autres filtres UNIX avant de le passer à *castor*. C'est à travers cette possibilité que réside la plus grande souplesse de ce langage : on bénéficie de tous les utilitaires du système UNIX pour enrichir le langage.

Le plus simple des filtres à utiliser est le préprocesseur du langage C : */lib/cpp*. Grâce à lui l'utilisateur peut modulariser la description d'une scène complexe en la séparant en plusieurs fichiers puis en réunissant ces différents fichiers avec des directives *#include*. Il peut aussi s'affranchir des caractères $, !, @ en utilisant des macro-définitions, par exemple :

```
#define union $U
#define inter $I
#define diff  $D
```

Des macro-définitions plus complexes permettent de définir des classes d'objets paramétrés qui doivent apparaître plusieurs fois avec des valeurs de paramètres différentes. Voici un exemple d'une telle définition où on décrit une huisserie de fenêtre.

```
#define MENUISERIE_EXT(X1,X2,Y1,Y2) \
@t(0,0,-E_me/2) $U( \
  @t(X1,Y1,0) @a(X2-X1,L_me,E_me) !cu(R_alu,V_alu,B_alu,T_alu), \
  @t(X2-L_me,Y1+L_me,0) @a(L_me,Y2-Y1-2*L_me,E_me) !cu(R_alu,V_alu,B_alu,T_alu), \
  @t(X1,Y2-L_me,0) @a(X2-X1,L_me,E_me) !cu(R_alu,V_alu,B_alu,T_alu), \
  @t(X1,Y1+L_me,0) @a(L_me,Y2-Y1-2*L_me,E_me) !cu(R_alu,V_alu,B_alu,T_alu) \
  )
```

Cette macro-définition définit un cadre formé de quatre "blocs". Ce cadre est symétrique par rapport au plan des abscisses et des ordonnées. Il a une épaisseur de *E_me* selon l'axe des côtes, et les blocs ont une largeur de *L_me* dans le plan des abscisses et des ordonnées. La figure 2.11 permet d'observer la signification des paramètres *X1*, *X2*, *Y1* et *Y2*.

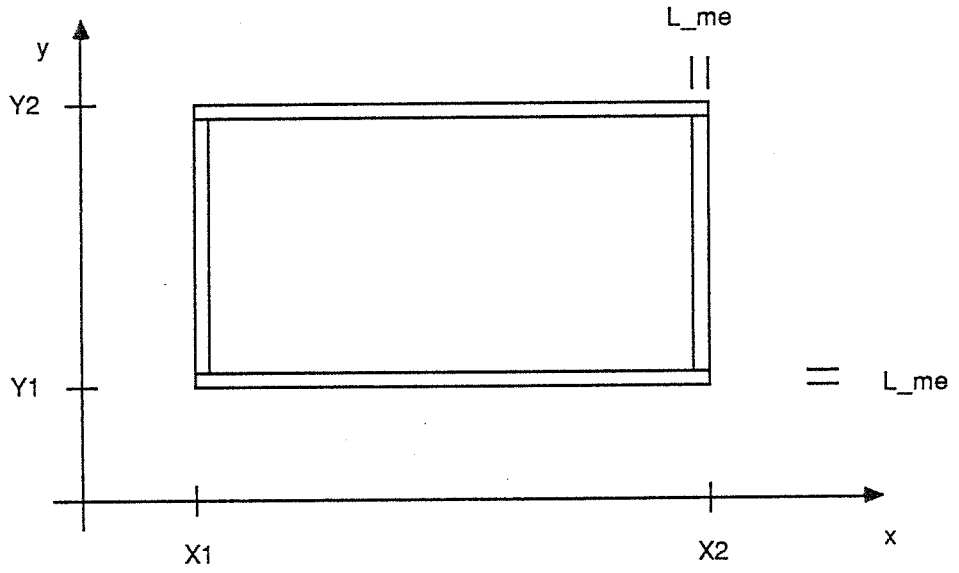


figure 2.11

Nous avons également utilisé un filtre générant un fichier dans le langage **CASTOR** à partir d'une chaîne composée de 0, de 1 et de couples de crochets. Ce filtre permet de définir une géométrie d'arbre à partir de chaînes elles-mêmes générées par des grammaires. Cet exemple sera développé dans le troisième chapitre de ce rapport.

2.7.3.6 Exemples d'applications
Ce langage a été utilisé pour définir la géométrie de plusieurs projets architecturaux. Le premier d'entre eux et aussi le plus simple a été la modélisation d'une variation architecturale autour d'une passerelle. Cette variation a été créée par Frank Chopin (architecte DPLG). La modélisation a nécessité environ une journée de travail en se servant des plans déjà dessinés. Le fichier comporte une centaine de lignes. La photo 2.1 montre l'une des vues réalisées d'après cette modélisation. La visualisation a été faite avec la méthode du "gz-buffer" que nous décrirons dans le quatrième chapitre de ce rapport.

Le deuxième des projets architecturaux modélisés a été celui concernant l'aménagement du chemin départemental 518 dans le département du Rhône. Le projet a été présenté par le groupe de plasticiens et d'architectes stéphanois **Lieu Dit**. Ce projet plus important a nécessité environ trois jours de travail pour la modélisation. La photo 2.2 montre une vue réalisée à partir de cette description. Le ciel a été fabriqué par une méthode fractale, il est du à Sabine Coquillart [COQU 84b].

Le troisième projet est le plus grand au niveau de la taille de la base de données, et il a nécessité plus d'une semaine de travail. Cette fois, toute l'architecture d'une construction est modélisée. Il s'agit d'un projet de restauration d'une maison ancienne (projet réalisé par l'atelier d'architecture **Frank Chopin et Philippe Merle**). A partir de cette modélisation, nous avons réalisé une animation où la caméra se déplace de l'extérieur vers l'intérieur de la construction.

Pour réaliser de telles animations, nous utilisons les possibilités de calcul du *shell* de UNIX combiné avec la possibilité qu'offre le préprocesseur */lib/cpp* de définir des macros sur la ligne de commande lançant son éxécution. Plus précisemment, nous exprimons à l'intérieur du fichier de description les paramètres de la caméra en fonction d'une variable *Time*. La valeur de cette variable étant elle-même définie au travers d'une macro de */lib/cpp* : TIME. Par exemple, pour un déplacement linéaire entre deux points de coordonnées (X1,Y1,Z1) et (X2,Y2,Z2), nous écrivons :

```
Time = TIME;
%eye ( X1+Time*(X2-X1), Y1+Time*(Y2-Y1), Z1+Time*(Z2-Z1));
```

dans le fichier de description. Sur la ligne de commande de */lib/cpp*, on indique la valeur de TIME de la façon suivante :

```
/lib/cpp -DTIME=0.35 ...
```

Cette ligne donnant le même résultat que si l'on avait écrit dans le fichier

```
#define TIME 0.35
```

On peut ainsi donner des valeurs numériques à l'extérieur du fichier. Les photos 2.3, 2.4, 2.5, 2.6 montrent des vues extraites de l'animation réalisée avec le fichier de description de la maison restaurée.

2.7.3.7 Limitations du langage

Au niveau purement langage, il manque dans **CASTOR** une réelle notion de variable et les structures de contrôle qui en feraient un véritable langage de programmation. Cet obstacle peut être contourné par l'écriture de sur-ensembles de langages existants qui y ajoutent les notions connues par *castor*, à savoir : opérations booléennes, transformations affines et primitives. Dans l'équipe, trois implantations ont étés réalisées dans cet esprit sur les langages Lisp, C et C++. Cela est possible grâce à l'aspect langage du système dès sa conception. Les autres conséquences de cet aspect sont, comme prévues, sa portabilité et la précision que l'on peut obtenir dans la description des scènes. Le point négatif concerne l'interaction du système.

Chapitre 3

MODELISATION DEDIEE

3.1 INTRODUCTION

Nous présentons dans ce chapitre les modélisations utilisées pour les terrains et pour la végétation. Nous détaillons plus particulièrement une modélisation procédurale d'arbre qui utilise le langage de représentation de solides que nous avons décrit dans le chapitre précédent. Ce modèle produisant une suite cohérente dans le temps des étapes de la pousse d'un arbre, nous avons pu réaliser une animation. Notons enfin que toute modélisation procédurale pourrait être implantée dans notre système par une technique analogue à celle que nous avons utilisée pour les arbres.

3.2 TERRAINS

Pour les applications de synthèse d'images, les terrains sont le plus souvent modélisés par leur surface. En pratique, ces surfaces sont elles-mêmes représentées selon deux méthodes :

- la première donne les altitudes aux nœuds d'une grille régulière,
- la seconde se base sur un réseau de triangles de formes et de tailles différentes.

3.2.1 Représentation au moyen d'une grille régulière

Les grilles de base de la première de ces deux représentations peuvent être de plusieurs sortes, nous pouvons citer les maillages carrés, rectangulaires, triangulaires ou hexagonaux [PETR 87]. Cette représentation se prête bien à une collecte automatique des données. Sa simplicité n'est cependant pas sans inconvénients : les points de la grille ne coïncident pas nécessairement avec les points caractéristiques du terrain comme les points situés sur des lignes de crête ou au contraire au creux des vallées.

En fait, une telle représentation consiste à *échantillonner* la surface. Intuitivement, la fréquence d'échantillonnage correspond à la densité des points de la grille. Le théorème de Shannon se traduit donc en disant que la distance entre deux points de la grille doit être plus petite que la moitié de la longueur de la plus petite déformation du terrain que l'on veut représenter. Cependant, pour que ce théorème et ses conséquences puissent être appliqués, il faudrait pouvoir *filtrer* le terrain, de façon à en ôter les *fréquences* supérieures à la moitié de la fréquence d'échantillonnage avant l'échantillonnage, et de même filtrer le signal de la grille pour retrouver la surface du terrain. Ces deux opérations me semblent irréalisables,

bien que la deuxième ait un sens théorique. La théorie de l'échantillonnage est donc difficile à utiliser. Les défauts engendrés par l'absence de ces filtrages sont toutefois nettement visibles comme on peut s'en rendre compte sur la photo 3.1 où les marches de la carrière introduisent des changements très brutaux d'altitude, et donc des fréquences très élevées. Intuitivement, la densité des points de la grille doit être élevée pour pouvoir représenter avec suffisamment de précision les parties accidentées de la région modélisée. La grille étant régulière, ceci va conduire à avoir une grande redondance des données dans les parties plus calmes de la région. Cette redondance est inutile, voire encombrante. Les photogrammétristes ont proposé un *échantillonnage progressif* où la densité des points est plus élevée dans les régions accidentées (cf. figure 3.1).

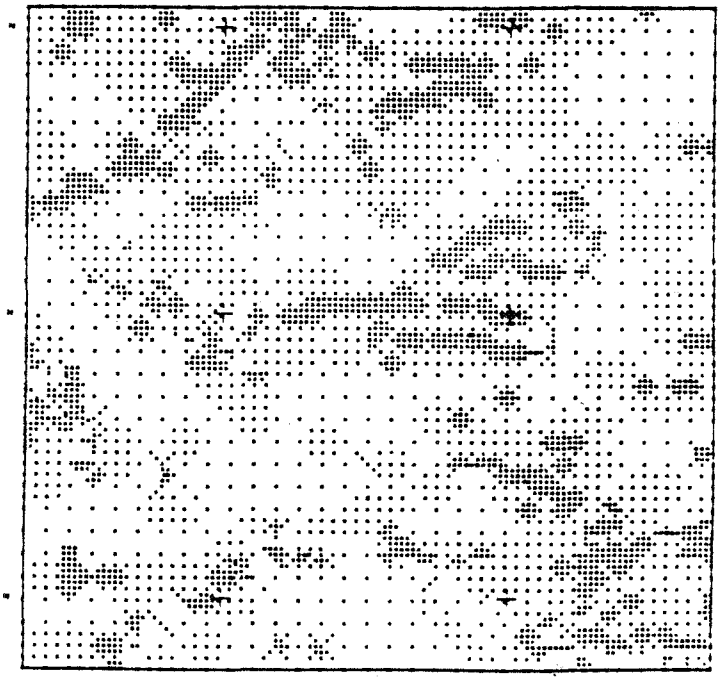


figure 3.1

Il faut enfin signaler que plusieurs méthodes ont été développées pour construire une représentation de terrain par grille altimétrique à partir de points aléatoirement distribués. On peut trouver des références sur ce sujet dans [PETR 87].

Le problème du filtrage après échantillonnage est en fait un problème d'interprétation de la représentation, la formulation complète du problème étant de savoir quelle est la surface *modélisée* par la représentation. Cette question a des réponses différentes selon l'utilisation qui est faite de la modélisation numérique du terrain. Pour notre part, nous ne considérons que l'application *synthèse d'images*. De plus, nous ne considérons que le cas des mailles rectangulaires puisque les deux seuls fichiers que nous possédons sont de ce type. La question a été traitée dans [HUGL 80] pour un maillage carré. Les différents modèles étudiés dans cette publication pour une visualisation du terrain en perspective sont les suivantes :

1. division des mailles carrées en deux triangles selon la première diagonale,
2. division des mailles carrées en deux triangles selon la deuxième diagonale,
3. division de chaque maille carrée en quatre triangles partageant un sommet qui est l'isobarycentre des quatre sommets de la maille,
4. traitement de la maille carré comme un polygone non plan.

Dans ce dernier cas, il utilise malgré tout une interpolation linéaire. La forme géométrique de la maille n'est alors définie que lors du remplissage de la projection de celle-ci. La géométrie de la maille est donc dépendante du point de vue. Cela ne pourrait avoir de conséquences facheuses que pour une scène comportant simultanément un terrain et des objets qui l'intersectent. Nous pensons également que des artefacts pourraient apparaître dans le cas d'une animation. Par ailleurs, il étudie quatre ombrages sur ces différentes géométries :

1. ombrage uniforme pour les géométries 1, 2, 3 et 4,
2. ombrage de Gouraud pour la géométrie 4,
3. ombrage mélange d'ombrage de Gouraud et d'ombrage uniforme pour la géométrie 4,
4. ombrage de Phong pour la géométrie 4.

L'ombrage uniforme est comme on peut le prévoir insuffisant. Parmi les trois autres il a une préférence pour l'ombrage 3 parce que la discontinuité d'éclairement entre deux mailles voisines (due à l'influence de l'ombrage uniforme), laisse le découpage apparent, ce qui *<<accentue l'effet de la vision perspective d'une part et d'autre part, sert à donner une échelle à la représentation du terrain>>* (extrait de [HUGL 80]). Il note de plus que pour les terrains l'ombrage de Phong, beaucoup plus coûteux que celui de Gouraud, n'est pas utile puisqu'il est utilisé en synthèse d'images essentiellement parce qu'il permet d'obtenir des réflexions spéculaires, et que celles-ci sont quasiment inexistantes sur un terrain naturel.

En ce qui nous concerne nous utilisons

- des facettes triangulaires (pour éliminer le problème d'une géométrie dépendant du point de vue puisque nous voulons visualiser simultanément objets du sur-sol et terrain),
- un ombrage de Gouraud pour le terrain (puisque cette méthode reste peu coûteuse et ne laisse pas apparaître de défauts).

3.2.2 Représentation par mosaïque de triangles

Ce type de représentation, s'il perd la régularité de l'autre méthode permet néanmoins d'incorporer facilement les points et les lignes caractéristiques du terrain modélisé.

La fabrication de la mosaïque de triangles à partir d'un ensemble de points aléatoirement distribués a fait l'objet de nombreuses études, car outre le fait qu'il fallait réduire la complexité des premiers algorithmes pour avoir des temps d'exécution raisonnables, il fallait aussi pouvoir assurer que le résultat obtenu était indépendant du point à partir duquel commençait la triangulation. Là encore [PETR 87]

donne de nombreuses références et décrit les principales méthodes, en particulier la méthode de triangulation de Delaunay.

Ne disposant pas de fichiers de description de terrains utilisant cette méthode, nous ne l'avons pas étudié à fond. [SCHA 83] permet de se rendre compte que les triangles sont visualisés tels quels et qu'ils ne représentent pas une surface de degré plus élevé.

3.3 VEGETATION

Deux types de travaux ont été effectués pour modéliser les éléments végétaux d'un paysage. Ceux-ci correspondent en fait à deux niveaux de détails de la modélisation. Le premier de ces deux niveaux convient lorsque ces éléments sont vus de loin comme par exemple le cas d'une forêt, d'un champ,... vus d'avion, ou encore le cas d'un arbre dans le lointain d'un paysage. Dans ce cas un modèle géométrique très simple (polygone ou quadrique) associé à une texture suffit à modéliser ces objets. Par contre si l'on veut réaliser une vue depuis l'*intérieur* de la forêt, ou si l'on veut modéliser une scène avec une maison et quelques arbres dans laquelle aussi bien les arbres cachent en partie la maison que l'inverse, une modélisation géométrique aussi simple n'est pas suffisante. Comme par ailleurs une modélisation manuelle de tous les détails d'un arbre ou d'une forêt n'est pas envisageable, les modélisations procédurales se sont imposées. Nous allons donner un aperçu sur ces deux techniques dans les deux paragraphes suivants.

3.3.1 Modèles basés sur les textures

Il y a deux utilisations de cette technique suivant l'objet auquel on applique la texture :

- le terrain lui-même (cas d'un champ pour une visualisation aérienne par exemple),
- d'autres objets géométriques disposés sur le terrain (cas des arbres isolés par exemple).

Pour le premier cas, il y a deux problèmes à résoudre, à savoir d'une part comment définir les différentes textures, et d'autre part comment définir les zones où interviennent chacunes de ces textures.

Définition des textures
D'après [GHAZ 85], on peut classer les méthodes de génération de textures en quatre groupes :

- les méthodes fractales,
- les méthodes structurelles,
- les méthodes stochastiques,
- les méthodes spectrales.

A notre connaissance, seules les méthodes spectrales ont été utilisées pour la modélisation des textures à appliquer sur des terrains. Le point de départ des travaux de ce type est [SCHA 80] dont le modèle de base a la forme suivante :

$$T(u,v) = A_0 + \sum_{i=1}^{i=n} A_i \cos(2\pi i f_i u + 2\pi i g_i v + \phi_i)$$

où A_0 désigne la valeur moyenne de la fonction T, A_i l'amplitude, f_i (respectivement g_i) la fréquence suivant u (respectivement v), et ϕ_i la phase de la i-ème des n composantes.

[GARD 84] présente un autre modèle basé lui aussi sur des sinusoïdes :

$$T(u,v) = \sum_{i=1}^{i=n} A_i.(\sin(2\pi f_i u + \phi_i) + 1)/2 \; . \sum_{i=1}^{i=n} A_i.(\sin(2\pi f_i v + \psi_i) + 1)/2$$

Ici, une modulation de phase permet d'éviter trop de régularité dans la fonction obtenue, celle-ci est obtenue grâce aux fonctions ϕ_i et ψ_i. Il propose d'utiliser pour ϕ_i (respectivement ψ_i) une fonction sinusoïdale de v (respectivement de u).

Définition des zones d'applications
Sur ce sujet [SCHA 83] est notre seule source de renseignements. Il ne mentionne que deux méthodes pour définir ces zones, et toutes deux considèrent le cas où l'altimétrie a été définie par une mosaïque de triangles :

- La première méthode consiste à considérer d'une part le réseau de polygones définissant les différentes zones de culture, et d'autre part le réseau de polygones définissant l'altimétrie, puis à chercher tous les polygones résultant du recouvrement de ces deux réseaux, et enfin à trianguler les polygones obtenus. Ainsi le résultat est un réseau de triangles dont l'intérieur ne contient qu'une seule texture et dont on connaît l'altitude à chaque sommet.

- La deuxième méthode consiste lors de la construction de la base de données à donner pour certains points à l'intérieur de chaque triangle (ces points sont les *noyaux*) le type de texture associée à ce point. Cet ensemble de points à l'intérieur du triangle permet de partitionner le triangle en zones, chacune de ces zones étant l'ensemble des points du triangle plus proches de l'un des noyaux que des autres noyaux. Ainsi la texture de chaque point du triangle est celle du noyau dont il est le plus proche.

Ce problème de représentation des zones de texture est un problème de représentation d'un plan (au sens architectural du terme). Aussi il nous semble que l'on pourrait utiliser la structure de données *carte planaire* développée par Dominique Michelucci et Michel Gangnet [MICH 84] pour pouvoir définir des plans d'architectures.

Arbres définis grâce à des textures
Lorqu'il ne s'agit plus seulement d'appliquer une texture sur la géométrie du terrain pour donner des apparences de cultures, une nouvelle géométrie doit être définie. C'est ce que fait [GARD 84] en utilisant des ellipsoïdes. Grâce à une paramétrisation avec deux variables de ces ellipsoïdes, il peut associer à chaque point de la surface de ces objets une valeur de texture. Cette valeur ne sert pas seulement à définir une couleur mais elle sert aussi de paramètre de transparence. Cela revient à dire que la texture influe sur la géométrie de l'objet ; ainsi celui-ci n'a plus une frontière nette qui nuirait à la représentation d'arbres dont la forme géométrique est irrégulière.

[GARD 84] présente ce modèle avec une méthode d'élimination des parties cachées originale pour une scène modélisée par des ellipsoïdes. Avec Jacqueline Argence [ARGE 85], nous avons réutilisé ce modèle pour l'intégrer dans une méthode de visualisation par z-buffer (cf. photo 3.2). [GARD 85] utilise les mêmes fonctions de textures pour réaliser des effets de nuages en les appliquant aussi bien sur des plans que sur des ellipsoïdes, et toujours en utilisant le principe de transparence pour obtenir des silhouettes irrégulières.

3.3.2 Modèles procéduraux

3.3.2.1 Les méthodes existantes

Nous arrivons maintenant aux modélisations dont la géométrie tridimensionnelle est beaucoup plus détaillée. Il y a dans ces modélisations trois parties toutes trois aussi importantes dans l'obtention d'un bon résultat. La première est la définition de la structure topologique de l'arbre, c'est-à-dire le nombre de branches sur le tronc, le nombre de rameaux sur chaque branche,... La deuxième définit les longueurs entre les différents embranchements et les angles de départ de chaque branche par rapport à celle qui la supporte. Il s'agit donc dans ces deux premières étapes de définir le *squelette* de l'arbre. La troisième étape consiste à habiller ce squelette de façon à lui donner du volume.

[MARS 80] est la première implantation d'une telle méthode ; cependant les trois étapes y sont confondues et la modélisation utilisée n'est pas récursive : il n'y a qu'un seul niveau. Les paramètres de contrôle sont les suivants :

- le nombre de feuilles,
- la longueur de chaque branche,
- la description d'une feuille,
- la couleur,
- la position de l'arbre,
- la taille de la feuille,
- la distance entre les branches,
- la distance entre les feuilles,
- une graine pour un générateur aléatoire.

Les objets tridimensionnels utilisés pour la modélisation des branches et du tronc ne sont pas précisés dans l'article.

Plus récemment, d'autres méthodes utilisant des algorithmes plus élaborés (et aussi plus récursifs) ont été conçues. [SMIT 84] présente une méthode basée sur l'utilisation d'une grammaire et donc la génération d'un langage pour réaliser la première étape. Il ne présente que des langages utilisant l'alphabet { 0, 1, [,] }.

Nous allons donner deux exemples de règles citées dans [SMIT 84]. Dans le premier ensemble de règles, le contexte d'un caractère n'intervient pas sur la chaîne qui va remplacer ce caractère :

$0 \rightarrow 1[0]1[0]0$
$1 \rightarrow 11$
$[\rightarrow [$
$] \rightarrow]$

Les trois premières générations de chaînes engendrées par ces règles à partir de l'axiome 0 sont les suivantes :

```
1[0]1[0]0

11[1[0]1[0]0]11[1[0]1[0]0]1[0]1[0]0

1111[11[1[0]1[0]0]11[1[0]1[0]0]1[0]1[0]0]1111[11[1[0]1[0]0]
11[1[0]1[0]0]1[0]1[0]0]11[1[0]1[0]0]11[1[0]1[0]0]1[0]1[0]0
```

Voici un autre ensemble de règles. Celles-ci sont dépendantes du contexte, les petits chiffres définissent le contexte, seul le chiffre de taille normale est remplacé par la chaîne figurant à droite de la flèche :

$_0 0 _0 \rightarrow 0$
$_0 0 _1 \rightarrow 1[1]$
$_0 1 _0 \rightarrow 1$
$_0 1 _1 \rightarrow 1$
$_1 0 _0 \rightarrow 0$
$_1 0 _1 \rightarrow 11$
$_1 1 _0 \rightarrow 1$
$_1 1 _1 \rightarrow 0$

Et voici les quinze premières générations engendrées par l'axiome 0 :

```
11
00
01[1]
111[0]
000[11]
001[1][10]
01[1]1[0][111]
111[0]0[11][000]
001[11]11[10][001[1]]
01[1]1[00]00[111][01[1]1[0]]
111[0]1[01[1]]01[1][100][1[1]1[0]0[11]]
000[11]1[111[0]]111[0][101[1]][0[0]1[11]11[10]]
001[1][10]1[000[11]]000[11][1111[0]][11[1[1]]1[00]00[111]]
01[1]1[0][111]1[001[1][10]]001[1][10][1000[11]][10[0[0]]1[0
1[1]]01[1][100]]
111[0]0[11][000]1[01[1]1[0][111]]01[1]1[0][111][1001[1][10]
][111[1[1][1[1]]]1[111[0]]111[0][101[1]]]
```

On peut constater sur cet exemple qu'il est très difficile de prévoir quels seront les résultats obtenus à partir de règles, même si elles sont très simples, dès que le contexte intervient. De plus [SMIT 84] ne présente aucune méthode pour les deuxième et troisième étapes.

[AONO 84] propose un modèle récursif où chaque branche se divise en deux rameaux à son extrémité. Les paramètres de contrôle sont les suivants :

- le niveau d'arrêt de la récursion,
- le rapport de contraction de la longueur des branches par rapport à la longueur de celle qui les supporte,
- les angles d'embranchement des deux rameaux par rapport à la direction de la branche qui les supporte,
- le tronc, c'est-à-dire le segment qui engendre (et qui supporte!...) toute la structure.

On voit que dans ce modèle [AONO 84] s'intéresse surtout à la deuxième étape, puisqu'au niveau topologique, il génère un arbre binaire et que par ailleurs il ne parle pas d'un habillage tridimensionnel. Il présente toutefois d'autres modèles topologiques mais le choix reste très limité et inscrit dans le corps du programme, contrairement à [SMIT 84] où l'utilisateur peut engendrer de nouvelles topologies grâce aux grammaires.

A notre avis, un des défauts majeurs des modélisations précédentes vient de ce qu'ils ne tiennent pas compte de l'évolution naturelle de l'arbre, c'est-à-dire de la façon dont il pousse. En effet, augmenter le niveau de récursivité dans [AONO 84], ou augmenter le nombre de générations dans [SMIT 84] ne donnera pas un arbre plus âgé ou plus fourni, mais risque plutôt (surtout pour [SMIT 84]) de transformer complètement la structure obtenue. Ainsi, si, comme cela est suggéré par [SMIT 84] on calcule la position angulaire des branches en fonction de leur position verticale sur le tronc, le résultat sera de faire tourner les branches, puisque de nouvelles branches peuvent apparaître en dessous d'une qui existe déjà.

[LIEN 87] apporte une solution à ce problème en se rapprochant beaucoup plus d'une simulation de la réalité de la croissance des végétaux. Pour cela, il utilise les résultats des travaux de botanistes. Les travaux présentés dans [LIEN 87] ne concernent toutefois que les feuilles mais ils peuvent très certainement se prolonger pour d'autres éléments végétaux.

Pour ce qui est de l'étape habillage, nous pouvons citer [KAWA 82] et [BLOO 85] pour leurs méthodes tout à fait complémentaires l'une de l'autre. [KAWA 82] n'utilise dans sa modélisation que des primitives solides *simples*, à savoir des cylindres et des cônes de révolution. Dans les grandes lignes, de telles géométries sont représentables grâce au langage CASTOR puisqu'il suffirait d'ajouter à ce langage une primitive permettant de décrire la partie assurant le raccord entre deux cylindres de rayons différents et inclinés l'un par rapport à l'autre pour avoir tout le matériel nécessaire à une modélisation des éléments utilisés par [KAWA 82]. Nous reproduisons sur la figure 3.2 la structure qu'il adopte pour un embranchement.

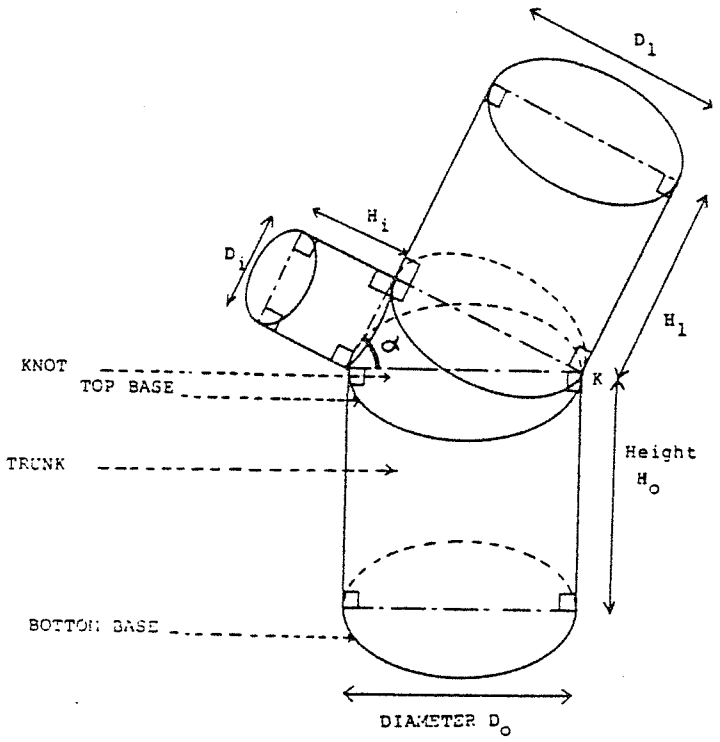


figure 3.2

L'utilisation de primitives solides aussi simple crée une géométrie trop anguleuse, aussi [BLOO 85] utilise de nombreuses fonctions splines dans sa modélisation. L'idée est de construire d'abord des courbes splines à partir du squelette pour y supprimer les angles. Ensuite, il déplace un disque de rayon variable le long de cette courbe en le laissant orthogonal à celle-ci, c'est donc une extrusion généralisée.

Si ces modélisations permettent toutes d'obtenir des résultats spectaculaires même s'ils ne respectent pas toujours la réalité végétale, c'est au prix de la création d'un très grand nombre d'objets susceptibles d'être affichés (polygones, quadriques,...). Ils sont donc difficilement utilisables pour définir un grand nombre d'arbres à représenter sur un terrain.

3.3.2.2 Notre implantation

Nous nous sommes inspirés des travaux de [SMIT 84] pour démarrer nos essais. Comme nous l'avons indiqué, il est très difficile de sentir intuitivement le lien entre une grammaire et les chaînes qui en seront obtenues. Ceci est dû à deux causes :

- d'une part, les éléments de l'alphabet ne sont pas associés à une sémantique dès la conception du langage,
- d'autre part, l'influence du contexte sur la chaîne de remplacement d'un caractère complique la compréhension de l'évolution des chaînes au cours des générations.

Pour pouvoir définir une sémantique associée aux symboles terminaux des grammaires utilisées, il nous fallait avoir quelques idées (éventuellement fausses) sur la façon dont les arbres poussent et quelles sont les différents "objets" qui les composent. Pour cela, nous avons regardé de près quelques spécimens. Cependant il ne nous a pas été possible de les étudier suffisamment longtemps pour pouvoir suivre l'évolution des embranchements. Nous nous sommes donc contentés d'inventer un modèle de pousse à partir des observations que nous pouvions faire sur des sujets d'une vingtaine d'années. Il va de soi qu'une démarche plus scientifique serait, comme [LIEN 87], de collaborer avec une équipe de botanistes et d'étudier les travaux de cette science : nous n'en avions ni les moyens, ni le temps.

Les éléments de modélisation les plus simples que j'ai trouvés s'appliquent au sapin :

- au sommet des plants quatre bourgeons, l'un deux partant verticalement dans le prolongement du tronc, les trois autres régulièrement disposés autour de ce dernier dans trois plans verticaux P_0, P_1 et P_2, écartés de 120° les uns des autres ;
- les trois branches situées un "étage" plus bas sont dans des plans verticaux P_0', P_1' et P_2' décalés d'environ 40° par rapport à P_0, P_1 et P_2 ; ce sont des branches simples poussées dans l'année ;
- les trois branches situées un autre "étage" plus bas sont aussi des branches simples et les plans verticaux qui les contiennent sont encore décalés de 40° par rapport aux plans P_0', P_1' et P_2'.
- à partir du troisième niveau, les branches portent des sous-branches, le niveau de récursion n'est cependant pas très grand et varie entre trois et quatre suivant les branches (on peut supposer que cela vient des conditions d'éclairement, ou autres...) ;
- ces branches s'étalent dans un plan horizontal, leurs extrémités portent trois bourgeons.

La figure 3.3 montre un exemple d'embranchement relevé sur une branche réelle de sapin.

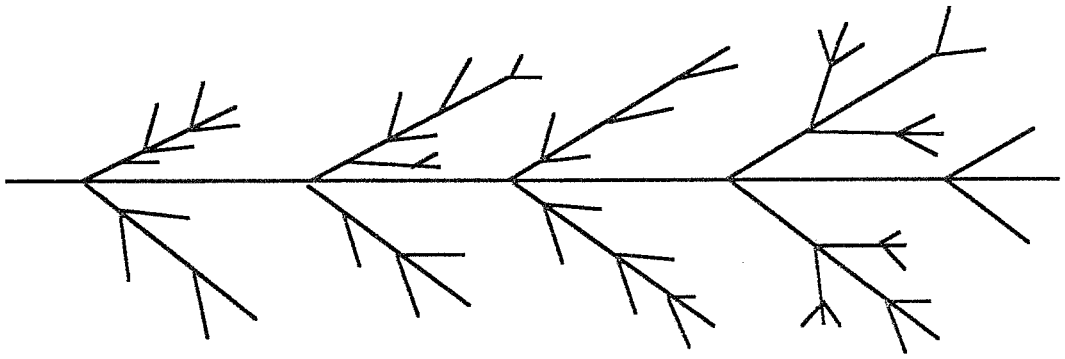


figure 3.3

Ainsi, nous pouvons déduire des observations précédentes des formules pour le calcul des angles d'embranchement. Tout d'abord, pour les branches prenant naissance sur le tronc, deux angles sont nécessaires (cf. figure 3.4) :

- le premier θ donne la rotation de la branche autour de l'axe du tronc ;
- le second ϕ donne l'écartement de l'axe de la branche par rapport à l'axe du tronc.

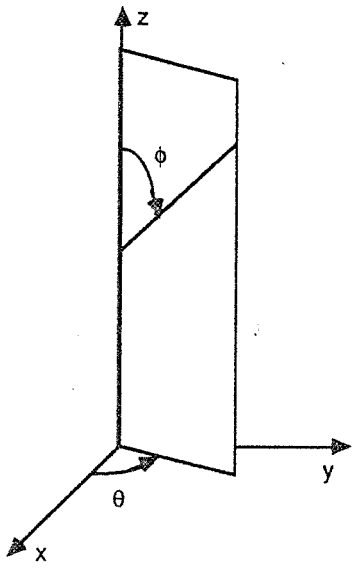


figure 3.4

Pour calculer la valeur de θ, nous avons besoin de deux informations sur la branche :

- la génération g à laquelle apparaît la branche,

- le numéro **n** de cette branche parmi celles qui apparaissent au même niveau du tronc à la même génération.

Ainsi, l'angle θ peut être calculé grâce à la formule :

$$\theta = \mathbf{g} \cdot \theta_0 + \mathbf{n} \cdot \theta_1$$

θ_1 est pris égal à 360° divisé par le nombre de branches qui naissent à un niveau de l'arbre. Cela signifie que les branches sont régulièrement disposées autour du tronc à un niveau donné.

L'angle ϕ peut être pris constant en première approximation, cependant on constate que, les branches s'alourdissant en vieillissant, cet angle augmente avec l'âge **a**. On peut donc utiliser la formule de calcul suivante :

$$\phi = \phi_0 + \mathbf{a} \cdot \phi_1$$

Pour les sous-branches qui naissent sur des branches, un seul angle est nécessaire puisque nous avons remarqué que les branches étaient planes. Cet angle α vaut α_0 ou $-\alpha_0$ suivant la valeur zéro ou un du numéro **n**.

Enfin, pour calculer le rayon r et la longueur l des branches, nous utilisons l'âge **a** de la branche avec une simple relation linéaire :

$$r = r_0 \cdot (\mathbf{a}+1)$$
$$l = l_0 \cdot (\mathbf{a}+1)$$

Ainsi, les trois variables **a**, **n** et **g** me semblent indispensables pour une modélisation simulant la pousse. On ne peut pas trouver les valeurs de ces variables en ne connaissant que la topologie d'embranchement, or avec les techniques décrites dans [SMIT 84], cette information topologique est la seule disponible lors de l'interprétation. Il faut donc que les valeurs de ces variables soient calculées en même temps que la génération des chaînes. Il faut enfin remarquer que l'âge d'une branche **a** est égale à la différence de l'âge de l'arbre **g**o et de la génération d'apparition de cette branche **g**.

Pour générer la topologie décrite ci-dessus, nous avons utilisé les règles de grammaire suivantes :

$0 \rightarrow 3(1)(1)(1)0$
$1 \rightarrow 3(2)(2)1$
$2 \rightarrow 3(3)(3)2$
$3 \rightarrow 3$

Sémantiquement, les caractères 0, 1, 2, 3 ont la signification suivante :

0 indique le morceau de tronc terminal,

1 les branches terminales de l'année situées à l'extrémité de branches naissant sur le tronc,

2 les branches terminales de l'année situées à l'extrémité de branches

naissant elles-mêmes sur d'autres branches ;

3 un morceau de tronc ou de branche qui n'engendrera plus rien.

A partir de l'axiome 0, les quatre premières générations sont les suivantes :

```
3(1)(1)(1)0

3(3(2)(2)1)(3(2)(2)1)(3(2)(2)1)3(1)(1)(1)0

3(3(3(3)(3)2)(3(3)(3)2)3(2)(2)1)(3(3(3)(3)2)(3(3)(3)2)3(2)(
2)1)(3(3(3)(3)2)(3(3)(3)2)3(2)(2)1)3(3(2)(2)1)(3(2)(2)1)(3(
2)(2)1)3(1)(1)(1)0

3(3(3(3)(3)3(3)(3)2)(3(3)(3)3(3)(3)2)3(3(3)(3)2)(3(3)(3)2)3
(2)(2)1)(3(3(3)(3)3(3)(3)2)(3(3)(3)3(3)(3)2)3(3(3)(3)2)(3(3
)(3)2)3(2)(2)1)(3(3(3)(3)3(3)(3)2)(3(3)(3)3(3)(3)2)3(3(3)(3
)2)(3(3)(3)2)3(2)(2)1)3(3(3(3)(3)2)(3(3)(3)2)3(2)(2)1)(3(3(
3)(3)2)(3(3)(3)2)3(2)(2)1)(3(3(3)(3)2)(3(3)(3)2)3(2)(2)1)3(
3(2)(2)1)(3(2)(2)1)(3(2)(2)1)3(1)(1)(1)0
```

La figure 3.5 montre les interprétations graphiques tridimensionnelles de ces quatre générations et de la cinquième.

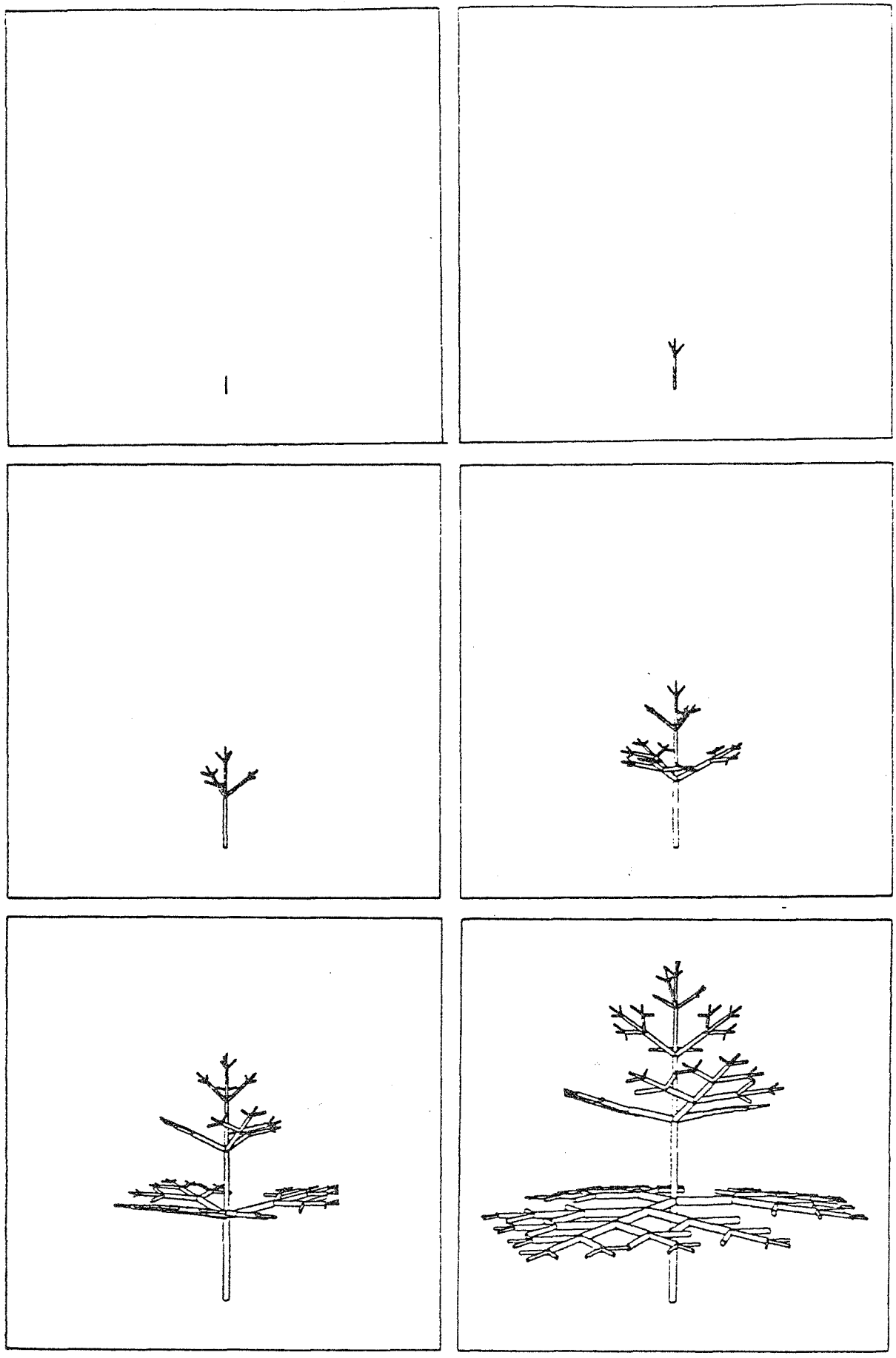


figure 3.5

Si l'on part de l'axiome 1, ce qui correspond en fait à la descendance d'une branche naissant sur le tronc, les cinq premières générations sont :

3(2)(2)1

3(3(3)(3)2)(3(3)(3)2)3(2)(2)1

3(3(3)(3)3(3)(3)2)(3(3)(3)3(3)(3)2)3(3(3)(3)2)(3(3)(3)2)3(2)(2)1

3(3(3)(3)3(3)(3)3(3)(3)2)(3(3)(3)3(3)(3)3(3)(3)2)3(3(3)(3)3(3)(3)2)(3(3)(3)3(3)(3)2)3(3(3)(3)2)(3(3)(3)2)3(2)(2)1

3(3(3)(3)3(3)(3)3(3)(3)3(3)(3)2)(3(3)(3)3(3)(3)3(3)(3)3(3)(3)2)3(3(3)(3)3(3)(3)3(3)(3)2)(3(3)(3)3(3)(3)3(3)(3)2)3(3(3)(3)3(3)(3)2)(3(3)(3)3(3)(3)2)3(3(3)(3)2)(3(3)(3)2)3(2)(2)1

La figure 3.6 représente ces cinq générations. On voit ainsi que ces règles permettent de contrôler le niveau de récursivité dans la descendance des branches.

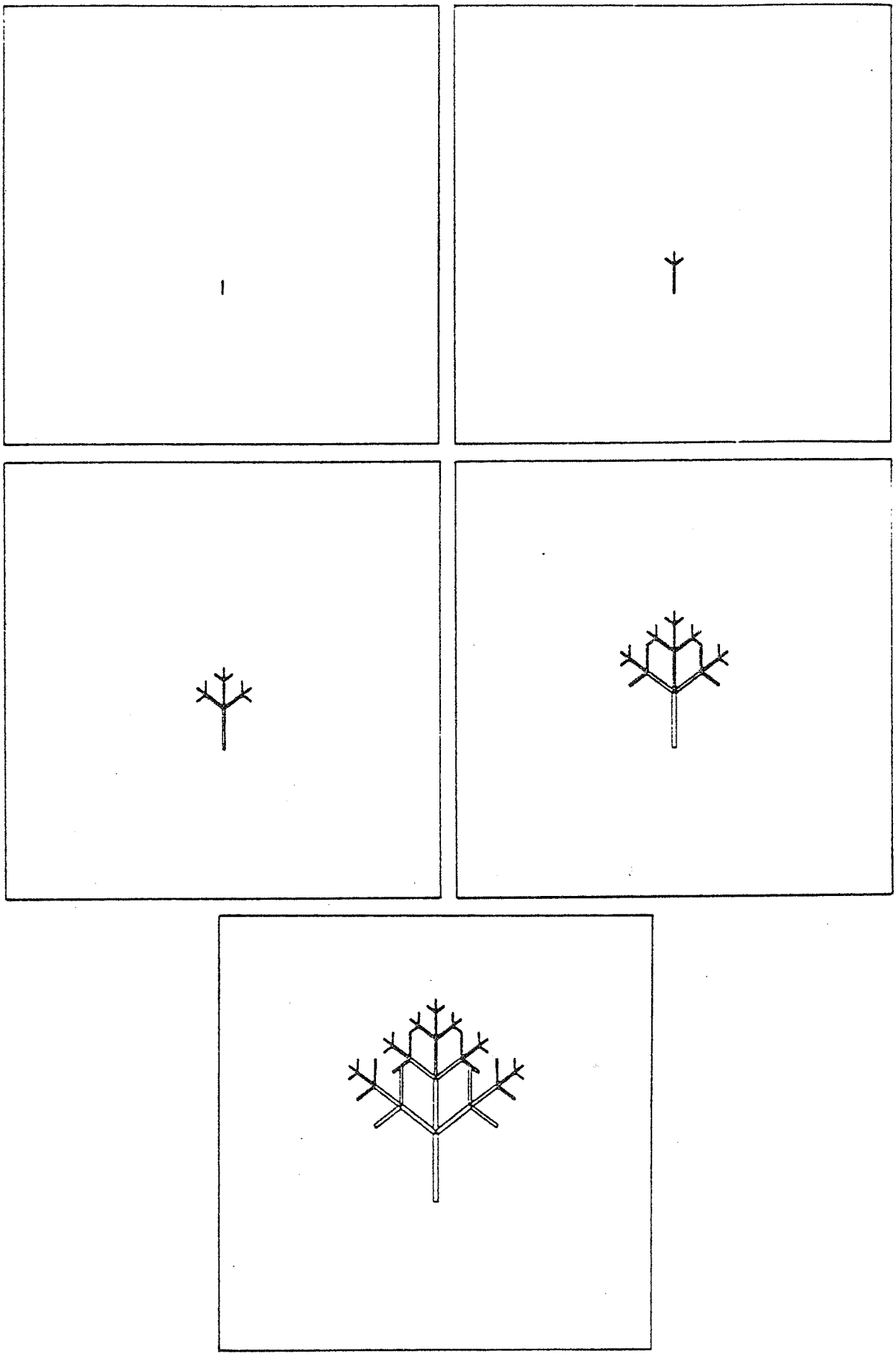


figure 3.6

Comme nous l'avons dit, nous gérons trois variables **a**, **g**, et **n** dont la signification a été donnée ci-dessus. Les valeurs de ces variables apparaissent entre une paire de crochets après chaque caractère dans l'ordre **a**, **n**, **g**. Ainsi, les deux premières chaînes engendrées par l'axiome 0 sont :

3[1,0,0]([0,0,1]1[0,0,1])([0,1,1]1[0,1,1])([0,2,1]1[0,2,1])0[0,2,1]

3[2,0,0]([1,0,1]3[1,0,1]([0,0,2]2[0,0,2])([0,1,2]2[0,1,2])1[0,1,2])([1,1,1]3[1,1,1]([0,0,2]2[0,0,2])([0,1,2]2[0,1,2])1[0,1,2])([1,2,1]3[1,2,1]([0,0,2]2[0,0,2])([0,1,2]2[0,1,2])1[0,1,2])3[1,2,1]([0,0,2]1[0,0,2])([0,1,2]1[0,1,2])([0,2,2]1[0,2,2])0[0,2,2]

Pour une généralisation, c'est-à-dire si l'on voulait pouvoir gérer d'autres variables que **a**, **g**, et **n**, on pourrait indiquer ceci dans les règles au moyen de la syntaxe suggérée par l'exemple suivant. Cet exemple ne reprend que la définition de ces trois variables.

0[a,g,n] → 3[a=a+1,n=n,g=g]([a=0,n=0,g=g0]1[a=0,n=0,g=g0])([a=0,n=1,g=g0]1[a=0,n=1,g=g0])([a=0,n=2,g=g0]1[a=0,n=2, g=g0])0[a=0,n=0,g=g0]

1[a,g,n] → 3[a=a+1,n=n,g=g]([a=0,n=0,g=g0]2[a=0,n=0,g=g0])([a=0,n=1,g=g0]2[a=0,n=1,g=g0])([a=0,n=2,g=g0]2[a=0,n=2, g=g0])1[a=0,n=0,g=g0]

2[a,g,n] → 3[a=a+1,n=n,g=g]([a=0,n=0,g=g0]3[a=0,n=0,g=g0])([a=0,n=1,g=g0]3[a=0,n=1,g=g0])([a=0,n=2,g=g0]2[a=0,n=2, g=g0])2[a=0,n=0,g=g0]

3[a,g,n] → 3[a=a+1,g=g,n=n]

Enfin, on pourrait aussi intégrer les règles géométriques comme des règles de réécriture à appliquer comme dernière étape. Pour engendrer la description de l'arbre en langage **CASTOR**, nous utilisons les règles suivantes :

0 | 1 | 2 | 3 sont remplacés par `@a(Radius0 * (Age+1), Radius0 * (Age+1), Length0 * (Age+1)) !cy( T_BRANCH )` suivi de , `@t(0, 0, Length0 * (Age+1)) $U(` si un autre objet est supporté par le morceau de branche représenté par le caractère 0, 1, 2, ou 3 qui a déclenché la règle présente ; c'est-à-dire si le caractère suivant est soit encore 0, 1, 2, 3, soit une parenthèse ouvrante ;

(est remplacé par `@r(0, 0, 1, Generation * Theta0 + Numero * Theta1) @r(0, 1, 0, Phi0 + Age * Phi1) $U(` si la parenthèse est de niveau zéro, c'est-à-dire si elle décrit une branche attachée au tronc ;

(est remplacé par `@r(1, 0, 0, if (Numero) Alpha0; else -Alpha0) $U(` pour les parenthèses correspondant aux branches attachées à d'autres branches ;

) reste inchangée.

Nous n'avons pas étudié le moyen de tout représenter d'une façon compacte avec une syntaxe permettant de décrire tous les objets nécessaires :

- les caractères à remplacer (typiquement, 0, 1, 2,...) ;
- les variables qui modifient leur valeurs à l'intérieur des règles (**a**, **g**, **n**) ;
- les variables externes au règles (g0, le niveau de récursivité) ;
- les règles exprimées ci-dessus, règles qui définissent l'interprétation géométrique.

Nous n'avons qu'implanté les règles décrites dans deux programmes en langage C. La modification de ces règles entraîne donc pour le moment une modification de ces programmes et une compilation.

Chapitre 4

VISUALISATION

4.1 INTRODUCTION

Si l'on se réfère à [MART 84], on sait qu'un système de synthèse d'images utilise six classes d'informations. Les initiales de leurs noms sont respectivement I,M,A,G,E, et S. Ces initiales signifient :

Identité — moyen de nommer les objets à synthétiser ;

Morphologie — informations qui décrivent la forme d'un objet ;

Aspect — la couleur, la matière, la texture ;

Géométrie — pour définir les informations relatives à la taille et la position des objets dans la scène. Le couple Géométrie-Morphologie permet de définir des équations ou des coordonnées pour tous les points, droites, plans ou autres éléments géométriques des objets ;

Eclairement — pour positionner et décrire les sources lumineuses, leur puissance, leur directivité, leur couleur,...

Structure — description des relations entre les objets qui constituent la scène, par exemple :

- le verre est *sur* la table,
- la table est formée *d'un ensemble indissociable* d'un plateau et de quatre pieds.

Ces informations sont effectivement toutes disponibles (à des degrés divers de précision cependant) dans un fichier de description écrit avec le langage **CASTOR** tel que nous l'avons décrit en 2.7.3.

A ces données qui décrivent une maquette tridimensionnelle viennent s'ajouter deux classes d'informations géométriques : d'une part, les informations définissant la projection de l'espace tridimensionnel dans le plan de l'écran (informations que nous nommerons **Gv** pour Géométrie de Visualisation) ; d'autre part, l'écran lui-même (par sa taille en pixels) et le positionnement de l'image synthétisée sur l'écran (informations que nous nommerons **Ga** pour Géométrie d'Affichage).

Les classes d'informations Identité et Structure ne sont utiles que pour une interaction du processus de synthèse soit avec un utilisateur soit avec un programme d'application ; aussi, dans la suite, comme nous ne nous intéressons qu'à l'étape de visualisation, nous ne considérerons que les informations : **M, A, G, E, Gv** et **Ga.**

Un processus complet de synthèse d'images doit prendre en compte ces six classes d'informations et les combiner pour fabriquer une image. Nous symboliserons les processus de synthèse par des couples de parenthèses. Un processus complet sera donc représenté par :

(M.A.G.E.Gv.Ga)

Les questions que nous abordons dans ce chapitre sont relatives au découpage d'un tel processus complet et à l'interdépendance des sous-processus. Après avoir passé en revue les différentes morphologies dans la section 4.2, nous étudierons, dans les sections 4.3, 4.4 et 4.5, différentes étapes de la synthèse d'images, à savoir :

- l'élimination des parties cachées,
- le rendu,
- l'antialiassage.

La section 4.6 présente l'application aux terrains. Enfin les deux dernières sections (4.7 et 4.8) présentent respectivement l'implantation actuelle de notre environnement destiné à la synthèse d'images et les développements futurs que nous envisageons pour lui.

4.2 LES MORPHOLOGIES

Dans le chapitre 2, nous avons décrit de nombreuses façons de représenter un objet solide tridimensionnel, et dans le chapitre 3, nous avons donné des exemples de modélisation et de représentation d'autres types d'objets.

Ces représentations donnent lieu à la construction de surfaces ou de volumes qui ne sont pas nécessairement représentables avec la même structure de données. Nous appellerons chacune de ces différentes structures une primitive : l'ensemble des instanciations de ces primitives définissant la morphologie de la scène. Pour fixer les idées, nous allons donner une liste (non exhaustive) des primitives couramment utilisées.

polygones définis par les coordonnées de leur sommets,

polygones définis par des listes de pointeurs sur les sommets,

Par rapport à la morphologie précédente, cette structuration permet aux processus gérant les données morphologiques de savoir qu'un sommet est commun à plusieurs polygones sans les difficultés dues aux imprécisions dans les calculs.

polyèdres Il y a ici beaucoup plus d'informations structurelles sur les diffé-

rents éléments géométriques (sommets, arêtes et faces) qui définissent un polyèdre. Ceci permet aux processus de trouver des listes d'arêtes pour les faces, de trouver les faces voisines à une face donnée,... On peut citer comme exemple typique la structure *Winged-Edge* décrite dans [BAUM 75].

liste de surfaces gauches

Les équations de ces surfaces peuvent être de différents degrés : quadriques, bicubiques,... De plus elles peuvent être représentées sous différentes formes : liste de points de contrôle, équation paramétrique, équation implicite,... Elles peuvent également être accompagnées de contraintes qui, elles aussi, peuvent prendre différentes formes aux niveaux modélisation et représentation.

structure tridimensionnelle

D'une façon générale, toute maquette tridimensionnelle peut être directement utilisée par un processus de synthèse.

modèles procéduraux

Ces modèles ne génèrent qu'une partie d'une morphologie d'un des types précédents à un niveau de détail juste suffisant à la visualisation depuis le point de vue souhaité.

modèles texturés

Ce sont les modèles où la texture intervient dans la visibilité des objets comme on l'a vu dans les exemples de nuages ou d'arbres de [GARD 84].

Le vocable de morphologie recouvre donc un vaste ensemble de primitives qui ont des niveaux de structuration très différents et des caractéristiques qui conduisent souvent à des algorithmes de visualisation assez spécifiques.

4.3 L'ELIMINATION DES PARTIES CACHEES

L'élimination des parties cachées reste le point fondamental de la synthèse d'images, autour duquel s'articulent les autres étapes.

Historiquement, la première approche qui a été utilisée pour traiter ce problème a été de projeter les primitives puis de traiter le problème de recherche des parties visibles. Cette première approche, que nous nommerons dans la suite *approche conventionnelle*, sera opposée à la méthode du *lancer de rayons*.

Nous voulons dans cette section essayer de dégager une taxinomie des algorithmes d'élimination des parties cachées. Pour cela, nous avons besoin dans un premier temps d'avoir une quantité suffisante de matériau, à savoir un nombre assez grand d'algorithmes d'élimination de parties cachées. La famille des algorithmes de visualisation de polygones est sans nul doute la plus nombreuse car elle est étudiée depuis une vingtaine d'années. Nous allons donc examiner, dans la section 4.3.1,

chacun des membres de cette famille en dégageant les idées qui nous permettront d'obtenir des éléments de classification. Puis nous évoquerons la méthode du *lancer de rayons* dans la section 4.3.2. Nous pourrons alors aborder les méthodes utilisables pour les surfaces en 4.3.3, puis pour des mélanges de plusieurs sortes de primitives en 4.3.4. Nous terminerons par l'étude des algorithmes visualisation pour la modélisation CSG dans la section 4.3.5.

4.3.1 Le cas des polygones

Dans cette section, nous ne voulons nous intéresser qu'aux problèmes de détermination des parties visibles d'un ensemble de polygones. Nous ne voulons donc pas ici mêler les calculs nécessaires au rendu avec ces algorithmes ; rappelons que ceci sera étudié dans la section 4.4. L'antialiassage sera étudié en 4.5.

Ainsi, nous supposerons que l'aspect d'un pixel peut être déterminé en ne connaissant que l'identificateur du polygone visible sur ce pixel. Dans les figures de cette section cette information permettant d'identifier les polygones sera indiquée par la lettre **I**.

Il est utile de rappeler le principe de remplissage d'un polygone. La question est classique et on peut trouver la description d'un de ces algorithmes dans [FOLE 82]. Ces algorithmes procèdent en deux étapes, la première calcule l'intersection de la ligne courante de balayage avec toutes les arêtes du polygone et trie ces intersections par ordre croissant d'abscisses. La deuxième balaye les pixels situés entre chaque paire d'intersection. La figure 4.1 montre l'organisation de ce processus. La même allure apparaîtra dans certains des algorithmes d'élimination des parties cachées.

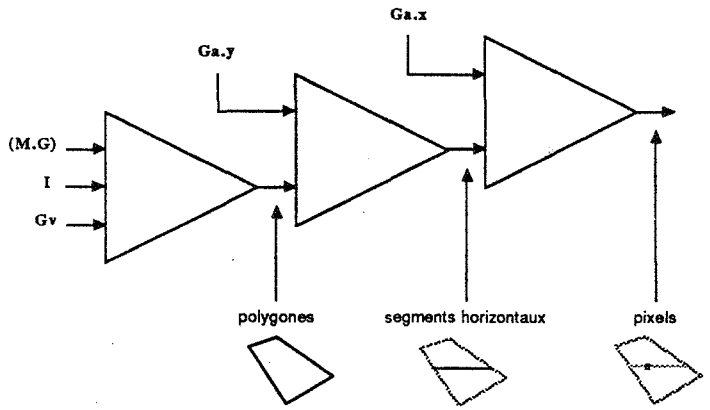


figure 4.1

4.3.1.1 Les différents algorithmes
Tout processus réalisant l'élimination des parties cachées fait intervenir un opérateur de *composition* chargé de déterminer ce qui est visible. C'est dans le sens où cet opérateur fait intervenir plusieurs objets qu'il s'agit d'un opérateur de composition.

Cette composition peut avoir lieu à plusieurs niveaux du processus de synthèse suivant les objets sur lesquels elle s'applique :

- des points (algorithme du type z-buffer) ;
- des segments coplanaires (algorithme du type Watkins) ;
- des polygones (algorithmes de l'un des types Warnock, Atherton & Weiler, Michelucci, Newell, Fuchs) ;
- des polygones convexes, *puis* des ensembles de polygones convexes linéairement séparables et possédant d'autres propriétés, nommés *clusters* (algorithme du type Schumacker).

Tous ces algorithmes utilisent une mémoire où doivent figurer tous les éléments susceptibles d'être visibles juste avant l'opérateur de composition. Ce dernier peut alors effectuer un tri ou une sélection parmi ces éléments et/ou des opérations plus complexes telles que le découpage de certains objets.

4.3.1.1.1 Le z-buffer
Dans cet algorithme, la sélection des éléments visibles s'effectue au niveau des pixels. Pour chaque pixel de coordonnées (i,j) de l'écran sont conservées deux valeurs : id[i,j], l'identificateur du polygone visible en ce point, et z[i,j], la profondeur du polygone visible en ce point. Le processus effectuant le remplissage des polygones calcule $z_p[i,j]$ la profondeur du polygone courant, dont l'identificateur est id_p, sur le pixel de coordonnées (i,j). Avec ces notations, l'algorithme de l'opérateur de composition peut s'exprimer ainsi : si la profondeur $z_p[i,j]$ est plus petite que la profondeur z[i,j], cela signifie que, entre tous les polygones se projettant sur ce pixel, P est le plus proche et il faut donc mettre la valeur $z_p[i,j]$ dans z[i,j] et la valeur de id_p dans id[i,j].

La figure 4.2 donne une représentation de l'ordonnancemment de cet algorithme. L'étape de *composition* est symbolisée par une boîte à côtés obliques.

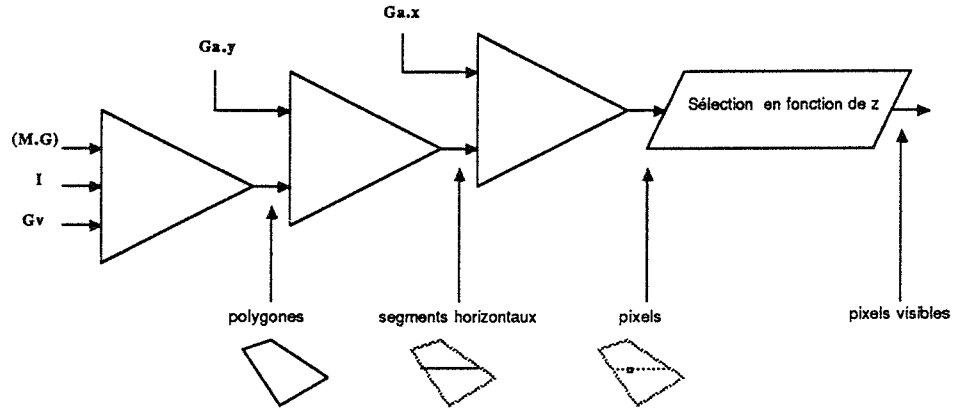


figure 4.2

On voit qu'on retrouve le même schéma d'organisation que sur la figure 4 avec l'ajout de l'opérateur de composition en dernière étape.

4.3.1.1.2 Watkins

Dans cet algorithme, la composition se fait au niveau de segments coplanaires. Le plan commun à tous les segments composés à chaque étape est le plan passant par le point de visée et par une ligne de balayage de l'écran. La figure 4.3 représente l'organisation fonctionnelle de cet algorithme.

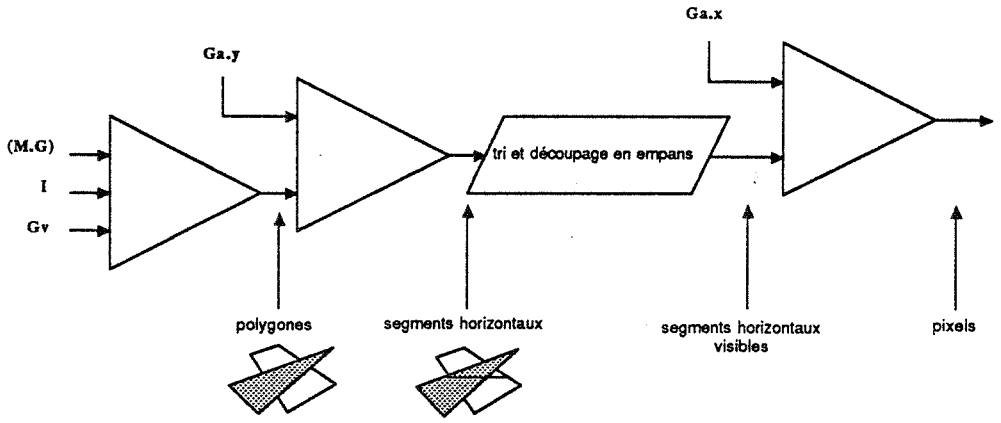


figure 4.3

La synthèse des informations Ga doit intervenir très tôt du fait de l'utilisation de lignes de balayage. De ce fait, cet algorithme est complètement dépendant du terminal de visualisation utilisé, et plus précisément de sa résolution.

4.3.1.1.3 Warnock
Pour cet algorithme, comme pour ceux des trois paragraphes suivants, la composition s'effectue au niveau des polygones. Cet algorithme travaille dans l'espace image et découpe de manière récursive celui-ci jusqu'à ce que la situation soit suffisamment simple, soit parce qu'un seul polygone recouvre un morceau d'écran, soit parce que ce morceau d'écran est lui-même réduit à la surface d'un pixel. Le schéma fonctionnel de ce polygone est représenté sur la figure 4.4.

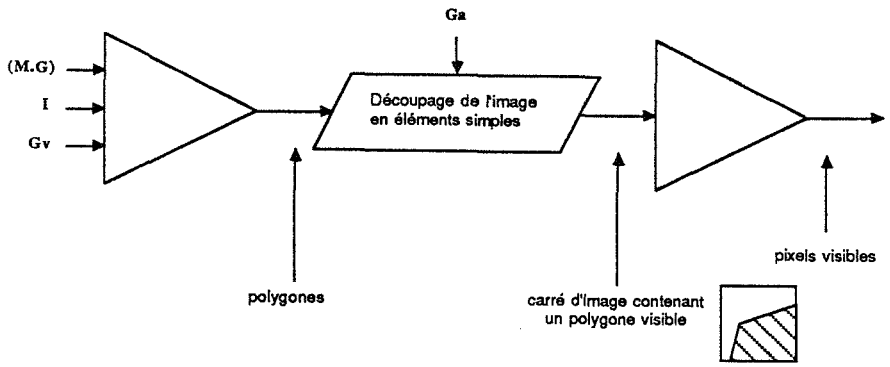


figure 4.4

variante 1
Remarquons que, tel qu'il est décrit dans [FOLE 82], le découpage récursif de l'espace image est réalisé avec les coordonnées écran. On pourrait aussi bien effectuer ce découpage avec des coordonnées normalisées représentées par des réels, ce qui permettrait d'atteindre un niveau de précision aussi grand que voulu (plus exactement aussi grand que le permet la représentation des nombres réels sur la machine utilisée).

variante 2
Une deuxième variante de cet algorithme consiste à ne pas découper l'espace image en deux par son milieu selon chacun des axes x et y, mais par les coordonnées d'un sommet de l'un des polygones.

4.3.1.1.4 Atherton et Weiler
L'algorithme d'Atherton et Weiler [WEIL 77] travaille dans l'espace objet. Grâce à un tri et à des découpes sur les polygones qu'il traite, il donne à la sortie de la phase de composition la liste des morceaux de polygones visibles. Ces polygones sont ensuite affichés par un processus analogue à celui présenté dans l'introduction de la section 4.3.1.

La différence avec la première variante de l'algorithme de Warnock présentée ci-dessus est dans la méthode utilisée pour le découpage. Warnock utilise un découpage récursif alors que Atherton effectue les découpages par rapport à des plans contenant certains des côtés des polygones. On peut donc voir l'algorithme

d'Atherton et Weiler comme le prolongement de la deuxième variante de l'algorithme de Warnock qui découpait l'espace image en utilisant un sommet d'un polygone comme pivot, puisque lui-même utilise une arête d'un polygone comme axe de découpe.

L'algorithme de Warnock se prête bien à une implantation câblée ou en langage d'assemblage sur une machine ne disposant pas d'opérations flottantes puisqu'il n'utilise pour ses découpes récursives que des divisions par deux. Par contre l'algorithme d'Atherton ou la deuxième variante de l'algorithme de Warnock du paragraphe précédent, s'ils permettent de diminuer le nombre de subdivisions en les optimisant par rapport aux objets, compliquent ces opérations de subdivision et nécessitent l'emploi de coordonnées réelles.

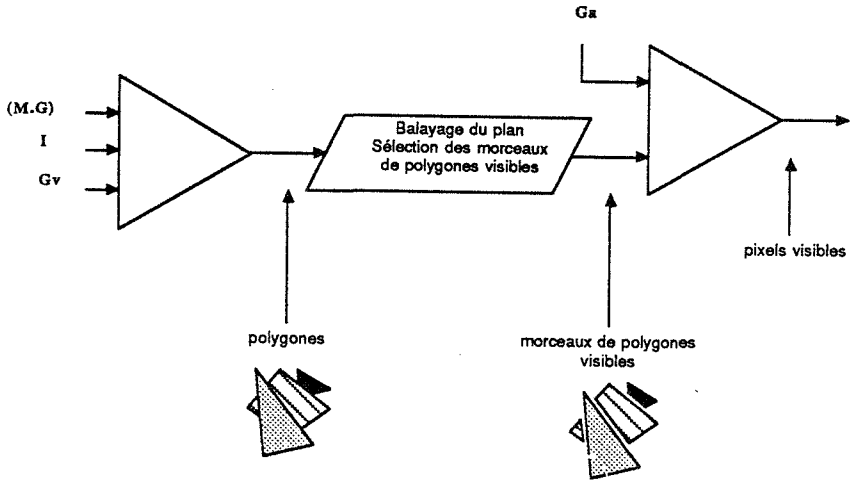


figure 4.5

4.3.1.1.5 Michelucci
Cet algorithme, décrit dans [MICH 87a], détermine dans l'espace objet quels sont les morceaux de polygones visibles ; il produit les mêmes résultats que l'algorithme d'Atherton. L'algorithme de Michelucci optimise la recherche des intersections des projections des arêtes des polygones grâce à la méthode de Bentley et Ottman [BENT 79].

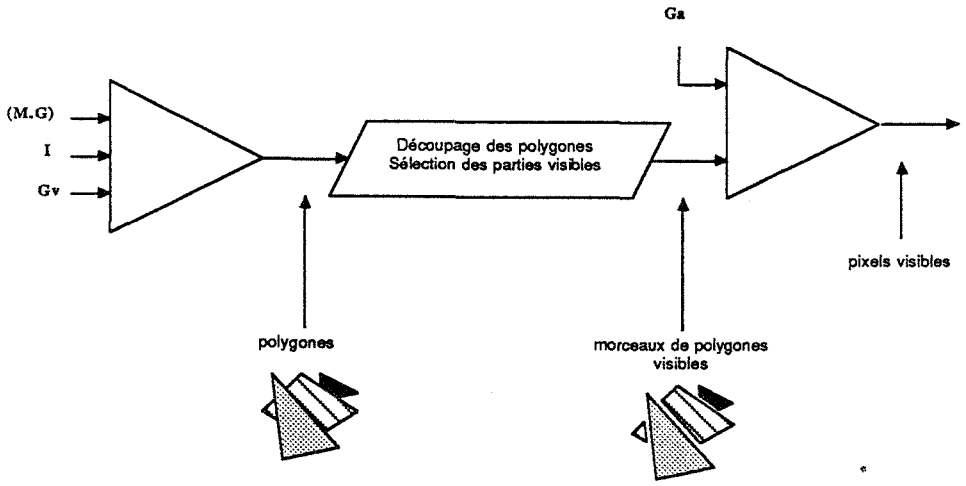


figure 4.6

4.3.1.1.6 Newell, Newell et Sancha

Ici, l'opérateur de composition est découpé en deux étapes : le tri, puis la sélection des parties visibles. Les polygones sont triés par ordre de profondeur par rapport à l'écran, du plus lointain au plus proche. Dans le cas où il y a chevauchement de profondeur entre deux polygones, l'un est découpé en deux par le plan contenant le deuxième polygone. Cette première phase est exécutée dans l'espace objet. Ensuite ces polygones sont envoyés suivant l'ordre de tri dans le processus de remplissage qui les traite successivement. Ainsi la sélection des pixels visibles s'effectue de façon implicite puisque c'est le dernier remplissage qui aura écrit la couleur d'un pixel qui détermine la couleur de ce pixel. Bien entendu, ce dernier remplissage correspond au polygone le plus proche de l'œil en ce point.

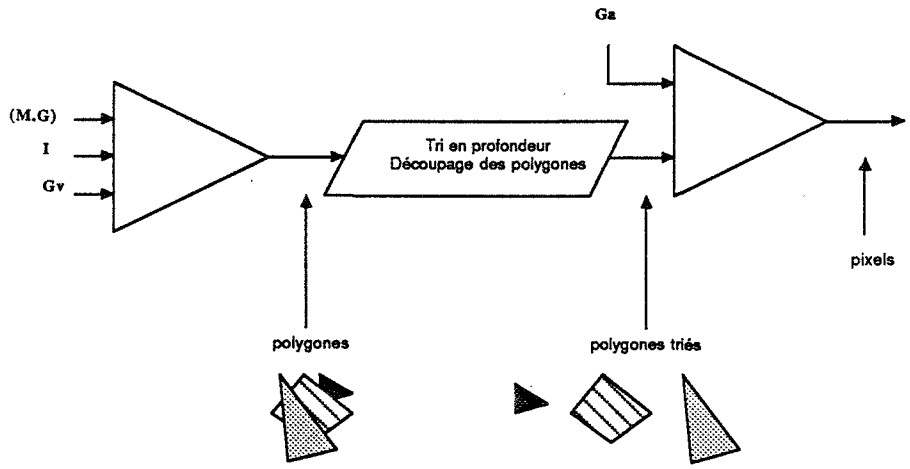


figure 4.7

Ce mode de visualisation exige la réinscriptibilité du support conservant l'image. Ceci est donc possible avec une mémoire à semi-conducteurs, mais pas avec une pellicule photographique ou une imprimante.

4.3.1.1.7 Schumacker
Cet algorithme ne travaille pas sur des données quelconques : il faut que les polygones soient convexes, et que les *cluster*s soient linéairement séparables. Un *cluster* est un ensemble de faces sur lequel on peut définir une numérotation qui, pour tout point de vue, permet de trouver un ordre d'affichage simplement en éliminant les faces *arrières* (c'est-à-dire les faces vues de derrière par rapport à l'intérieur de l'objet solide auquel elles appartiennent).

Cet algorithme fabrique un ordre d'affichage sur les polygones (donc nécessairement dans l'espace objet). Cet ordre est engendré rapidement une fois le point de vue connu grâce à l'utilisation de structures engendrées auparavant. Ces structures sont :

- les *cluster*s, qui sont donc des ensembles de polygones numérotés, le même numéro pouvant être utilisé plusieurs fois (cf. figure 4.8),
- un arbre de partition binaire de l'espace, qui existe de par l'hypothèse de séparabilité vérifiée par les *cluster*s. La figure 4.9 montre un exemple d'un tel arbre pour trois *cluster*s dans un plan.

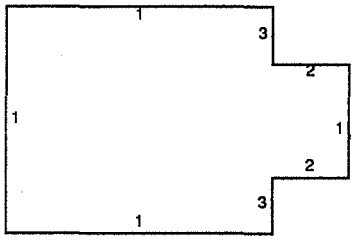


figure 4.8

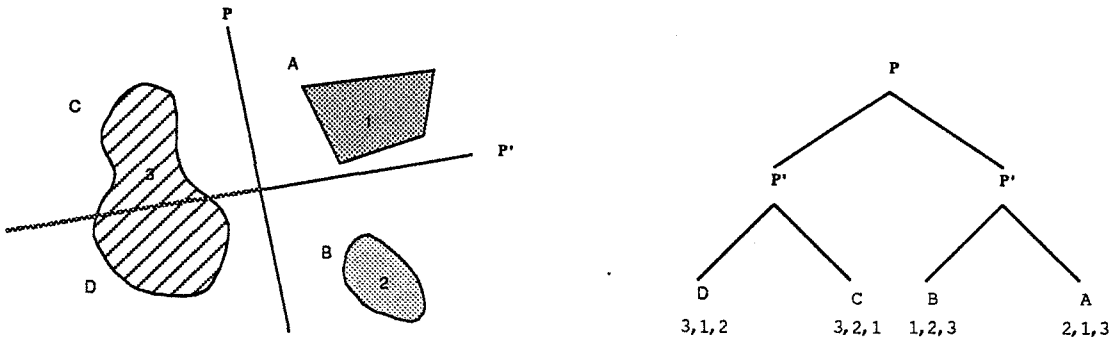


figure 4.9

La figure 4.10 montre le schéma d'organisation de l'algorithme de Schumacker et fait apparaître la pénétration de l'information *Géométrie de Visualisation.* De la sorte, cet algorithme est intéressant lorsque les processus indépendants de l'information géométrie de visualisation produisent des résultats qui peuvent être utilisés pour plusieurs images. En particulier, c'est le cas lorsque l'environnement de polygones n'est pas fonction du temps : par exemple le terrain d'évolution d'un avion, avec sa végétation, ses constructions,... En fait cet algorithme a été développé à l'origine pour les simulateurs de conduite d'avions.

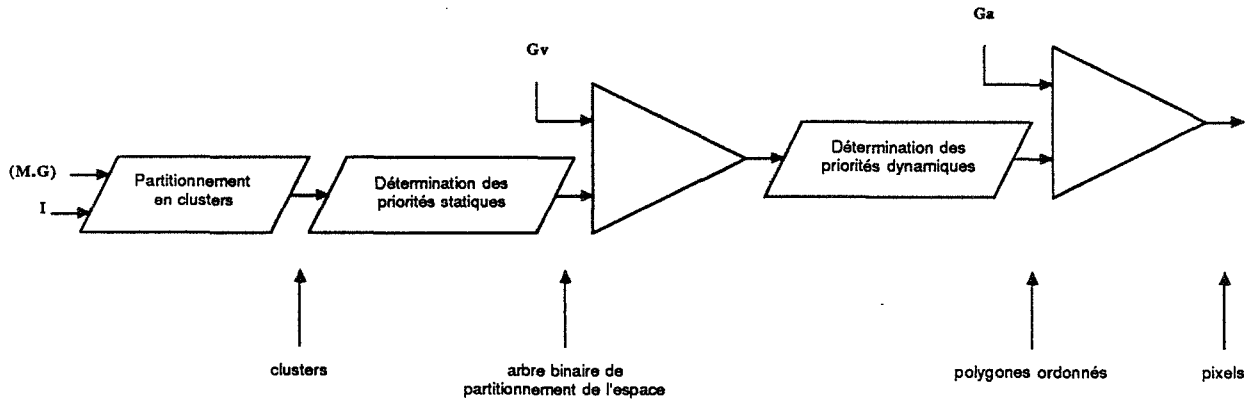


figure 4.10

Tout comme l'algorithme de Newell, Newell et Sancha, cet algorithme ne peut être utilisé que si le support d'image est réinscriptible.

4.3.1.1.8 Fuchs
C'est encore un algorithme qui génère un ordre d'affichage pour les polygones comme l'algorithme de Newell, Newell et Sancha, et celui de Schumacker. C'est d'ailleurs un peu une combinaison de ces deux algorithmes ; en effet, comme celui de Schumacker, il commence par construire un arbre de partition binaire de l'espace. Cependant, comme l'algorithme de Newell, il ne travaille que sur des polygones. On peut ainsi le voir comme l'algorithme de Schumacker où les *cluster*s sont réduits à des polygones. Une différence cependant avec l'algorithme de Schumacker est qu'il génère automatiquement toutes les découpes des polygones lorsqu'ils ne peuvent être séparés par des plans, alors que pour l'algorithme de Schumacker, si l'on en croit [SUTH 74], il faut définir manuellement (*sic*) des *cluster*s plus petits lorsqu'on ne peut trouver de numérotation sur ceux-ci!

Terminons par une dernière ressemblance entre l'algorithme de Fuchs et celui de Newell : le découpage des polygones, lorsque ceux-ci ne sont pas linéairement séparables, est fait par rapport au plan contenant l'un des polygones.

4.3.1.2 Essai de classification
Nous constatons une première classification de ces algorithmes suivant la *dimension* des objets qu'ils composent :

- dimension 0 : des points,
- dimension 1 : des segments,
- dimension 2 : des polygones,
- dimension 3 : des polyèdres.

Une deuxième clé pour cette classification est l'espace dans lequel la composition s'effectue : l'espace-image ou l'espace objet. Ainsi on peut dresser un tableau à deux entrées où nous pourrons inclure chacun des algorithmes présentés.

	Objet	Image
points		z-buffer
segments	Watkins	
polygones	Atherton et Weiler Michelucci-Fuchs	Warnock
polyèdres	Schumacker	

figure 4.11

Une exception toutefois, l'algorithme de Newell, Newell et Sancha n'apparaît pas dans ce tableau car il utilise à la fois l'espace image et l'espace objet. Par ailleurs, nous n'avons classé l'algorithme de Watkins ni comme un algorithme travaillant dans l'espace image ni comme un algorithme travaillant dans l'espace objet. Cela vient de ce que nous n'avons pas détaillé ce qui se passe à l'intérieur du processus de composition qui doit trier les segments et sélectionner les morceaux de segments visibles. Cet algorithme garde la liste des abscisses des points d'intersection du plan de balayage avec les arêtes sous forme de réels, et il détermine la visibilité des polygones sur ces points en calculant la profondeur en réel également. Pour cette étape de composition, il se place donc dans l'espace objet, alors que l'étape de parcours des plans de balayage se place dans l'espace image. Il y a de nombreuses possibilités de variantes ; essayons de raisonner pour déduire les algorithmes possibles à ce niveau.

[SUTH 74] a classé les algorithmes suivant l'ordre dans lequel étaient utilisées les différentes coordonnées pour faire le tri. Ce tri peut se faire selon une direction spécifique, *X*, *Y* ou *Z*, ou selon une combinaison de direction comme *(XY)* pour l'algorithme de Warnock. Le couple de parenthèses indique cette combinaison de directions. Cela l'a conduit à donner la classification suivante (dans laquelle j'ai éliminé les différences entre *XY* et *YX*, grâce à la symétrie complète de ces deux coordonnées pour tous les algorithmes : il suffirait de faire tourner l'ecran de 90° pour échanger ces deux coordonnées ; les algorithmes privilégient en général *Y* à cause du sens du balayage de rafraîchissement des écrans) :

ZYX Newell et Schumacker,

YXZ Watkins,

YZX algorithme non essayé,

(XY)Z Warnock,

Z(XY) variation de Newell.

La notation choisie par [SUTH 74] ne permet pas de faire apparaître la distinction entre ce qui se passe dans l'espace image et ce qui se passe dans l'espace objet. Aussi, je propose de noter avec une majuscule lorsque le tri est effectué dans l'espace objet et avec une minuscule lorsqu'il est effectué dans l'espace image. Ainsi les algorithmes que nous avons étudiés pourrons être décrits par les mots suivants :

xyz z-buffer,

yXZ Watkins,

(XY)Z Atherton, Michelucci,

(xy)Z Warnock,

Z(yX) Newell,

(ZYX) Schumacker, Fuchs.

Cette notation permet de définir les algorithmes d'une façon beaucoup plus précise, et d'en décrire un nombre beaucoup plus grand que la notation de [SUTH 74]. Cependant, des deux critères de classification que j'ai proposés n'apparaît que celui qui différencie l'espace objet et l'espace image par l'utilisation de majuscules et minuscules. Je complète donc cette notation en faisant apparaître dans quel sous-espace sont composés les objets par un couple de crochets :

xy[z] z-buffer,

y[XZ] Watkins,

[(XY)Z] Atherton, Michelucci,

[(xy)Z] Warnock,

[Z(yX)] Newell,

[(ZYX)] Schumacker.

Ainsi la notation est complète par rapport aux critères de classification. Il faut remarquer que nous avons pu la compléter, grâce à l'utilisation de crochets, parce que les objets composés ne sont pas quelconques par rapport aux axes de tri :

- dans la méthode de Watkins, les segments sont coplanaires dans un plan contenant les directions de deux axes de coordonnées ;
- dans la méthode du z-buffer, les points composés sont colinéaires sur une droite parallèle à l'axe des z.

Cela n'est toutefois pas une restriction sur les algorithmes représentables par cette notation ; en effet, si on fait un tri dans une direction indépendamment de la composition, on est amené à couper les polygones de façon à avoir un ordre absolu dans cette direction, ce qui signifie qu'on n'aura donc pas besoin de la considérer

lors de la composition.

Une dernière remarque, le couple de crochets représente l'espace dans lequel a lieu la composition. De ce fait, le nombre de lettres que contient cette paire de crochets donne la dimension de l'espace de composition et non la dimension des objets composés. Ainsi dans l'espace de dimension trois, on peut composer soit des volumes avec la méthode de Schumacker, soit des surfaces avec les méthodes de Newell, de Warnock,...

Nous allons alors pouvoir systématiser notre étude pour les algorithmes travaillant ligne par ligne, et classés, jusqu'ici, sous le nom d'algorithme de Watkins. Ils partagent tous le premier tri qui s'effectue grâce à un tri par seaux et un suivi des arêtes, le plus souvent grâce à une méthode incrémentale. Les algorithmes diffèrent pour les étapes suivantes qui peuvent donc être : *XZ Xz xZ xz ZX Zx zX zx*. Ce qui peut être éventuellement complété par un couple de parenthèses encadrant le couple de lettres. Examinons les diverses possibilités :

(XZ) Dans cette variante, on cherche d'abord toutes les intersections des segments dans le plan. Un tri de ces intersections et des extrémités des segments selon l'axe des abscisses permet de déterminer sur chacun des empans, définis par les couples de deux abscisses successives, lequel parmi tous les segments est visible. Cette visibilité ne change pas sur un empan grâce à la convexité des segments. La recherche des intersections peut se faire soit avec la méthode de Bentley et Ottman, soit avec une méthode récursive par subdivisions successives.

XZ La version *XZ* procède de la même façon pour la deuxième étape, mais elle cherche les intersections des segments simultanément au tri. Le premier tri, selon l'axe des abscisses, peut se faire de deux façons différentes : soit par une méthode de balayage, soit par une découpe récursive selon cet axe. La première méthode consiste à balayer le plan par une droite d'équation $x=x_0$. Cette droite intersecte certains des segments, ce qui permet de définir pour chaque valeur de x_0 la liste des segments actifs. Il reste à trouver les valeurs de x_0 où la tête de liste change. Cette tête de liste décrit en effet quel est le segment visible jusqu'au prochain changement.

Cette méthode évite donc par rapport à la précédente un certain nombre de calculs en ne cherchant pas les intersections de tous les segments mais seulement celles qui sont indispensables, c'est-à-dire celles qui modifient la tête de liste.

La méthode *(XZ)* avec recherche des intersections au moyen de l'algorithme de Bentley et Ottman est très proche de la méthode *XZ*. En effet, l'algorithme de Bentley et Ottman recherche les intersections de segments dans un plan grâce à un balayage du plan selon la direction de l'un des axes de coordonnées. La différence réside dans le fait que pour le problème de l'élimination des parties cachées qui nous intéresse, on n'a pas besoin de connaître toutes les intersections.

Xz Tout comme dans la version *XZ*, on fait d'abord un tri avec un balayage

selon l'axe des abscisses, la sélection s'effectue en profondeur avec un échantillonnage des profondeurs, ce qui n'apporte rien par rapport à la méthode précédente.

xZ Ici, un échantillonnage dans la direction de l'axe des abscisses permet à la méthode *xZ* de faire un balayage *point par point* (par analogie avec la méthode de balayage *ligne par ligne* qui échantillonne selon l'axe des ordonnées). C'est cependant assez peu naturel de terminer cette méthode de balayage point par point sans échantillonner en profondeur, ce que fait au contraire la méthode *xz*.

Au lieu d'un balayage selon *x*, on peut aussi avoir un découpage récursif selon cet axe, mais encore une fois ce serait curieux de conserver les abscisses avec beaucoup moins de précision que les profondeurs.

(xz) La méthode *(xz)* découpe récursivement le plan jusqu'à ce que la situation soit suffisamment simple à l'intérieur du morceau de plan, c'est-à-dire, jusqu'à ce qu'un seul segment intersecte le morceau de plan, ou qu'un seul segment traverse tout le morceau de plan et que les profondeurs de ses extrémités soit plus petites que celles des extrémités de tous les autres segments. Cet algorithme paraît assez lourd à mettre en oeuvre.

xz Cette méthode balaye selon l'axe des abscisses puis pour chaque pixel, elle sélectionne le segment le plus proche. C'est en fait un parcours simultané de tous les segments dans le plan *xz* avec de plus une sélection selon la profondeur. Cette méthode est analogue à la méthode du z-buffer, mais ne nécessite pas l'usage du tampon lui-même.

Le gain que l'on peut espérer de la méthode *(xz)* par rapport à la méthode *xz* vient de la cohérence horizontale qui fait qu'on ne change pas de segment visible en chaque pixel.

Dans les méthodes où le premier tri s'effectue selon la profondeur, on va trouver les méthodes basées sur la priorité, c'est-à-dire les méthodes qui vont définir un ordre d'affichage pour les segments. Par ailleurs, les méthodes qui regroupent les deux directions au moyen de parenthèses sont bien évidemment les mêmes que celles qui ont été décrites ci-dessus.

ZX La méthode *ZX* consiste à trier les segments à afficher selon leur profondeur comme la méthode de Newell trie les polygones selon leur profondeur. Dans une telle méthode, il faut aussi couper les segments se recouvrant en profondeur ; ce qui implique que l'usage de la seule composante profondeur n'est pas suffisante pour faire ce tri, ce qui est aussi le cas pour la méthode de Newell.

Zx Tout à fait analogue à la méthode précédente, celle-ci trie les segments selon leur profondeur. L'échantillonnage selon les abscisses ne peut intervenir qu'après la recherche des intersections dans le plan *XZ*. La méthode est donc la même que *ZX*.

zX Le premier tri en profondeur peut être effectué avec un tri par seaux avec ainsi une notion de segments actifs. La sélection se fait en parcourant ceux-ci depuis les plus proches jusqu'aux plus lointains. L'algorithme se termine lorsque toute la largeur de l'écran est remplie.

zx Identique à la méthode précédente sauf que le test de remplissage de la largeur de l'écran se fait cette fois sur l'affectation de tous les pixels.

4.3.1.3 Conclusion

Les critères que nous avons décrits permettent de classer les algorithmes d'élimination des parties cachées. Toutefois, une synthèse s'impose.

Le plus significatif parmi ces critères me paraît être celui de *dimension* des objets composés ; c'est celui qui sépare les algorithmes en trois classes :

- z-buffer,
- Watkins,
- les autres.

Le critère *espace image/espace objet* n'est en fait qu'un critère de précision sur les nombres représentant les coordonnées des sommets des polygones. Du choix qui est fait à ce niveau va dépendre la précision du résultat que l'on va obtenir en sortie de tel ou tel algorithme. Pour s'en convaincre, examinons deux de ces algorithmes qui ne diffèrent que par l'usage de l'*espace image* ou de l'*espace objet* : celui de Warnock et celui de Michelucci. Le premier travaille dans l'espace image, il subdivise cet espace jusqu'à arriver à une situation simple : soit un seul polygone cache tous les autres, soit on est au niveau du pixel. Le résultat obtenu est une image, c'est-à-dire que pour chaque pixel, on sait quel est le polygone le plus proche au centre du pixel. C'est une information *discrète* au sens topologique du terme. Par contre, le résultat de l'algorithme de Michelucci qui effectue tous ses calculs sur des nombres flottants (ce qui lui vaut le label *espace objet*) donne une liste de polygones visibles qui ne se recouvrent pas. Ce n'est que par dessus cette information que vient se greffer la visualisation effective. Ce qui revient à dire que la pénétration de l'information *géométrie de visualisation* est maximum. La sortie de cet algorithme est donc aussi bien utilisable pour générer une image sur un écran que pour fabriquer un dessin sur une table traçante.

Une distinction algorithmiquement plus importante que cette séparation *espace image/espace objet* me paraît plutôt être la méthode qui est employée pour chercher les intersections des objets composés. Cette distinction apparaît dans la description que j'ai donnée des méthodes permettant d'achever l'algorithme de Watkins. Mais ici encore, c'est la comparaison des algorithmes du genre *[(XY)Z]*, indépendamment de la distinction *espace image/espace objet*, qui va éclaircir ce point. Nous allons donc regarder de ce point de vue les algorithmes de Atherton, Warnock et Michelucci.

Le premier recherche ces intersections en prenant les polygones par rapport auxquels il fait les découpes les uns après les autres. Il évite une complexité en $O(n^2)$ par un tri grossier en z préliminaire, ce qui conduit

à diminuer tout de suite le nombre de polygones à couper par rapport au premier de ces polygones. Il cherche donc les intersections directement.

Le second procède par découpe récursive.

Le troisième procède par balayage.

On peut enfin constater que notre classement peut être interprété comme une position du processus de composition par rapport aux sous-processus d'un processus de remplissage d'un polygone. Nous avons décrit dans l'introduction de la section 4.3.1 le plus courant de ces processus de remplissage. Tous les algorithmes d'élimination des parties cachées que nous avons décrits, à l'exception de celui de Warnock, utilisent cette technique de remplissage des polygones. Le z-buffer compose après les deux sous-processus du remplissage, Watkins compose entre ces deux sous-processus, et les autres composent avant ceux-là. Quant à l'algorithme de Warnock, il effectue le remplissage par bloc carré sur l'image au fur et à mesure qu'il détermine des portions de polygones visibles.

4.3.2 L'approche inverse : le lancer de rayons

Le point de départ de la méthode du *lancer de rayons* est à l'opposé de celui des algorithmes traditionnels. Au lieu de considérer les objets puis d'essayer de les trier, cette méthode considère en premier lieu les pixels de l'écran et essaye de trouver quels sont les objets qui interviennent dans la définition de la couleur de ce pixel. Cette façon de procéder simule en partie et à l'envers le trajet des rayons lumineux ; de la sorte, il est assez facile d'introduire la simulation de lois simples de l'optique géométrique. Par *lois simples*, j'entends des lois qui, à un rayon, n'associent qu'*un* nouveau rayon, comme par exemple la réflexion, la réfraction,... Il est beaucoup plus difficile de considérer les lois de diffusion ou des lois d'interactions d'objets entre eux qui sont la résultante d'un nombre *a priori* infini de rayons lumineux.

Rappelons l'algorithme de base :

```
Pour chaque point de l'écran faire
        Trouver la droite passant par le point de visée et le point de l'écran;
        Pour chaque objet faire
                tester l'intersection entre la droite et l'objet;
        Trier les intersections;
        Pour la plus proche calculer le rendu du point et afficher le résultat;
```

Il va de soi que ce n'est ici que l'algorithme naïf qui est complètement inutilisable sous cette forme.

4.3.3 Généralisation des algorithmes aux surfaces

Le but de ce paragraphe n'est pas de détailler tous les algorithmes d'affichage de surfaces gauches, mais d'essayer de dégager les principes généraux qui permettent de les classer.

Les surfaces paramétriques sont représentées par trois fonctions de deux variables réelles à valeur réelle :

$X = f(u,v)$
$Y = g(u,v)$
$Z = h(u,v)$

où les variables u et v appartiennent à l'intervalle [0,1]. Lorsque nous parlons de surfaces bicubiques, cela signifie que chacune des trois fonctions f, g, h, est une fonction polynomiale de degré trois par rapport à chacune des variables. Pour les surfaces paramétriques, nous étudierons les publications suivantes :

- [CATM 74]
- [LANE 79]
- [LANE 80], qui présente trois algorithmes, un de Blinn, un de Whitted et un de Lane et Carpenter. Nous les désignerons dans la suite par [blin 80], [whit 80], et [lane 80] respectivement, bien qu'ils fassent tous référence au même article.
- [SCHW 82]

Pour les surfaces quadriques, seul [MAHL 72] sera considéré.

Un rapide examen des algorithmes étudiés montre que ceux-ci, à l'exception de [MAHL 72], font une approximation des surfaces par des éléments linéaires pour effectuer la composition des objets. Nous allons essayer dès l'abord de cette étude de les classer suivant l'endroit des algorithmes où intervient cette approximation.

[LANE 79] et [lane 80] approximent dès le départ la surface, paramétrique pour [LANE 79], bicubique pour [lane 80], par un ensemble de morceaux de surfaces réglées dont les droites génératrices sont toutes orthogonales à l'axe des ordonnées. Cette approximation est fonction d'un test de planéité sur les sommets du morceau de surface. Une telle approximation permet d'assurer que les intersections des plans de balayage avec ces surfaces réglées sont des segments de droites. Par ailleurs, les frontières de ces surfaces sont aussi des segments de droites. Ces deux qualités permettent d'effectuer le remplissage de ces surfaces réglées comme s'il s'agissait de polygones en utilisant la méthode décrite en 4.3.1. Ces deux algorithmes, basés sur une méthode de balayage ligne par ligne, font une approximation linéaire au niveau du segment ; cependant, il est difficile de définir le degré de l'approximation de cette méthode. La dimension de cette approximation est deux et elle est fort proche d'une approximation polygonale.

Rappelons ici le canevas de ces algorithme décrits dans [LANE 79] et [lane 80]. Ces algorithmes procèdent en deux étapes :

1. les morceaux de surfaces sont triés selon la valeur maximum qu'ils peuvent prendre en y,
2. lors du traitement de chaque ligne de balayage, les morceaux de surface qui naissent sur la ligne courante sont subdivisés jusqu'à ce que :

a. soit le sous-morceau créé n'intersecte pas la ligne courante, auquel cas il est mis dans la liste créée à la première étape,

b. soit il est suffisamment plat pour être affiché comme un polygone et il est mis dans la liste des polygones actifs.

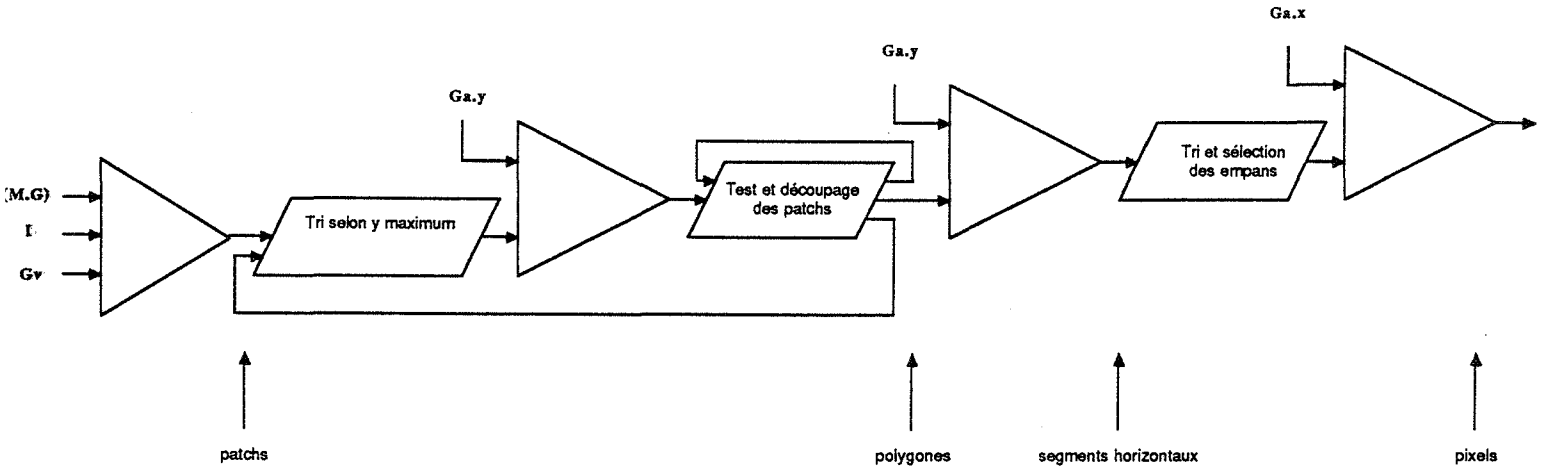


figure 4.12

[whit 80] et [SCWH 82] font une approximation des intersections des surfaces avec les plans de balayage par des ensembles de segments. Un des principaux problèmes de l'affichage de surfaces gauches est que les silhouettes ne sont pas nécessairement des frontières de surfaces dans l'espace de modélisation tridimensionnel. La méthode d'affichage de [LANE 79] ou [lane 80] contourne le problème en créant une approximation de dimension deux. En effet, pour cette approximation, les silhouettes sont nécessairement des frontières des sous-morceaux fabriqués. Ainsi toutes les courbes limitant la projection du morceau de surface sont approchées par des lignes brisées.

Pour éviter l'aspect irrégulier dû à ces lignes brisées, il faut déterminer les points d'appui des segments d'une façon plus précise.

La méthode de [whit 80] détecte une approximation de la courbe des points silhouettes en s'appuyant sur un découpage du morceau de surface en sous-morceaux presque plats. En ce sens, cette méthode est très proche de [lane 80]. La différence est que [whit 80] approche les morceaux de surface par des primitives que [SCHW 82] appellera des *polygones à bords courbes* (*curved-edge polygon*). En effet, ce sont aussi des surfaces réglées dont les droites génératrices sont orthogonales à l'axe des ordonnées, mais dont les frontières sont des courbes cubiques et non des lignes brisées. Le défaut principal de la méthode est qu'elle ne peut trouver des approximations des courbes silhouettes que si ces dernières coupent les frontières des sous-morceaux fabriqués. C'est toutefois une amélioration par rapport aux algorithmes de [LANE 79] et [lane 80], puisque ceux-ci se contentaient des silhouettes qui étaient les frontières des sous-morceaux.

L'algorithme de [SCHW 82] s'appuie sur les mêmes idées, mais il trouve, lui, tous les points de silhouette par des méthodes numériques et construit des *polygones à bords courbes* s'appuyant sur ces points et sur les points d'intersection du plan de balayage avec les frontières des morceaux de surface.

Les deux algorithmes terminent en résolvant numériquement l'équation décrivant l'intersection entre le *bord courbe* et la ligne de balayage, ce qui permet donc de trouver ensuite des couples (x,z) de points extrémités des segments qui peuvent enfin être triés selon les méthodes habituelles utilisées pour les polygones dans la méthode de Watkins après l'étape *Y*, comme nous l'avons décrit en 4.3.1.2. [whit 80] a choisi un tri en profondeur avant le tri sur les abscisses ; il génère donc une liste de priorité. Quant à [SCHW 82], il choisit de commencer par le tri sur les abscisses.

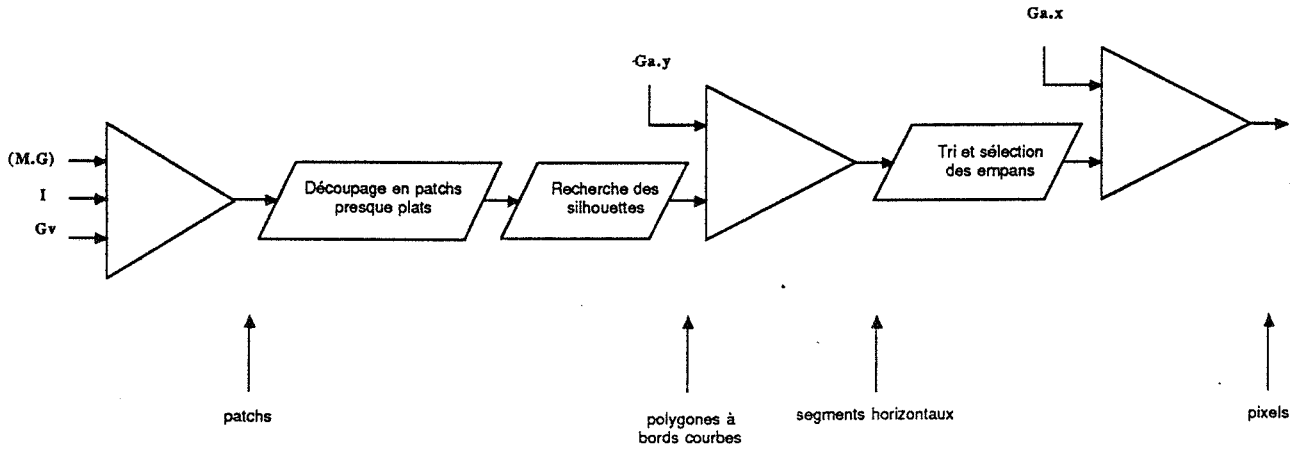


figure 4.13

[MAHL 72], [CATM 74] et [blin 80] ne font l'approximation qu'au niveau du pixel ; ils recherchent donc l'approximation maximum pour que l'affichage n'ait pas l'air d'avoir été obtenu par approximation. Plus de précision serait inutile, puisqu'elle ne serait pas visible sur l'image.

Ces trois algorithmes ont toutefois des approches différentes. En effet, [MAHL 72] travaille d'une façon exacte dans l'espace objet en effectuant ses calculs sur des représentations exactes (ie: à la précision flottante de la machine) des coniques intersections des surfaces quadriques avec les plans de balayage. Il travaille donc entièrement dans l'espace objet, et l'approximation linéaire n'est faite que pour l'affichage des pixels.

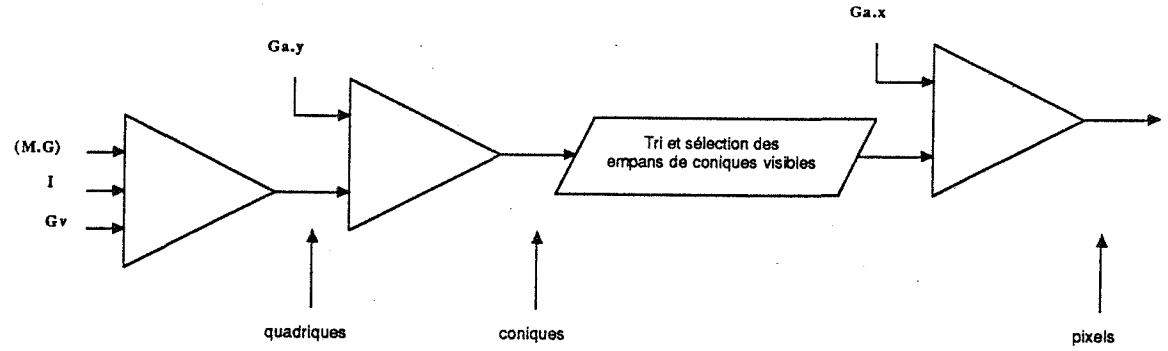


figure 4.14

[CATM 74] découpe récursivement les morceaux de surface qui lui sont fournis jusqu'à ce que ceux-ci ne recouvrent plus qu'un pixel ; l'élimination des parties cachées est réalisée grâce à une méthode de z-buffer.

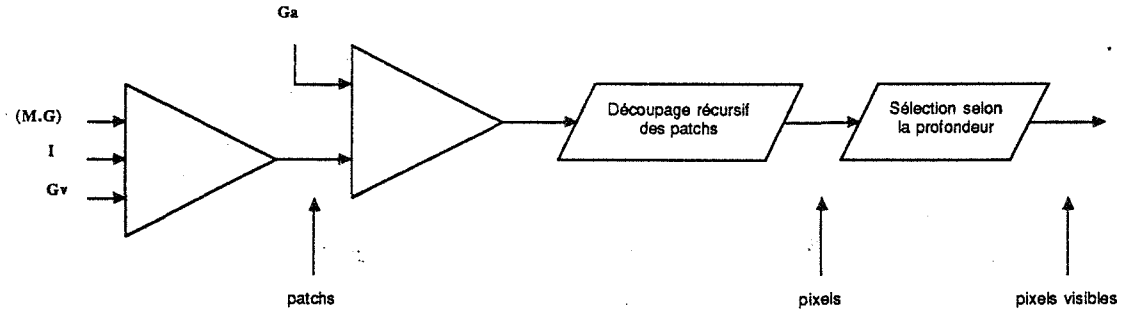


figure 4.15

Quant à [blin 80], il résoud les systèmes d'équations numériques permettant de trouver :

- les points d'intersection du plan de balayage avec les frontières du morceau de surface,
- les points de silhouette pour chaque ligne de balayage.

Cela définit pour la ligne de balayage une liste d'abscisses qui, une fois triée, donne une liste d'empans en couplant deux par deux les abscisses. Pour chaque point de coordonnées (x_0,y_0) de chaque empan, il termine en résolvant le système :

$f(u,v) = x_0$
$g(u,v) = y_0$

La détermination de (u,v) solution de ce système lui permet de calculer une profondeur pour le pixel de coordonnées (x_0,y_0). Cet algorithme se place donc définitivement dans l'espace image.

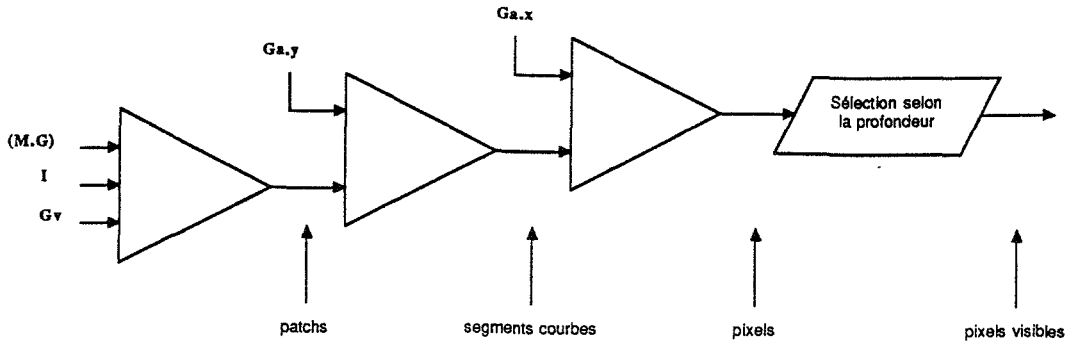


figure 4.16

Finalement, on peut résumer ces algorithmes de la façon suivante :

y[XZ] Lane et Carpenter, Schweizer et Cobb, Mahl,

y[ZX] Whitted,

yx[z] Blinn,

(xy)[z] Catmull.

Tous les algorithmes présentés, à l'exception de celui de Mahl, travaillent à un certain moment sur une approximation du premier degré des surfaces. De ce fait, l'algorithme de Mahl doit calculer les intersections d'objets dont les équations de représentation ne sont pas linéaires.

Il est à noter enfin qu'aucun algorithme travaillant sur les surfaces ne procède par génération de liste de priorité. Cela provient du fait qu'une surface, même toute seule, nécessite une élimination de parties cachées puisque certaines de ses parties peuvent en cacher d'autres. C'est ce qui entraîne d'ailleurs l'existence de lignes de silhouettes différentes des frontières de la surface.

4.3.4 Généralisation des algorithmes aux mélanges de surfaces

L'étude des algorithmes de visualisation pour les surfaces nous invite à penser que la composition doit être effectuée en pratique sur des objets géométriques de *petit* degré. *Petit* signifie en pratique égal à un pour les surfaces. En effet, parmi tous les algorithmes visualisant des surfaces, seul celui de Mahl travaille sur des éléments de degré strictement supérieur à un, degré qui dans ce cas vaut deux.

Il est par ailleurs assez facile d'en comprendre les raisons : éliminer les parties cachées conduit à résoudre des problèmes de tri soit dans l'espace de dimension trois, soit dans une de ses projections sur un espace de dimension deux. On est donc amené à résoudre des problèmes d'intersection lorsque le tri ne peut pas être fait d'une façon triviale dans une direction. Pour s'en rendre compte, il suffit de considérer une nouvelle fois l'algorithme de Newell, Newell et Sancha qui, lorsque la séparation de deux objets ne peut être réalisée dans la direction *Z*, découpe les polygones. Or, comme nous l'avons dit dans le chapitre 2 pour les calculs d'opérations booléennes, ces calculs d'intersections conduisent à résoudre des systèmes d'équations numériques dont le degré sera d'autant plus élevé que les surfaces ont des degrés élevés.

Nous allons donc étudier la composition d'objets hétérogènes lorqu'ils sont approchés par des éléments géométriques linéaires : polyèdres, polygones, segments de droites, points. Le rappel de ces quatre éléments nous ramène ainsi à l'approche que nous avons utilisée dans l'introduction des algorithmes d'élimination des parties cachées pour les polygones. Ensuite, nous décrirons une approche de composition qui fait intervenir une liste de priorité, qui, si elle n'est pas complètement générale, paraît cependant intéressante.

4.3.4.1 Composition par approximation

4.3.4.1.1 Approximation en dimension trois

Ces compositions fabriquent une scène tridimensionnelle qui est soit polyédrique, soit polygonale. Le premier cas correspond à une conversion de représentation approchée de modèles solides tels que nous les avons décrits au chapitre 2. Le deuxième cas ne diffère du premier que parce qu'il n'assure pas la construction d'une représentation cohérente d'un solide : il ne génère, par exemple, que les facettes vues de devant depuis le point de visée.

Dans les deux cas, le problème de visualisation se trouve être simplement un problème de visualisation de polygones. Les méthodes que nous avons citées sont donc immédiatement utilisables.

C'est donc une forme de composition très simple à mettre en oeuvre. Ses inconvénients sont ceux d'une facettisation *a priori* des objets. En particulier, le nombre de polygones engendrés pour que l'approximation polygonale soit suffisamment proche de la primitive originelle est souvent très grand.

Toutefois, les méthodes utilisées par [LANE 79] et [lane 80] peuvent permettre de trouver une solution au problème du stockage du grand nombre de polygones approximants : comme une *quasi-facettisation* est effectuée simultanément à un balayage ligne par ligne, tous les polygones d'approximation ne sont pas calculés à l'avance et la représentation concise initiale de la primitive est conservée

aussi longtemps que possible. Par contre, pour que ces méthodes soient utilisables, on doit pouvoir commencer par la première étape des algorithmes [LANE 79] ou [lane 80], c'est-à-dire qu'il faut être capable de trouver un majorant pour l'extension selon l'axe des ordonnées des objets. De plus, pour que la méthode soit efficace, ce majorant ne doit pas être trop éloigné de la borne supérieure. C'est le plus souvent possible grâce à l'utilisation de boîtes englobantes. Ce principe a été utilisé par [WHIT 81] avec également l'idée de permettre à des utilisateurs de créer des programmes de visualisation à partir d'une librairie de procédures. L'unification des objets se fait dans l'implantation de Whitted à deux niveaux. Pour l'affichage ultime des primitives, celles-ci doivent être approchées en fin de compte par des polygones convexes (ce sont bien ici de vrais polygones, contrairement à [LANE 80]). Cependant cette étape de conversion est simultanée au balayage en Y. Tout comme dans [LANE 79], [WHIT 81] procède en deux étapes :

1. les primitives sont triées selon la valeur maximum qu'elles peuvent prendre en y,
2. lors du traitement de chaque ligne de balayage, les primitives sont transformées :
   a. soit en primitives qui n'intersectent pas la ligne courante, primitives qui sont ajoutées à la liste créée à la première étape,
   b. soit en polygones. Ceux qui intersectent la ligne courante sont intégrés dans la liste des polygones actifs qui, elle-même, génère la liste des segments qui seront composés ; ceux qui ne coupent pas la ligne courante sont des primitives qui sont ajoutées à la liste créée à la première étape.

De plus, l'implantation de Whitted permet d'avoir comme sortie une structure qui décrit les segments à composer. On peut représenter cette méthode grâce au schéma de la figure suivante.

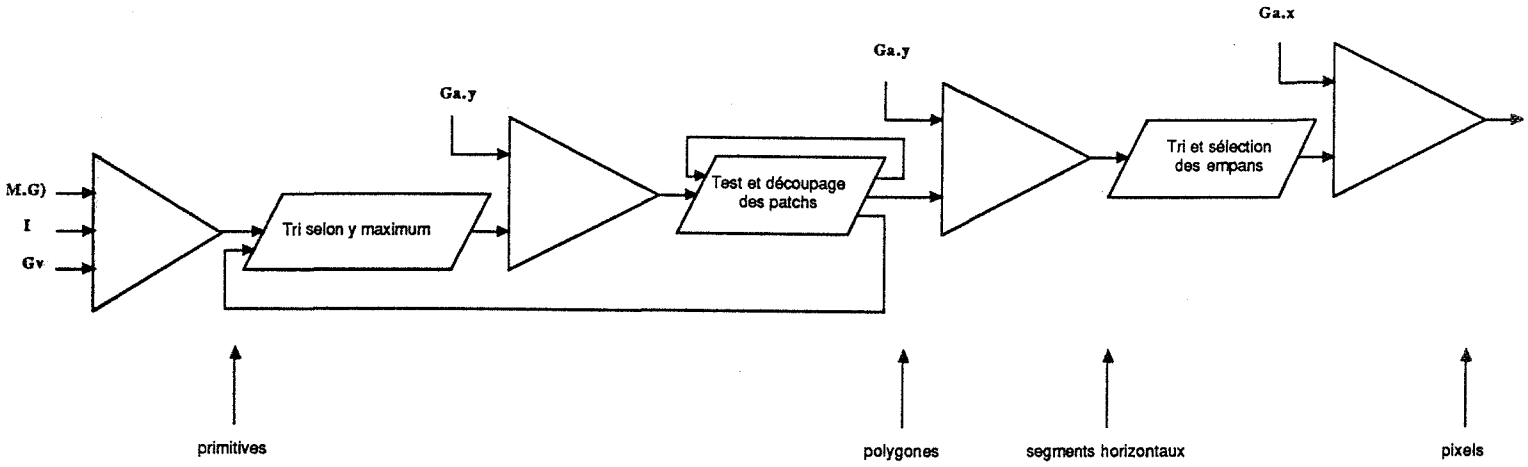


figure 4.17

4.3.4.1.2 Approximation en dimension deux
On retrouve ici le principe des méthodes par balayage ligne par ligne. Comme dans les algorithmes de [LANE 79], [lane 80], [whit 80] et [SCWH 82], on fabrique un certain nombre de segments qui sont ensuite composés. La détermination de la visibilité peut être faite avec une des méthodes données en 4.3.1.2.

4.3.4.1.3 Approximation en dimension un
La composition en dimension un s'effectue sur une droite et travaille sur des points. Cette méthode est donc simplement celle du z-buffer. Décrite pour les polygones, cette méthode se généralise en fait sans difficulté aux autres primitives : pour chaque objet, il faut être capable de calculer la position en profondeur d'un point dont on connaît la projection sur l'écran. C'est ici que l'on va retrouver la similitude entre la méthode du z-buffer et le lancer de rayon puisque chacun d'eux compose à ce niveau. La différence est l'ordre dans lequel sont traités les objets d'une part et l'image d'autre part.

Le z-buffer traite les objets un par un et mémorise l'information de profondeur pour chaque pixel. Le lancer de rayon traite tous les objets simultanément, mais les pixels un par un.

En résumé, il n'y a pas de difficulté à composer au niveau du pixel. Cependant des problèmes surgissent lorqu'il s'agit d'antialiasser les images produites avec cette méthode. Nous reparlerons des méthodes permettant de résoudre ces problèmes dans la section consacrée à l'antialiassage.

4.3.4.2 Composition par liste de priorité
Comme nous l'avons indiqué en 4.3.3, une composition de surfaces par liste de priorité n'est pas possible. Il peut donc paraître curieux de proposer ce procédé comme un moyen de composition encore plus général. C'est pourtant ce que fait [CROW 82]. Un graphe orienté décrit les priorités et les dépendances des objets les uns par rapport aux autres. Les nœuds du graphe représentent les objets et les arcs les relations de dépendance. Lorque le tri préliminaire en profondeur sur les centres des sphères englobantes n'est pas suffisant pour séparer les objets, ou qu'un circuit est détecté dans le graphe, il y a raffinement en testant successivement :

- les projections des sphères englobantes,
- les projections des boîtes englobantes,
- les sphères englobantes,
- les boîtes englobantes.

Dans le cas où tous ces tests échouent, les objets sont considérés comme inséparables.

L'image complète peut alors être fabriquée en confiant à plusieurs processus distincts la génération d'un objet, ou d'un groupe d'objets inséparables. La composition finale empile toutes ces images intermédiaires selon la priorité déterminée par le graphe.

Le problème général de composition se retrouve donc en fin de compte pour ces processus intermédiaires lorsqu'ils doivent traiter plusieurs primitives différentes. Le dernier recours dont dispose l'algorithme de [CROW 82] est l'approximation polygonale...

Nous reparlerons de cette approche dans la dernière section de ce chapitre, section consacrée elle aussi à la composition mais après les considérations que nous voulons donner sur le rendu des images de synthèse et sur l'antialiassage de ces mêmes images.

4.3.5 Visualisation d'arbres CSG

La modélisation par arbre CSG a été décrite en 2.3.3. Etant donnés d'une part son importance, et d'autre part le fait que notre langage de modélisation utilise ce schéma de représentation, nous allons décrire ici les algorithmes de visualisation dédiés à cette modélisation. Ils peuvent se classer en trois catégories suivant la dimension de l'espace dans lequel ils évaluent les opérations booléennes. Nous commencerons par la dimension un parce que les problèmes y sont les plus simples, puis nous verrons la dimension deux et la dimension trois.

4.3.5.1 Evaluation en dimension un

L'evaluation des opérations booléennes s'effectue sur une droite. Les algorithmes d'élimination des parties cachées que nous avons étudiés et qui composent en dimension un sont le z-buffer et le lancer de rayon. Si pour l'élimination des parties cachées des surfaces ils devaient composer des points, pour une évaluation de visibilité sur un arbre CSG, il doivent composer des segments. Chacun des segments correspond à l'intersection du rayon avec un (morceau de) solide. Le principe de cette évaluation a été exposé dans [ROTH 82] et il peut être résumé par la figure suivante (figure extraite de [ROTH 82] page 121) :

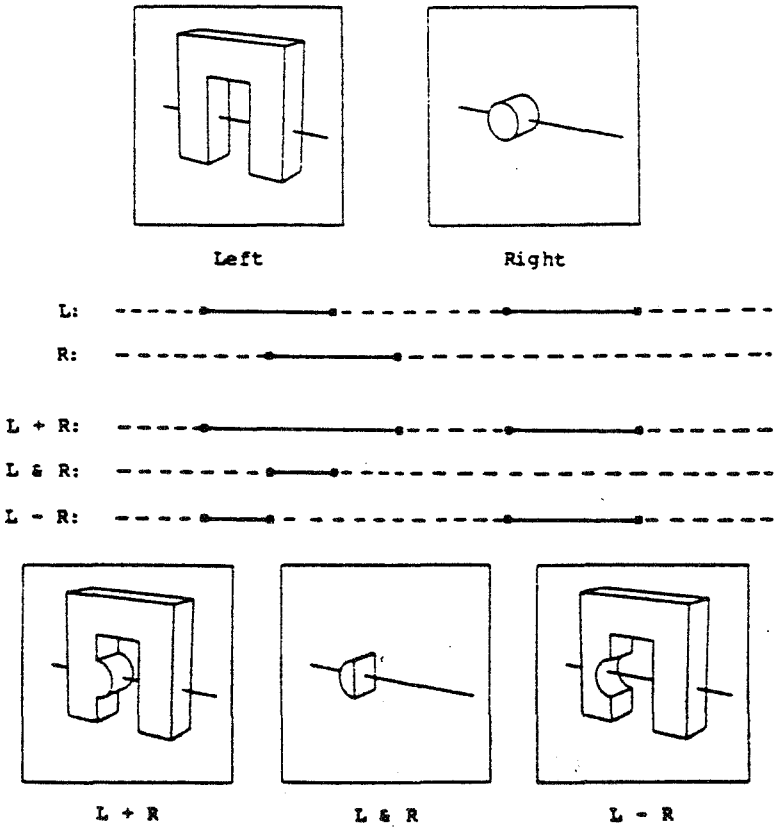


figure 4.18

Il se pose par contre le problème de la représentation et du stockage des segments nécessaires à cette évaluation. Le lancer de rayon n'étudiant qu'un pixel à la fois, il passe en revue tous (aux optimisations près) les objets et fabrique une liste triée par point d'entrée des segments. Le problème d'une généralisation de la méthode du z-buffer vient de ce que l'on considère les objets les uns après les autres. Une première méthode utilisée par [OKIN 84] et [HOOK 86] consiste à conserver, pour chaque pixel, la liste qui était construite avec la méthode du lancé de rayon. L'ensemble de ces listes doit être mise à jour lors du traitement de chaque nouvel objet. Le z-buffer doit donc pouvoir disposer, lors de sa généralisation pour la modélisation par arbre CSG d'une allocation dynamique de mémoire. De plus, la

taille de la mémoire qu'il est nécessaire d'allouer est beaucoup plus importante que la taille de celle nécessaire au stockage de la profondeur de chaque pixel pour la méthode ordinaire du z-buffer.

Une autre approche est proposée dans [ROSS 86]. Le principe est décrit dans l'algorithme suivant :

```
pour chaque face de chaque primitive solide
    pour chaque point P d'une grille dense sur la face courante †
        si la profondeur de la projection du point P est plus petite que celle conservée dans le z-buffer
            si le point P est sur la surface du solide CSG *
                mettre à jour couleur et profondeur
```

Le cœur de l'algorithme d'évaluation réside dans le test de classification (marqué d'un astérisque) du point par rapport à la surface du solide. Cette classification est réalisée selon les principes décrits dans [TILO 80].

Cette deuxième méthode présente l'important avantage par rapport à la précédente de ne pas nécessiter d'allocation dynamique de mémoire, ni même plus de mémoire que le simple algorithme du z-buffer. Par contre, la boucle sur la grille *dense* de points (marquée d'une dague) peut être :

- soit trop serrée, et on évalue plusieurs fois le même pixel ;
- soit trop lâche, et on laisse des trous dans les surfaces visualisées.

Citons enfin [GOLD 86] dont la méthode se rapproche de celle que nous avons étudiée en premier. Le stockage des listes de segments est évité par la combinaison de deux techniques :

- la réorganisation de l'arbre en un arbre *normalisé*, c'est-à-dire un arbre de construction qui possède les deux propriétés suivantes :
  - toute branche droite d'une intersection ou d'une différence doit être une primitive,
  - toute sous-arbre situé en dessous d'une intersection ou d'une différence ne doit contenir aucune union ;
- le recalcul des segments pour chacune des primitives chaque fois que nécessaire plutôt que leur stockage.

La deuxième technique n'est utilisable que parce que [GOLD 86] utilise une machine câblée dans laquelle le calcul des points d'entrée et de sortie du rayon dans chaque primitive est très rapide. Pour une implantation logicielle, ce recalcul doit être remplacé par un stockage et on retrouve le problème de la très grande taille mémoire nécessaire.

4.3.5.2 Evaluation en dimension deux
Nous nous plaçons donc dans un plan. La méthode d'évaluation va donc être une extension de l'algorithme de composition de Watkins. Ce dernier compose des segments dans un plan, son extension, décrite dans [ATHE 83], compose des polygones dans le plan de balayage. L'algorithme de Watkins tire parti de la cohérence pour ne pas faire une sélection en profondeur sur chaque pixel ; il n'effectue cette sélection que sur les extrémités des segments composés. De même, l'algorithme d'Atherton tire parti de la cohérence pour ne pas faire l'évaluation booléenne sur chaque pixel ; il ne la fait que lorsqu'il y a intersection de polygones ou entrée dans un nouveau polygone (ces polygones sont ici les polygones intersections des solides polyédriques avec le plan de balayage).

4.3.5.3 Evaluation en dimension trois
Ces méthodes doivent trouver les surfaces frontières des objets solides représentés par un arbre de construction CSG. Ce sont donc les méthodes de conversion de représentation dans le schéma de représentation par frontières. Nous avons cité les méthodes dans les sections 2.4 et 2.5.

4.4 LE RENDU

Nous donnerons dans ce paragraphe quelques indications sur les formules de calculs d'éclairement les plus utilisées. Nous ne voulons pas entrer dans le détail des nombreuses publications sur ce sujet, mais nous voulons montrer comment ces méthodes de calcul peuvent se combiner aux algorithmes d'élimination des parties cachées. Nous avons utilisé pour ce paragraphe les éléments figurant dans [FOLE 82] et [MAGN 85].

4.4.1 Les éléments nécessaires aux calculs du rendu

La réflexion diffuse est la caractéristique des surfaces parfaitement mates. Une telle surface disperse la lumière qu'elle reçoit uniformément dans toutes les directions, si bien qu'une telle surface est vue de la même couleur quelle que soit la direction depuis laquelle on la regarde. Pour ces surfaces, on utilise la loi de Lambert qui donne la quantité de lumière réfléchie (dans toutes les directions) en fonction du cosinus de l'angle θ entre le vecteur unitaire **L** dirigé depuis le point de la surface considérée vers la source lumineuse et le vecteur unitaire **N**, normal à la surface en ce point :

$$I_d = k_d \cos(\theta) = k_d (\mathbf{L} \cdot \mathbf{N}) \qquad (1)$$

L'effet obtenu est très dur et ressemble à celui donné par un flash dans le noir absolu. En fait, dans les situations usuelles, une lumière ambiante baigne tous les objets indépendamment de leur position. Cette lumière est le résultat de multiples réflexions de la lumière sur les différents objets ; ce qui fait que mêmes les objets qui ne sont pas directement éclairés par le flash (si l'on réutilise notre comparaison) sont visibles. Cette lumière ambiante est modélisée par une constante qui vient s'ajouter à la lumière diffuse de la formule précédente :

$$I = I_a + I_d = I_a + k_d (\mathbf{L} \cdot \mathbf{N}) \qquad (2)$$

Pour que deux polygones parallèles et de la même couleur intrinsèque soient distinguables, on peut introduire la distance r entre l'œil et le point de la surface :

$$I = I_a + I_d = I_a + k_d (\mathbf{L} . \mathbf{N}) / (k + r) \qquad (3)$$

Cette formule est encore insuffisante pour modéliser les surbrillances dues à la réflexion spéculaire. Aussi [PHON 75] a proposé le modèle empirique suivant :

$$I = I_a + I_d + I_s = I_a + (k_d (\mathbf{L} . \mathbf{N}) + k_s (\mathbf{R} . \mathbf{V})^n) / (k + r) \qquad (4)$$

où **R** désigne le vecteur symétrique du vecteur **L** par rapport à la normale **N** (c'est la seule direction où la lumière serait réfléchie si la surface était un miroir parfait), **V** désigne le vecteur dirigé vers le point de visée, et enfin n est un coefficient indiquant le degré de réflexion de la surface (n devrait être infini pour un miroir parfait).

Ces quelques éléments nous suffisent pour déterminer quelles sont les informations géométriques qui doivent être disponibles pour chaque point de la surface affichée, ce sont :

- pour les formules (1) et (2) et dans le cas où la source lumineuse est située à l'infini (et donc le vecteur **L** est indépendant du point de la surface) :
  - la normale **N** à la surface,
- pour tous les autres cas :
  - la normale **N** à la surface,
  - les coordonnées du point de la surface.

4.4.2 Le processus de rendu par rapport aux processus de composition

Ainsi, on comprend l'intérêt de disposer (au moins) de la normale pour chaque pixel que l'on affiche. On voit aussi qu'une quantité très importante de calculs doit être effectuée dans cette étape de calcul de rendu puisque chaque pixel doit être traité séparement. Aussi est-il très intéressant de tirer parti de la cohérence de l'image à ce niveau. La première étude sur ce sujet est celle de [GOUR 71] qui présente la méthode connue sous le nom de lissage de Gouraud. En fait, deux problèmes sont mélangés dans cette étude :

- d'une part, la détermination d'une normale (approximative) sur chacun des sommets d'un réseau de polygones connus seulement par les coordonnées de leurs sommets et leurs relations de voisinage ;
- et d'autre part, l'interpolation linéaire sur la surface d'un polygone d'un paramètre dont on ne connaît les valeurs que sur les sommets.

La première étape permet d'utiliser une des lois d'éclairement citées dans la section 4.4.1 pour calculer une couleur sur les sommets. Cependant, en général, ces réseaux de polygones sont issus de la facettisation d'une surface de degré supérieur et il est alors plus simple de transmettre les composantes des véritables vecteurs

normaux (c'est-à-dire les vecteurs normaux à la surface d'origine avant la facettisation) avec les coordonnées des sommets du polygone.

La deuxième partie de [GOUR 71] nous intéresse plus ici. Sa méthode d'interpolation est combinée avec le remplissage de polygone que nous avons décrit en 4.3.1 ; de la sorte elle permet d'exploiter à fond la cohérence en faisant les calculs d'interpolation d'une manière incrémentale. Originellement, elle était utilisée pour interpoler une couleur, mais elle est bien sûr utilisable pour interpoler toute grandeur linéaire.

Cette méthode a été généralisée dans [PHON 75] pour l'interpolation de la normale. Cette interpolation nécessite un calcul de racine carrée pour chaque pixel et est donc très coûteuse en temps de calcul. Aussi [DUFF 79] a proposé une amélioration de l'efficacité en combinant cette interpolation de la normale avec le calcul de l'éclairement proposé par [PHON 75], mais il ne me semble pas très intéressant de mêler ces deux étapes. [BISH 86] propose lui aussi d'autres techniques pour réduire au maximum les calculs à effectuer sur chaque pixel en mêlant l'étape de calcul de la normale et celle de calcul du rendu. Toutefois, ses idées sont exploitables pour calculer une normale approchée sans avoir à calculer une racine carrée pour chaque pixel.

Nous allons donc pouvoir séparer les algorithmes d'éliminations des parties cachées pour des scènes polygonales en deux classes par rapport au calcul de rendu entre ceux qui peuvent exploiter les idées incrémentales abordées ci-dessus et les autres. Cela revient à séparer les algorithmes qui balayent les polygones *entiers* tels qu'ils les reçoivent ; ce sont donc les algorithmes du z-buffer, de Watkins et de Schumacker. Pour ces algorithmes, les valeurs des attributs à interpoler sont connues sur tous les sommets. L'algorithme de Newell n'a à découper les polygones que dans des cas assez rares où il y a occlusion mutuelle de plusieurs polygones. Ainsi, il peut presque être considéré comme faisant partie de cette catégorie.

Par contre, les algorithmes de Warnock, d'Atherton, de Michelucci et de Fuchs doivent calculer les nouvelles valeurs d'attributs sur les sommets des nouveaux polygones qu'ils fabriquent en coupant les polygones qu'ils reçoivent. Ceci les pénalise fortement en temps de calcul lorsque le nombre de découpes devient important comme c'est le cas pour une scène courante même de complexité moyenne.

4.5 L'ANTIALIASSAGE

L'antialiassage des images est une opération qui est maintenant indispensable. On ne peut en effet plus montrer des images où les segments sont représentés avec des marches d'escalier et où les frontières des polygones sont déchirées. C'est pourquoi je vous montre une telle image sur la photo 4.1. Nous ne donnerons pas ici de support théorique à l'antialiassage, support qui se base essentiellement sur le traitement du signal, l'échantillonnage et le filtrage. Ces informations peuvent se trouver dans [GHAZ 85], où elles sont appliquées à la synthèse d'images.

Nous ne retiendrons de [GHAZ 85] que la nécessité d'avoir sur chaque pixel plus d'information que celle qui consiste à connaître la couleur projetée au *centre* du pixel par un objet. En fait, il faut choisir une méthode de filtrage. Filtrer dans le domaine spatial consiste à effectuer le produit de convolution entre l'image de résolution infinie et une fonction qui est la réponse impulsionnelle du filtre considéré. En pratique cette fonction a un support borné que l'on appelle *fenêtre*. Par abus de langage, on appelle également *fenêtre* la fonction elle-même.

On peut choisir par exemple une fenêtre de Fourier, c'est à dire la fonction caractéristique du pixel (*pixel* qui est considéré ici comme la surface d'un carré) sur lequel on veut antialiasser. Une méthode exacte consisterait à calculer l'intégrale de la lumière réfléchie par chacun des points visibles depuis le pixel. Le *filtre* utilisé est dans ce cas un *filtre passe-bas idéal* qui malgré son nom ne l'est pas dans cette application. En effet, un tel filtre a une réponse en fréquence avec une coupure nette, ce qui n'est pas parfait et ne correspond pas au filtrage effectué par le faisceau d'électrons d'une caméra de télévision pour laquelle il y a un recouvrement entre les fenêtres utilisées pour le calcul des intégrales (intégrales "calculées" analogiquement par la caméra) pour des pixels voisins. La figure suivante montre une représentation graphique des fenêtres du filtre passe-bas idéal et de celui d'une caméra.

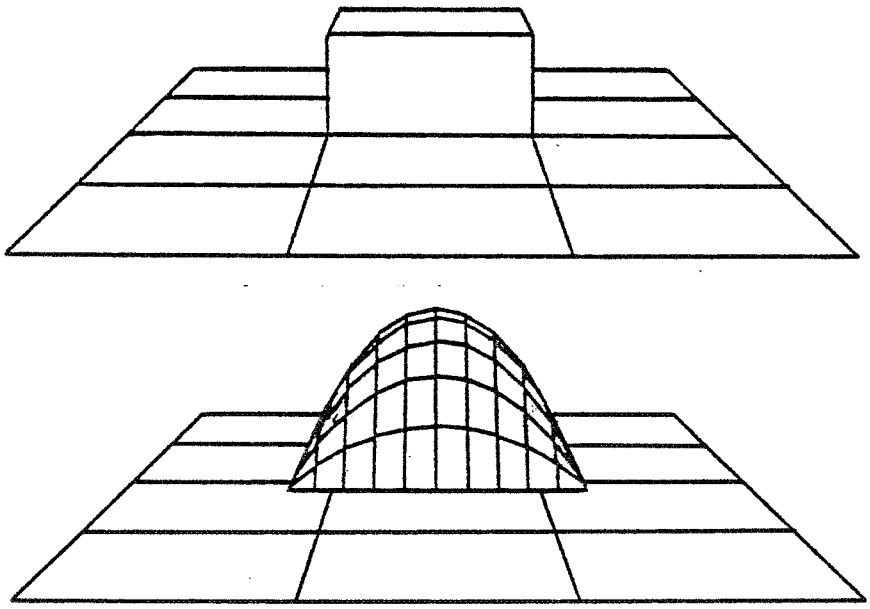


figure 4.19

Malgré tout, nous garderons dans la suite l'usage d'un tel filtre, tout comme d'ailleurs de nombreux algorithmes d'antialiassage : [PITT 80], [FUJI 83], [CARP 84], [PORT 84], [DUFF 85].

Une autre difficulté est de trouver dans quel espace doit se calculer l'intégrale. En effet, si on ne considère que des images en niveaux de gris, où seule intervient l'intensité lumineuse, on a une linéarité par rapport à la grandeur physique mesurable. Par contre, lorsque l'on considère des images en couleur, doit-on faire les calculs dans l'espace Rouge-Vert-Bleu, ou dans l'espace Teinte-Saturation-Intensité,... ou dans un autre? La réponse à cette question semble pour le moment assez empirique [TURK 86]. Nous supposerons, pour notre part, qu'un tel espace existe et que nous pouvons donc ajouter des couleurs,...

Une première approximation est destinée à simplifier le calcul de cette intégrale. Elle consiste à considérer que les projections des objets ont une couleur uniforme sur la surface d'un pixel. De cette façon, le calcul de l'intégrale se ramène à un calcul d'aire ; à savoir l'aire de la partie visible de la projection de chacun des objets dans le pixel. Ainsi, si pour le pixel représenté sur la figure 4.20, nous connaissons les trois couleurs c_1, c_2 et c_3 et les aires A_1, A_2 et A_3 qu'elles occupent sur le pixel, le calcul de la couleur du pixel s'effectue avec la formule $A_1c_1 + A_2c_2 + A_3c_3$.

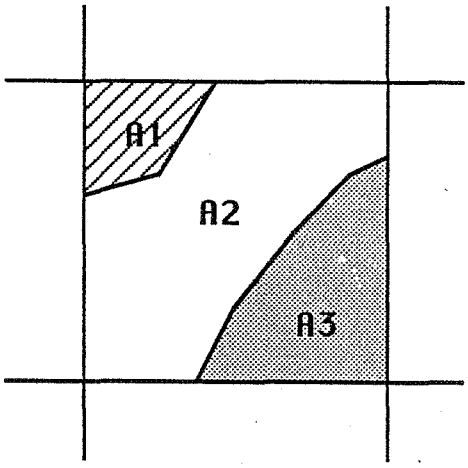


figure 4.20

Le nombre d'objets dont la projection intersecte le pixel considéré étant fini, l'intégrale est donc approchée par une moyenne. Chaque intensité est multipliée par l'aire de la surface qu'elle occupe, tous ces produits sont ensuite ajoutés. Il n'y a pas de division à faire si l'on prend l'aire d'un pixel comme unité de surface.

Formulé en terme d'aires, le problème de l'antialiassage ressemble à un problème d'élimination des parties cachées puisqu'on doit déterminer quelles sont les aires des projections des objets visibles. Cette remarque va nous permettre de voir quelles doivent être les conditions sur un algorithme d'élimination des parties cachées pour qu'on puisse l'améliorer en y intégrant un traitement d'antialiassage.

Pour antialiasser, il faut pouvoir disposer d'informations suffisamment précises au niveau du pixel pour calculer les aires voulues. Une première méthode tout à fait universelle consiste à *suréchantillonner* l'image. Cela signifie que l'image est calculée avec une résolution plus grande, multiple de la résolution désirée. Un filtrage par moyennage est ensuite appliqué sur chaque groupe de pixels de l'image suréchantillonnée correspondant à un pixel de l'image voulue.

Cette méthode, bien que pouvant s'appliquer à tous les algorithmes, n'est cependant pas sans poser des problèmes. En effet, le temps de calcul se trouve être multiplié au moins par le facteur de suréchantillonnage et par ailleurs la place mémoire, qui est déjà grande pour stocker une image "normale", s'en trouve agrandie d'autant. Toutefois, les méthodes par balayage de ligne n'ont besoin de mémoriser que les pixels de l'image suréchantillonnée qui sont sur la ligne de balayage courante à la résolution voulue. De même, la méthode du lancer de rayon qui travaille pixel par pixel n'a besoin de conserver l'image suréchantillonnée que sur un pixel et il n'y a pour lui aucun problème de place mémoire. Le problème de la taille mémoire ne se pose donc pas pour les algorithmes travaillant ligne par ligne ou pixel par pixel.

Le suréchantillonnage est à y bien regarder une méthode *image* d'antialiassage, *image* dans le sens *espace-image* de la classification des algorithmes d'élimination des parties cachées.

Complètement opposé à cette méthode, essayons de définir ce que pourrait être un algorithme d'antialiassage dans l'espace objet : cette fois, l'information de surface ne doit plus être calculée par une approximation par un suréchantillonnage, mais par un calcul travaillant sur les coordonnées *exactes* (c'est-à-dire à la précision de la machine) des objets. Ainsi, sur chaque pixel, on doit trouver quel est le morceau de polygone visible dans l'espace objet. Si l'on reprend alors notre liste d'algorithmes d'élimination des parties cachées donnée en 4.3.1.2, on peut voir que seuls les algorithmes d'Atherton et de Michelucci travaillent entièrement dans l'espace objet et donnent comme résultat la liste des polygones visibles. En effet, le dernier algorithme qui travaille entièrement dans l'espace objet est celui de Schumacker, mais ce dernier ne donne qu'un ordre d'affichage des polyèdres, et non la liste de ce qui est visible après projection. Ce sont donc typiquement les algorithmes d'Atherton et de Michelucci qui permettent ce calcul exact d'aire. Est-ce à dire que les autres algorithmes ne peuvent pratiquer d'autres techniques d'antialiassage que le suréchantillonnage ?

C'est vrai sans aucun doute pour la méthode de Warnock : en effet, cette méthode travaille entièrement dans l'espace image et découpe l'écran jusqu'à ce que la situation soit "simple". Arrivé au niveau du pixel, pour évaluer les aires, la méthode naturelle, prolongement du découpage qui a précédé, consiste à continuer ce découpage de façon à "resserrer" l'escalier autour du segment de séparation. Ce découpage récursif permet de définir une suite $(s_n)_{n\in\mathbf{N}}$ (cf. figure 4.21) pour chaque polygone intersectant le pixel par le rapport du nombre de sous-pixels entièrement vus pour chaque morceau de polygone par le nombre de sous-pixels nécessaires au recouvrement complet d'un pixel. Cette suite $(s_n)_{n\in\mathbf{N}}$ converge vers l'aire voulue.

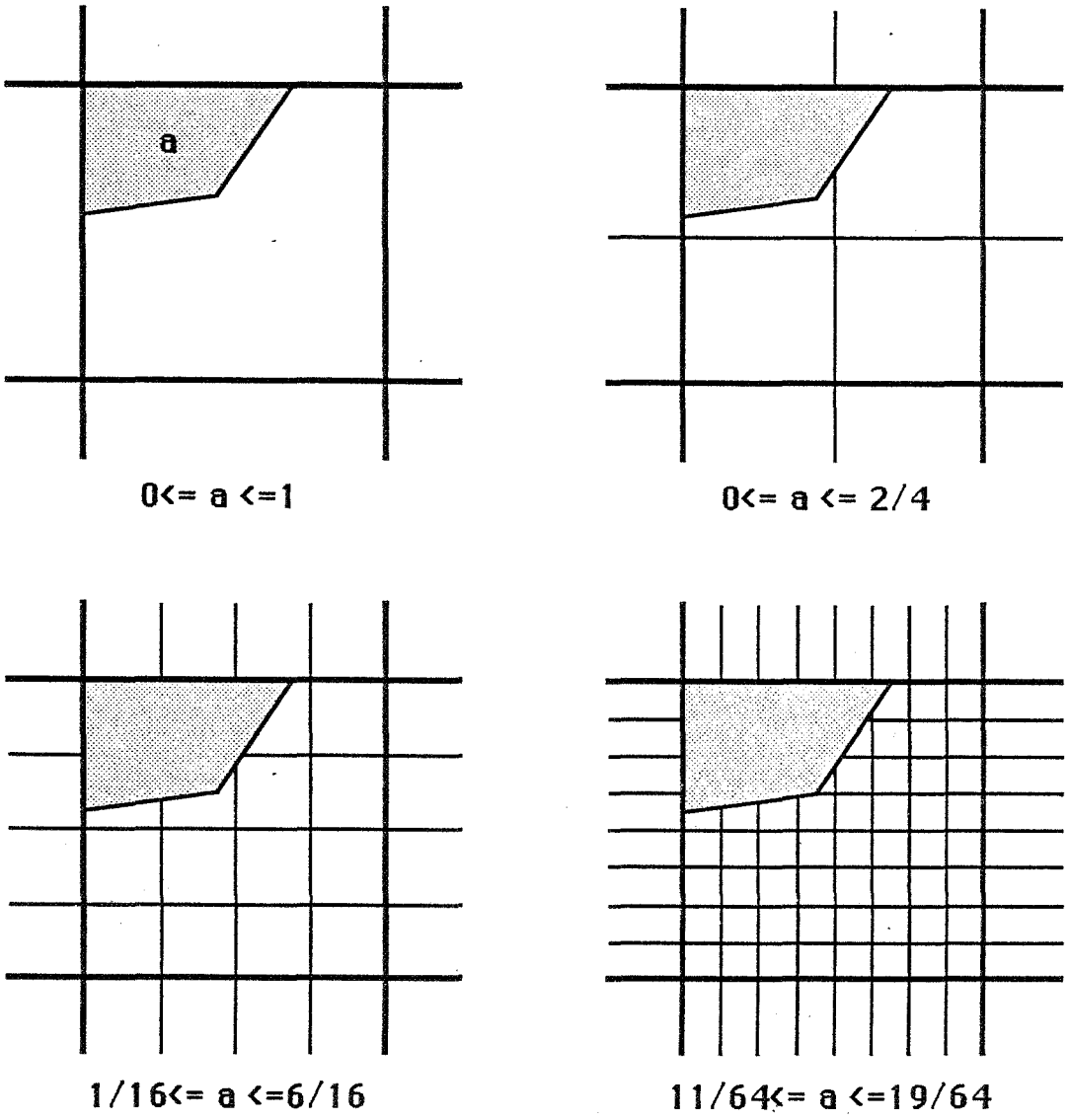


figure 4.21

Les algorithmes de Newell, Schumacker et du z-buffer ont un point commun : lors de l'affichage, ils ne traitent qu'un polygone à la fois. Cela est une source de difficultés dans un traitement d'antialiassage. En effet, un problème va surgir lorsque deux polygones partagent un bord. Cela peut se produire soit lorsqu'ils sont adjacents soit lorsqu'ils se recouvrent. Un algorithme d'antialiassage qui n'a d'informations que sur le polygone qu'il est en train d'afficher travaille en fait en deux dimensions. Examinons déjà quels sont les problèmes de l'antialiassage de polygones affichés un par un.

Nous n'aborderons pas dans ce cas le problème du calcul des aires. Pour cela, on pourra consulter [GHAZ 85] qui couvre les aspects théoriques et donne les références des méthodes et des articles traitant ce sujet.

Dans le cas où un seul polygone est affiché sur un fond, l'antialiassage se réduit aux calculs d'aires et de moyennes des couleurs. Par contre, lorsque plusieurs polygones arrivent successivement, les difficultés commencent. En effet, pour deux polygones qui partagent un côté, les pixels qui appartiennent à ce côté commun ne devraient être traités qu'une fois en faisant un mélange des couleurs des deux polygones. Cependant comme ceux-ci arrivent l'un après l'autre, un premier traitement d'antialiassage est effectué entre le premier polygone et le fond. Lorsque le deuxième polygone est affiché, un traitement est effectué en mélangeant la couleur du deuxième polygone avec la couleur présente des pixels, soit une couleur qui est la couleur du premier polygone mélangée avec la couleur du fond. Ainsi dans le résultat final, ces pixels qui intersectent les deux polygones ont une couleur qui est une moyenne faisant intervenir les couleurs des deux polygones, mais aussi la couleur du fond.

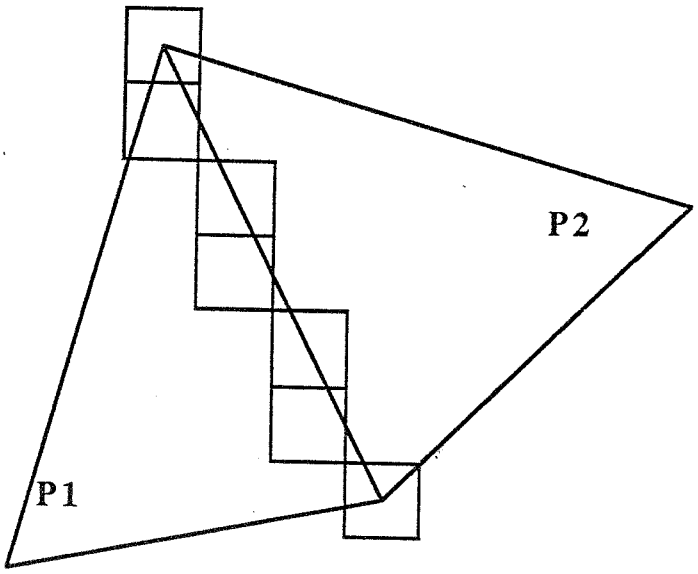


figure 4.22

Un palliatif, mais qui ne donne pas toutefois une solution exacte, consiste à lire la couleur du premier polygone sur un pixel voisin de celui que l'on est en train de traiter, ce pixel ayant souvent une valeur égale ou très voisine de la valeur correcte. Une telle méthode est alors utilisable pour les algorithmes d'élimination des parties cachées de Newell et de Schumacker. Pour que cette couleur lue sur le pixel voisin de celui que l'on est en train de traiter soit correcte, il faut cependant que l'on ne considère que des polygones suffisamment épais pour qu'ils couvrent plus d'un pixel.

Lorqu'un seul objet partage un pixel avec le fond, une seule aire suffit pour calculer la moyenne des couleurs, puisqu'alors on connaît aussi bien l'aire occupée par l'objet que l'aire occupée par le fond. Cette unique aire est la seule qui soit utilisée dans le cas que nous venons d'exposer. On sait donc déjà que cette information est insuffisante pour réaliser un traitement d'antialiassage exact. Toutefois, il est intéressant de stocker cette information avec une image qui représente un objet puisque la méthode que nous venons d'exposer permet de faire un traitement approché avec une méthode d'élimination des parties cachées par liste de priorité. Cette possibilité a été exploitée par [PORT 84].

[PORT 84] propose une représentation des images qui conserve en plus des trois composantes rouge, verte et bleue de la couleur, une information nommée le *canal alpha* (*alpha channel*), qui indique quel est le taux de recouvrement du pixel par cette couleur. Cette information lui permet d'antialiasser les superpositions d'images. Toutefois pour que sa méthode fonctionne, il faut que les différentes images soient empilées les unes sur les autres. Il faut donc en fait faire un tri analogue à celui utilisé pour la méthode d'élimination des parties cachées de Newell, Newell et Sancha.

Pour pouvoir utiliser cette idée de façon à en faire une méthode de composition pour les différentes primitives, il faut donc que deux conditions soient remplies :

- pouvoir trier les différentes primitives suivant un ordre d'affichage des plus lointaines vers les plus proches,

- transformer les primitives en images qui utilisent cette représentation ; c'est le rôle d'un algorithme d'antialiassage plus simple (dans le sens où il ne travaille pas sur l'image complète), lequel dispose nécessairement de l'information *taux de recouvrement des pixels.*

Pour que l'erreur qui est commise lors de l'affichage superposé de bords communs soit minimisée sur l'image, la solution qu'il propose, (au moins dans les exemples qu'il donne) est de ne pas avoir de corrélations entre les objets qui partagent les pixels. Cela revient à dire qu'il faut essayer de ne pas avoir d'objets adjacents, ni d'objets se recouvrant suivant une même arête ; cette condition est à rapprocher de la définition des *cluster*s de la méthode d'élimination des parties cachées de Schumacker.

Dans la méthode précédente, la liste de priorité des objets est produite à la main. Cette méthode est donc plus un outil interactif de composition d'images qu'un algorithme général d'élimination des parties cachées avec antialiassage. Duff a donc réutilisé cette représentation des images dans [DUFF 85] en y ajoutant une information de profondeur pour chaque pixel pour permettre de faire l'élimination des parties cachées avec une méthode de type z-buffer, tout en conservant les mêmes idées que [PORT 84] pour le traitement d'antialiassage. Cet algorithme détecte les intersections d'images (ou de polygones) ou leur bord commun en testant la profondeur non sur le centre du pixel mais sur ses quatre sommets. Ainsi lorsque les quatre sommets les plus proches appartiennent à la même image, l'affectation de la couleur et de la profondeur sont ceux de cette image, sinon on est dans un des cas suivants :

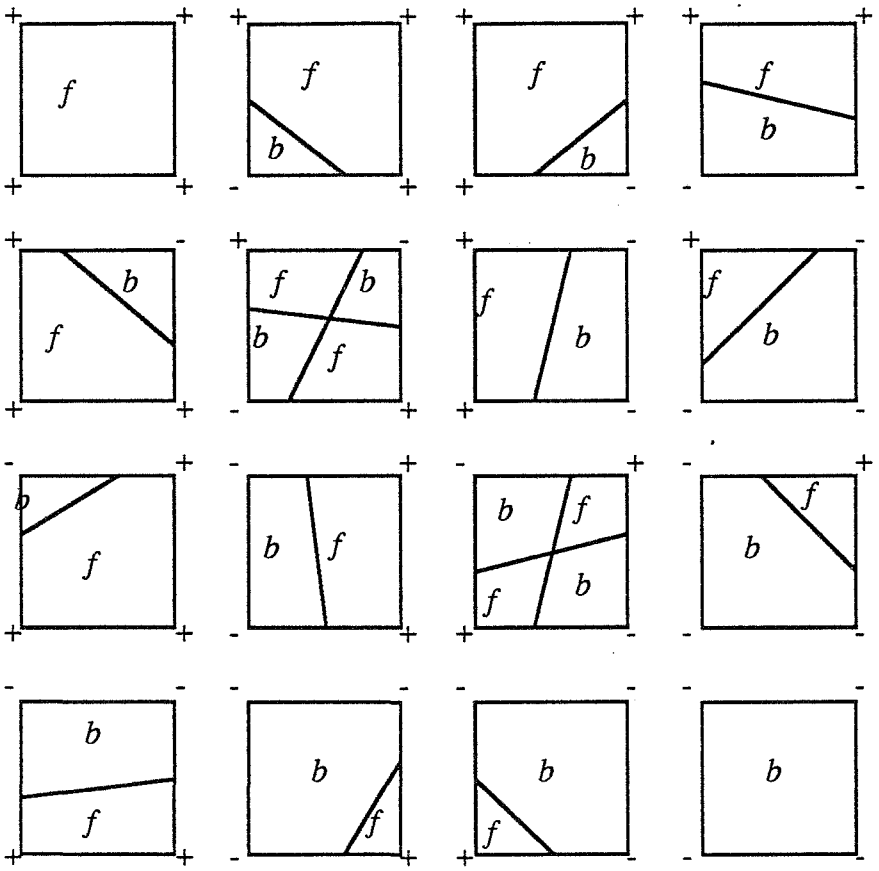


figure 4.23

Les taux de recouvrement des deux images permettent alors de calculer une valeur de couleur pour l'image résultat. Cette méthode tombe en défaut

- lorsque les deux images sont corrélées (ce qui revient à dire lorsque deux polygones partagent une arête),
- lorsque, sur un pixel, il y a plusieurs fois un des cas décrits par la figure 4.23.

Carpenter a publié dans [CARP 84] un algorithme qu'il a appelé A-buffer et qui combine l'antialiassage avec la méthode d'élimination des parties cachées du z-buffer. Dans cet algorithme, deux mots d'information sont nécessaires pour chaque pixel. Le premier est utilisé pour conserver la profondeur et le deuxième est utilisé soit pour stocker la couleur du pixel lorsqu'un seul objet le traverse, soit un pointeur sur une liste. Chaque élément de cette liste décrit un des objets intersectant le pixel. Ainsi [CARP 84] élimine la principale difficulté de l'antialiassage avec la méthode du z-buffer, à savoir que la représentation des données qu'utilise cette méthode est échantillonnée au niveau du point. Chacun des éléments de la liste utilise une structure de données relativement grande. Ainsi, pour chaque pixel, une grande quantité de mémoire peut être nécessaire. L'utilisation d'une liste impose

d'ailleurs l'usage d'une allocation dynamique de la mémoire. Cela est un sérieux inconvénient de cette méthode que, par ailleurs, [DUFF 85] cite comme difficile à implanter.

Cet algorithme de [CARP 84] est malgré tout une simplification (au niveau de la taille des données conservées et des calculs à effectuer pour chaque pixel) par rapport à l'*ultime* méthode de détermination de visibilité de [CATM 78] qui pour chaque pixel détermine les portions de polygones visibles grâce à l'algorithme d'Atherton et Weiler.

Catmull avait présenté dans [CATM 74] une première méthode d'antialiassage avec z-buffer. Contrairement à [CARP 84] et [CATM 78] la quantité d'information à conserver pour chaque pixel est bornée. On peut donc faire une allocation statique des tableaux nécessaires à ce stockage dès que la taille de l'image est fixée. Pour chaque pixel, il conserve :

- la couleur du polygone le plus proche, pondérée par l'aire de la surface qu'il occupe ;
- l'aire de la surface qu'il occupe,
- la couleur du polygone suivant en profondeur ;
- la direction de la normale par rapport au vecteur de visée (c'est-à-dire : est-ce que le polygone est vu de face ou de dos) ;
- le numéro de la primitive dont est issu le polygone.

Ces deux dernières informations permettent de détecter les silhouettes des primitives (il faut ici se souvenir que [CATM 74] est consacré à la visualisation de morceaux de surfaces), ce qui correspond à la détection de certains cas d'adjacence de polygones, ou en d'autres termes à la détection des corrélations entre images. Le cas de polygones intersectants n'est pas traité.

Enfin, nous pouvons mentionner la méthode de [FUJI 84]. Les auteurs présentent une machine graphique avec un z-buffer et traitement d'antialiassage. Ce dernier est assez rustique et les auteurs ne le détaillent pas beaucoup. Tous les cas problématiques ne reçoivent pas de solutions satisfaisantes : les changements de fond et les polygones intersectants ne sont pas traités. En particulier, l'affichage d'un fond après le premier plan est considéré par les auteurs comme un sujet de futures recherches.

4.5.1 L'algorithme du "gz-buffer"

4.5.1.1 Présentation

Cet algorithme que nous avons décrit dans [BEIG 86a] est lui aussi une amélioration de l'algorithme d'élimination des parties cachées utilisant une mémoire de profondeur pour chaque pixel. Le "gz-buffer" désigne une mémoire de stockage de la taille de l'image d'une façon analogue à la dénomination "z-buffer". Le "gz-buffer" se divise en deux parties, une qui est un "z-buffer" et une deuxième qui contient une information géométrique pour chaque pixel. Plus précisément, le

"g-buffer" contient pour chaque pixel le taux de recouvrement par le polygone le plus proche qui traverse le pixel. De la sorte, il contient le même type d'information que l'*alpha channel* de [PORT 84]. Nous avons choisi de mesurer ce taux de recouvrement en comptant le nombre de sous-pixels traversés par le polygone le plus proche. De la sorte, nous travaillons en quelque sorte par suréchantillonnage. Toutefois, toute méthode permettant de calculer ou d'approcher cette grandeur peut être combinée avec notre algorithme. Chaque pixel est virtuellement divisé en seize sous-pixels et cette information de recouvrement peut donc être stockée grâce à quatre bits par pixel. Lors de l'affichage d'un polygone noir sur fond blanc, le choix de subdivision en seize sous-pixels ne peut pas tirer partie de toute la dynamique qui est généralement disponible sur une mémoire d'images et qui est le plus souvent de 256 niveaux de gris, puisqu'il faudrait alors diviser les pixels en 256 sous-pixels. Ce choix de seize sous-pixels pour un pixel pourrait donc être remis en question, puisqu'il ne suffirait encore que d'un octet pour stocker cette information géométrique avec la précision suffisante pour utiliser toute la dynamique des mémoires d'images actuelles. Notre matériel (cf. annexe) ne permet, lui, de n'avoir que quatre bits pour chacune des couleurs primaires : rouge, vert et bleu. Dans ces conditions, notre choix correspond à la dynamique dont nous disposons sur notre mémoire d'images. De plus, nous disposons par ailleurs de quatre plans d'*overlay* dont l'utilisation usuelle est l'affichage de menu dans des programmes interactifs, mais sans utilité lors de la génération d'images ; aussi, nous pouvions les utiliser pour le stockage de ce taux de recouvrement tant que ce dernier n'avait pas besoin de plus de quatre bits d'information.

Nous noterons col[.,.], g[.,.], et z[.,.] les tableaux qui contiennent respectivement la couleur, le taux de recouvrement, et la profondeur de chaque pixel de l'écran.

4.5.1.2 L'algorithme

L'algorithme du "gz-buffer" se découpe en deux étapes : dans la première, il faut déterminer le taux d'occupation du polygone P en cours d'affichage sur chacun des pixels ; dans la deuxième étape il faut effectuer la mise à jour des tableaux col[.,.], g[.,.], et z[.,.].

Ainsi dans la première étape, on remplit deux tableaux gzp[.,.] et gp[.,.] qui ont *a priori* la taille de l'écran. gp[.,.] contient le nombre de sous-pixels traversés par le polygone **P** en cours d'affichage sans tenir compte de la visibilité du polygone **P** sur chacun de ces sous-pixels par rapport aux polygones précédemment affichés, donc sans utiliser le tableau z[.,.].

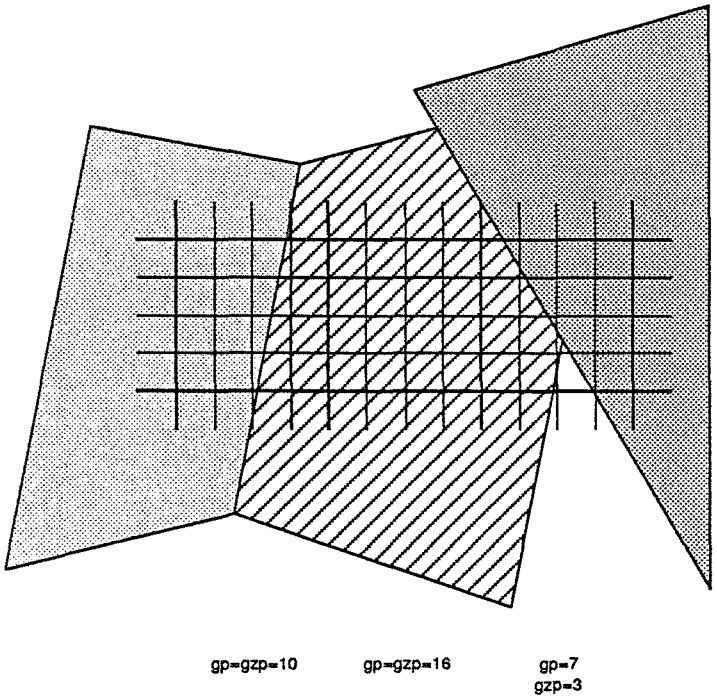


figure 4.24

gzp[.,.] garde le même type d'information mais tient compte de la visibilité du polygone **P** sur chacun des sous-pixels. En fait, comme nous n'avons qu'une valeur de profondeur pour chaque pixel, une première approximation est faite ici. Puisque cette valeur de profondeur est utilisée pour tous les sous-pixels, cela revient à considérer que le polygone précédemment affiché sur ce pixel y est parallèle à l'écran, alors que la profondeur du polygone en cours d'affichage est calculée pour tous les sous-pixels.

En pratique, ces tableaux ne sont pas remplis complètement avant que l'affichage effectif du polygone commence ; seules trois lignes consécutives du tableau sont disponibles puisque, comme on va le voir dans la description de la deuxième étape, trois lignes de ces tableaux sont suffisantes pour l'affichage d'une ligne d'écran du polygone **P**.

Etudions les possibilités de positionnement du polygone **P** par rapport aux polygones précédemment affichés :

1. **P** recouvre entièrement le pixel de coordonnées (x,y) ; les valeurs à affecter à col[x,y], g[x,y] et z[x,y] ne dépendent que de **P** et il n'y a donc pas de problème (cf. figure 4.25).

2. **P** ne recouvre pas entièrement le pixel de coordonnées (x,y) et ce pixel est plein (c'est-à-dire g[x,y] est égal à seize) ; nous devons mélanger la couleur présente sur le pixel col[x,y] avec la couleur du polygone **P** en les pondérant par le coefficient gzp[x,y]. Comme dans le cas précédent,

g[x,y] et z[x,y] sont calculés grâce au polygone **P** (cf. figure 4.26).

3. le polygone **P** ne recouvre pas entièrement le pixel de coordonnées (x,y) et ce pixel n'est pas plein (c'est-à-dire g[x,y] n'est pas égal à seize) ; nous devons affiner notre recherche pour trouver la couleur du polygone le plus proche en profondeur dont la couleur est mélangée avec une autre dans col[x,y] et dont la profondeur est conservée dans z[x,y].

   cas 3.1 : **P** est adjacent à un ancien polygone (cf. figure 4.27),

   cas 3.2 : **P** intersecte un ancien polygone (cf. figure 4.28),

   cas 3.3 : **P** est en partie caché par un ancien polygone et change la couleur de fond de celui-ci (cf. figure 4.29).

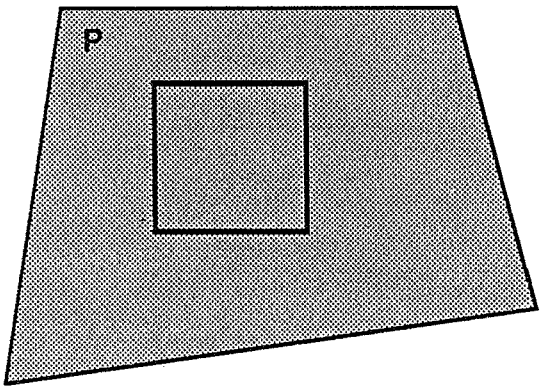


figure 4.25

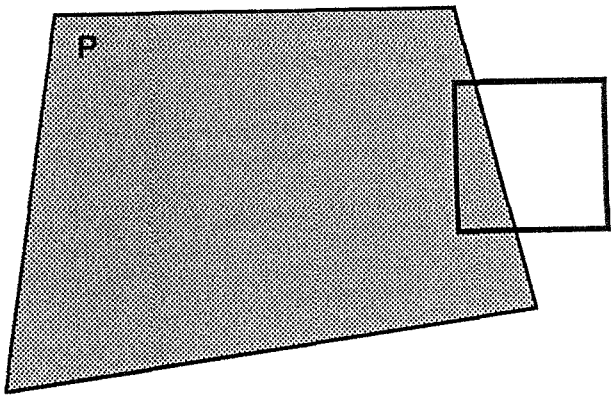


figure 4.26

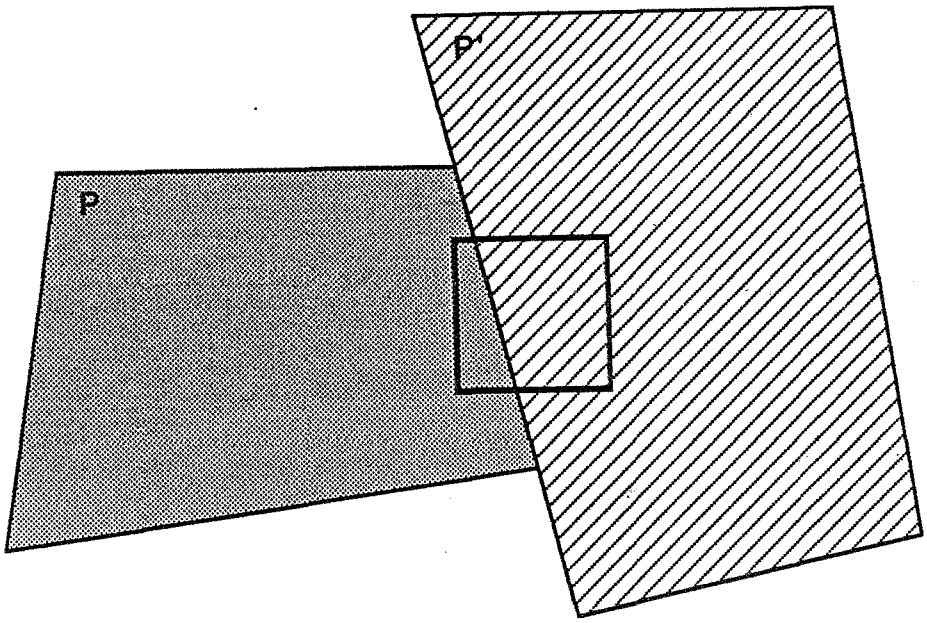


figure 4.27

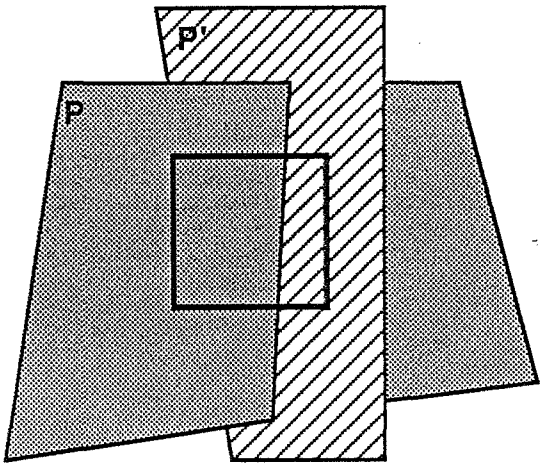


figure 4.28

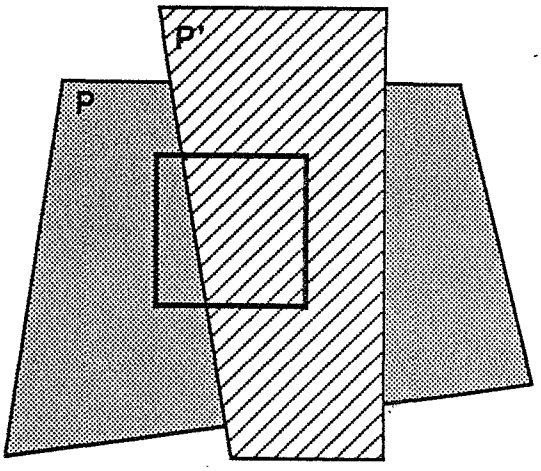


figure 4.29

Dans les deux premiers cas (3.1 et 3.2), nous devons mélanger la couleur du polygone P avec la couleur de l'ancien polygone au moyen du coefficient de pondération gzp[x,y]. Dans le troisième cas (3.3) le coefficient de pondération est g[x,y].

Nous devons donc maintenant voir comment nous pourrons discriminer ces différents cas les uns des autres et comment nous déterminerons la couleur de l'ancien polygone P' puisque cette couleur n'est pas disponible dans col[x,y] : col[x,y] contient une couleur qui correspond au mélange d'un arrière-plan avec la couleur de P'.

Nous avons déjà dit que les deux premiers cas (3.1 et 3.2) nécessitaient le même traitement. Ils n'arrivent que si gzp[x,y] n'est égal ni à zéro ni à seize. Le troisième cas (3.3) est plus difficile à identifier. Il survient lorsque :

- le pixel est entièrement recouvert par le polygone **P** sans tenir compte de la visibilité de **P** par rapport aux polygones déjà affichés (i.e. gp[x,y] est égal à seize),
- **P** n'est pas visible sur le pixel par rapport aux autres polygones,
- et **P** est visible sur au moins un des quatre pixels voisins du pixel de coordonnées (x,y).

Pour trouver la couleur du polygone **P'**, nous devons savoir si le bord entre **P** et **P'** est plutôt horizontal ou plutôt vertical. Dans les cas 1 et 2, l'information sur l'orientation du bord peut être trouvée grâce à la connaissance de **P** seulement et donc en n'utilisant que gzp[.,.], tandis que dans le troisième cas, cette information doit être touvée sur le bord de **P'**, donc en utilisant g[.,.].

Dans les cas 3.1 et 3.2, un pixel appartient à un bord plutôt vertical si et seulement si l'un des deux nombres gzp[x-1,y] et gzp[x+1,y] est égal à zéro et l'autre est égal à seize (cf. figure 4.30).

Dans le cas 3.3, un pixel appartient à un bord plutôt vertical si et seulement si g[x-1,y] et g[x+1,y] sont égaux à seize.

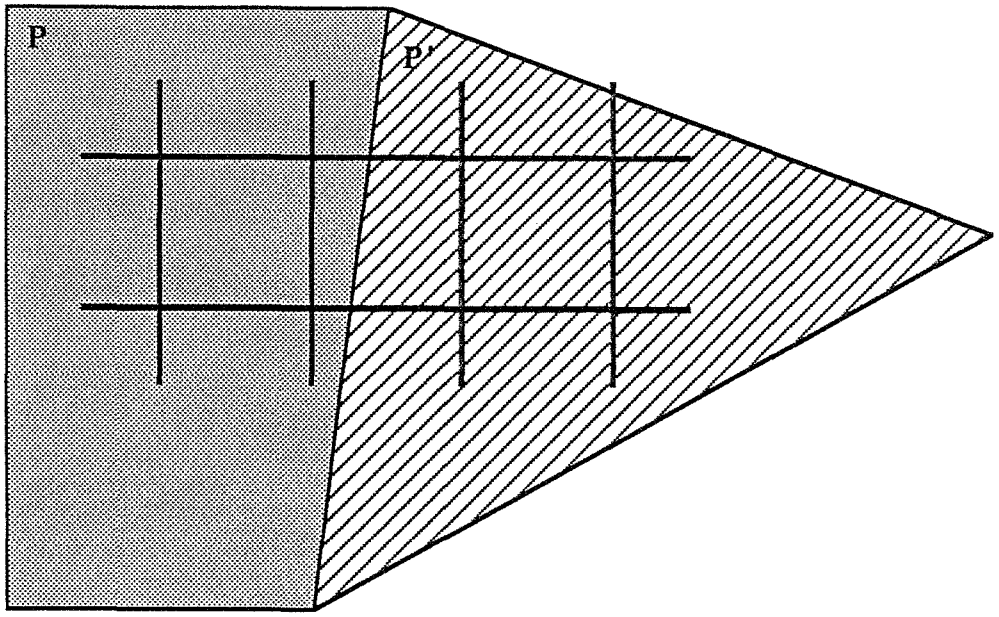


figure 4.30

Une fois que la direction (plutôt horizontale ou plutôt verticale) de la frontière entre les deux polygones a été trouvée, la couleur du polygone **P'** est prise comme étant celle d'un des pixels voisins du pixel de coordonnées (x,y). Dans le cas d'une arête plutôt verticale ce sera soit la couleur de celui de gauche, soit la couleur de celui de droite. La comparaison de gp[x-1,y] et de gp[x+1,y], et s'ils sont égaux celle de gzp[x-1,y] et de gzp[x+1,y] nous permet de choisir. Si gp[x-1,y] est strictement plus grand que gp[x+1,y], ou si gp[x-1,y] et gp[x+1,y] sont égaux et si gzp[x-1,y] est strictement plus grand que gzp[x+1,y], alors **P** est à la gauche de **P'** et la couleur de **P'** est évaluée comme égale à col[x+1,y]. Cette couleur est bien celle de **P'** à deux conditions :

- le polygone **P'** est suffisamment large,
- il n'y a pas d'autre polygone qui vient interférer sur ce pixel voisin.

D'une façon analogue, on choisit entre col[x,y-1] et col[x,y+1] pour le cas des arêtes plutôt horizontales.

On trouvera l'algorithme du "gz-buffer" ci-après. La variable *last* sert à conserver la couleur de l'ancien polygone. La fonction *blend* effectue le mélange des deux couleurs qui lui sont passées (premier et troisième paramètres) selon le coefficient défini par le deuxième paramètre.

```
début
  pour chaque pixel [x,y] sur l'écran faire
    début
    g[x,y] := 16;
    z[x,y] := ZMAX
    fin;

  pour chaque polygone faire
    calculer gp[.,y-1] et gzp[.,y] de la première ligne du polygone courant;
    pour chaque ligne y du polygone courant faire
      début
    calculer gp[.,y+1] et gzp[.,y+1];
      calculer i1 et i2, les abscisses d'intersection du polygone courant avec la
ligne courante;
      pour x:=i1 jusqu'à i2 faire
        si gzp[x,y] <> 0 (* le polygone courant est visible sur ce pixel *) alors
          début
          calculer zp, la profondeur du polygone courant sur ce pixel;
          si gzp[x,y] = 16 (* le polygone courant remplit ce pixel *) alors
            début
            pix[x,y] := col;
            z [x,y] := zp;
            g [x,y] := 16
            fin
          sinon (* le polygone courant ne remplit pas ce pixel *)
            si g[x,y] = 16 (* le pixel courant était plein *) alors
              début
              pix[x,y] := blend(col,gzp[x,y],pix[x,y]);
               (* on utilise comme coefficient de mélange le gzp du polygone
courant *)
              g [x,y] := gzp[x,y];
              z [x,y] := zp
              fin
            sinon (* le pixel courant n'était pas plein *)
              début
              si ((gzp[x-1,y] = 0) et (gzp[x+1,y] = 16)) ou ((gzp[x-1,y] = 16) et
(gzp[x+1,y] = 0)) alors
              (* l'arête est plutôt verticale sur ce pixel *)
                si ggp[x-1,y] < ggp[x+1,y] alors
                (* le polygone courant est sur la droite *)
                  last := pix[x-1,y]
                sinon
                  si ggp[x-1,y] > ggp[x+1,y] alors
                  (* le polygone courant est sur la gauche *)
                    last := pix[x+1,y]
                  sinon
                    si gzp[x-1,y] < gzp[x+1,y] alors
```

```
                (* le polygone courant est sur la droite *)
                  last := pix[x-1,y]
                sinon
                  si gzp[x-1,y] > gzp[x+1,y] alors
                  (* le polygone courant est sur la gauche *)
                    last := pix[x+1,y]
                  sinon
                    last := pix[x,y]
            sinon
            (* l'arête est plutôt horizontale sur ce pixel *)
              si ggp[x,y-1] > ggp[x,y+1] alors
              (* le polygone courant est sur le dessus *)
                last := pix[x,y+1]
              sinon
                si ggp[x,y-1] < ggp[x,y+1] alors
                (* le polygone courant est sur le dessous *)
                  last := pix[x,y-1]
                sinon
                  si gzp[x,y-1] > gzp[x,y+1] alors
                  (* le polygone courant est sur le dessus *)
                    last := pix[x,y+1]
                  sinon
                    si gzp[x,y-1] < gzp[x,y+1] alors
                    (* le polygone courant est sur le dessous *)
                      last := pix[x,y-1]
                    sinon
                      last := pix[x,y];
            pix[x,y] := blfin(col,gzp[x,y],last);
            g [x,y] := gzp[x,y];
            z [x,y] := zp
            fin
      fin
  sinon
    si (g[x,y] <> 16) (* le pixel courant n'est pas plein *)
    et (gzp[x-1,y]+gzp[x+1,y]+gzp[x,y]+gzp[x,y] <> 0) alors
  (* et le polygone courant est visible sur au moins un des pixels voisins *)
        début
        si (g[x-1,y] = 16) et (g[x+1,y] = 16) alors
         (* l'arête est plutôt verticale sur ce pixel *)
          si ggp[x-1,y] < ggp[x+1,y] alors
          (* le polygone courant est sur la droite *)
            last := pix[x-1,y]
          sinon
            si ggp[x-1,y] > ggp[x+1,y] alors
            (* le polygone courant est sur la gauche *)
              last := pix[x+1,y]
            sinon
              si gzp[x-1,y] < gzp[x+1,y] alors
              (* le polygone courant est sur la droite *)
                last := pix[x-1,y]
              sinon
```

```
                    si gzp[x-1,y] > gzp[x+1,y] alors
                    (* le polygone courant est sur la gauche *)
                      last := pix[x+1,y]
                    sinon
                      last := pix[x,y]
                sinon
                 (* l'arête est plutôt horizontale sur ce pixel *)
                  si ggp[x,y-1] > ggp[x,y+1] alors
                  (* le polygone courant est sur le dessus *)
                    last := pix[x,y+1]
                  sinon
                    si ggp[x,y-1] < ggp[x,y+1] alors
                    (* le polygone courant est sur le dessous *)
                      last := pix[x,y-1]
                    sinon
                      si gzp[x,y-1] > gzp[x,y+1] alors
                      (* le polygone courant est sur le dessus *)
                        last := pix[x,y+1]
                      sinon
                        si gzp[x,y-1] < gzp[x,y+1] alors
                        (* le polygone courant est sur le dessous *)
                          last := pix[x,y-1]
                        sinon
                          last := pix[x,y];
                pix[x,y] := blend(last,g[x,y],col)
                 (* on utilise comme coefficient de mélange le g de l'ancien poly-
gone *)
                fin
        fin fin
```

4.5.1.3 Le calcul de gzp[.,.] et de gp[.,.]

Ce calcul est analogue à un remplissage de polygone pour affichage sur une mémoire d'images. Les différences sont d'une part la résolution qui est celle d'une image suréchantillonnée et d'autre part la façon de traiter les bords des polygones par rapport aux pixels. En effet, si la première n'est qu'un changement de constantes par rapport aux algorithmes publiés, le second point, lui, n'est pas réellement traité, ni même cité. La question est de savoir à quelle condition un sous-pixel est considéré comme intérieur à un polygone. Ceci doit être fait d'une façon cohérente par rapport aux traitement d'antialiassage utilisé pour des polygones adjacents. En effet, considérons la figure 4.31 où les polygones P et P' sont voisins et partagent une arête.

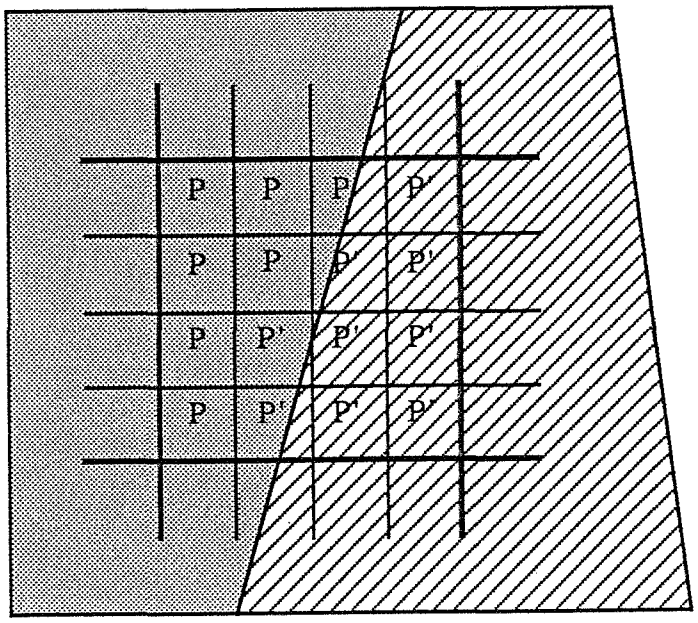


figure 4.31

Sans tenir compte de la profondeur, il faut que la somme du nombre de sous-pixels intérieurs à **P** et du nombre de sous-pixels intérieurs à **P'** soit égale au nombre de sous-pixels d'un pixel. De plus, se pose le problème de l'évaluation du nombre de sous-pixels recouverts par un polygone **P** pour un cas comme celui représenté sur la figure 4.32 :

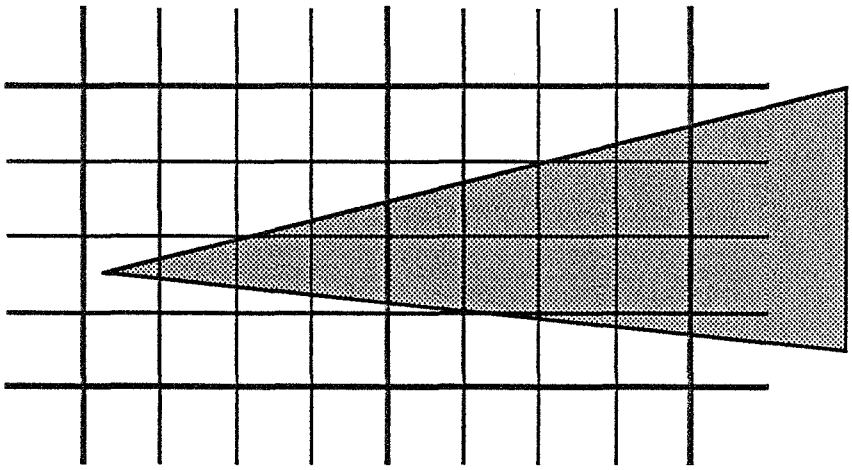


figure 4.32

Les algorithmes de remplissage de polygone considèrent les arêtes des polygones et pour chaque ligne de balayage mettent à jour la liste des arêtes actives. Pour résoudre le problème du changement d'arête active entre deux lignes de balayage (cf. figure 4.33), nous considérons nous des *paroi*s. Une paroi est une liste maximale d'arêtes qui ont la même orientation. Ainsi par rapport à la terminologie employée pour la description des algorithmes de remplissage, une *paroi* commence à un point de naissance et se termine à un point de mort. Les points de changement d'arête sur un bord sont tous internes à une *paroi*.

Nous avons choisi de considérer comme sous-pixels intérieurs tous les sous-pixels dont l'abscisse est comprise entre le minimum de :

- l'intersection de la ligne supérieure du pixel avec la *paroi* gauche,
- l'intersection de la ligne inférieure du pixel avec la *paroi* gauche,
- tous les sommets compris entre les lignes inférieure et supérieure du pixel et appartenant à la *paroi* gauche,

et le minimum *diminué d'une unité* de

- l'intersection avec la ligne supérieure du pixel avec la *paroi* droite,
- l'intersection avec la ligne inférieure du pixel avec la *paroi* droite,
- tous les sommets compris entre les lignes inférieure et supérieure du pixel et appartenant à la *paroi* droite.

Ainsi on a une parfaite symétrie entre les parois gauche et droite, ce qui assure que la somme des nombres de sous-pixels intérieurs pour deux polygones adjacents est bien le nombre de sous-pixels d'un pixel.

L'algorithme de remplissage que nous proposons commence donc par chercher les parois, ce qui est algorithmiquement linéaire par rapport au nombre de sommets du polygone. Chaque paroi est une liste d'arête. Simultanément, il construit une table des parois classées par un tri par seaux selon la partie entière de l'ordonnée de leur point de naissance. Cette partie est linéaire par rapport aux nombres de parois. Enfin, il se termine en évaluant les minima évoqués plus haut, tout en calculant les intersections avec les lignes inférieure et supérieure des pixels au moyen d'une méthode incrémentale.

L'évaluation de gzp est faite en même temps puisque la seule différence est qu'il faut tester, avant d'incrémenter le compteur de sous-pixels intérieurs à un pixel, si la profondeur courante sur le sous-pixel est plus petite que la profondeur conservée dans le tampon de profondeur.

La photo 4.1 montrait une ensemble de polygones affichés sans traitement d'antialiassage. La photo 4.2 est un zoom sur la photo 4.1. Les couples de photos 4.3 et 4.4, 4.5 et 4.6, 4.7 et 4.8, montrent l'affichage successif des polygones avec notre méthode d'antialiassage et un zoom sur la même zone que celle de la photo 4.2. On peut donc voir que l'ordre d'affichage des polygones n'était pas le plus favorable, puisqu'il y a changement de fond et passage d'un polygone entre deux autres.

4.5.1.4 Les défauts de la méthode du gz-buffer

Deux défauts affectent principalement l'algorithme du "gz-buffer" que nous venons de décrire.

Le plus gênant d'entre eux est le mauvais traitement qui est appliqué aux petits objets, du fait de la lecture de la couleur du polygone précédemment traité sur un pixel voisin de celui que l'on traite de nouveau ; en effet, dans ce cas, la couleur lue n'est pas la bonne si on sort du polygone (cf. figure 4.33).

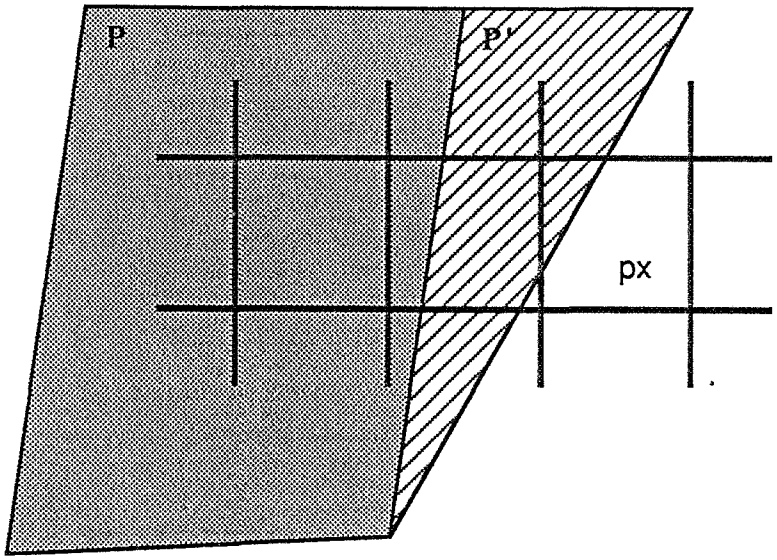


figure 4.33

D'autre part, les petits objets ont tendance à fabriquer des valeurs de g[.,.] et de gz[.,.] très "brouillées", c'est-à-dire avec lesquelles on ne pourra pas repérer la direction des arêtes parce que les pixels de deux bords proches et qui ne sont pas pleins se trouvent être voisins. Il y a en quelque sorte interférence entre les deux bords proches du polygone (cf. figure 4.34).

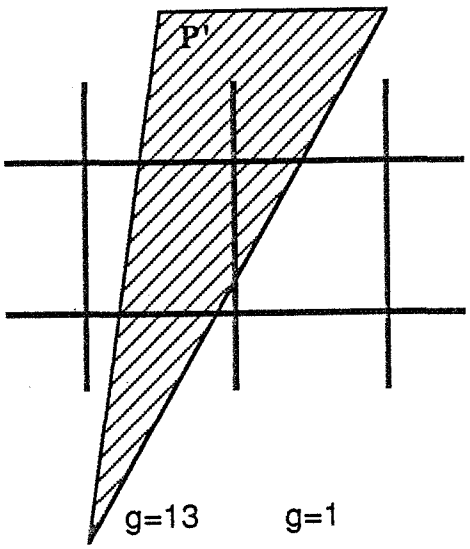


figure 4.34

Le même problème apparaît au voisinage des sommets, même pour un polygone qui autrement est "grand".

Le deuxième défaut est que cet algorithme ne peut faire des traitements corrects que sur les pixels traversés par au plus deux objets. Nous pouvons donner l'exemple où trois polygones P_1, P_2, et P_3 sont affichés dans cet ordre et le polygone P_3 est entre P_1 et P_2. Lorsque le pixel px de coordonnées (i,j) (cf. figure 4.35) est affiché, les tableaux gzp[.,.], gp[.,.] et g[.,.] contiennent les valeurs suivantes :

	i-1	i	i+1
gzp	0	0	16
gp	16	16	16
g	16	8	8

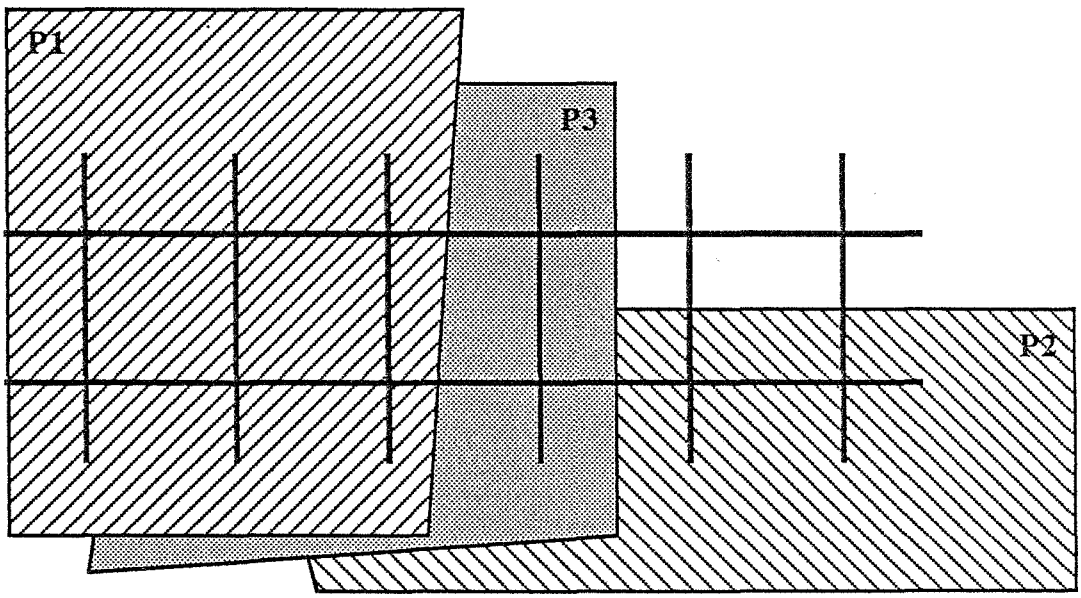


figure 4.35

Les valeurs de gzp et de gp sont caractéristiques d'un changement de fond et elles sont correctes. Par contre, les valeurs de g[i-1,j] et de g[i,j] viennent du polygone P_1, et celle de g[i+1,j] vient du polygonme P_2 ; il n'est pas possible de retrouver ces différentes origines avec les seules informations que nous conservons.

Ainsi le pixel px sera traité comme un cas de changement de fond avec une arête plutôt horizontale ; la valeur de la couleur de mélange sera lue soit au dessus de px soit en dessous de px, et cette valeur ne sera pas correcte.

4.5.1.5 Comparaison avec les autres méthodes

Nous allons dans ce paragraphe comparer l'algorithme du "gz-buffer" avec les autres algorithmes combinant l'élimination des parties cachées par un z-buffer et un traitement d'antialiassage, algorithmes que nous avons cités au début de cette section à savoir [CATM 74], [CARP 84], [FUJI 84] et [DUFF 85]. Les comparaisons sont présentées dans le tableau de la figure 4.36 :

	CATM 74	CARP 84	FUJI 84	DUFF 85	gz-buffer
polygones adjacents	oui-non*	oui	oui	oui	oui
changement de fond	oui-non*	oui	non	oui	oui
plus de deux objets par pixel	non	oui	non	non	non
petits objets	non	oui	non	non	non
véritable z-buffer	oui-non*	?	oui	oui	oui
mémoire bornée	oui	non	oui	oui	oui

figure 4.36

Le cas de [CATM 74] nécessite ici quelques explications pour les cas de polygones adjacents ou de changement de fond. Si l'élimination des parties cachées est faite avec la méthode du z-buffer, l'algorithme ne traite pas toujours correctement ces cas parce qu'il ne s'en aperçoit éventuellement pas. C'est d'ailleurs indiqué dans la publication : <<*L'algorithme par moyennage d'aires nécessite que les éléments soient traités selon leur ordre en z. [...] Si le z-buffer est utilisé et que l'ordre est mauvais, les erreurs n'apparaîtront que sur les silhouettes, où un effet de marches d'escalier pourra être visible.*>> (extrait de [CATM 74] pages 46 et 47).

On voit donc sur ce tableau que notre méthode se compare avantageusement aux autres méthodes. Par ailleurs, elle peut être améliorée au niveau des résultats si on utilise comme [CATM 74] une mémoire beaucoup plus importante et si l'on stocke les valeurs de couleurs *non mélangées*. De cette façon, on évite d'avoir à lire la couleur du polygone sur un pixel voisin avec tous les problèmes que cela pose (détermination de la direction de l'arête, problème des petits objets).

4.6 LE CAS DES TERRAINS

Les fichiers de données dont nous disposions représentent le terrain au moyen d'une grille altimétrique. Avec ce type de données, plusieurs algorithmes de visualisation sont utilisables. La méthode du lancer de rayons a été utilisée par [COQU 84b], et a nécessité une optimisation adaptée à ce type de données. Pour ce qui est des méthodes conventionnelles, nous en avons implantées deux, à savoir la liste de priorité et le z-buffer.

Le problème d'un fichier de données représentant un terrain par rapport aux fichiers de polygones usuellement utilisés en synthèse d'images est le grand nombre de facettes qui sont générés par un maillage. Aussi, il faut trouver un moyen permettant d'éliminer un grand nombre de celles (les plus nombreuses) qui seront en dehors du champ de vision. Pour cela, nous avons implanté la méthode décrite dans [COQU 84a]. Cette méthode se base sur le fait que l'altitude des points de la grille est bornée inférieurement par une altitude z_0 et supérieurement par une altitude z_1. Les plans horizontaux d'altitude z_0 et z_1 intersectent les plans limitant le cône de vision. [COQU 84a] montre que ces droites d'intersections permettent de définir deux polygones convexes G_t et G_p à bords parallèles tels que :

- G_t est inclus dans G_p ;
- toute maille incluse dans G_t est dans le cône de vision et ne nécessite pas de fenêtrage ;
- toute maille non incluse dans G_p n'intersecte pas le cône de vision et n'a donc pas à être considérée pour l'affichage ;
- les autres mailles nécessitent un fenêtrage.

En fait, cette sélection est un fenêtrage global tirant partie de la cohérence des données.

4.7 IMPLANTATION ACTUELLE : UNIX

4.7.1 Présentation générale

La figure 4.37 montre le schéma d'organisation des différents programmes qui ont été développés autour du langage de modélisation tridimensionnel CASTOR que nous avons décrit dans le deuxième chapitre de ce rapport.

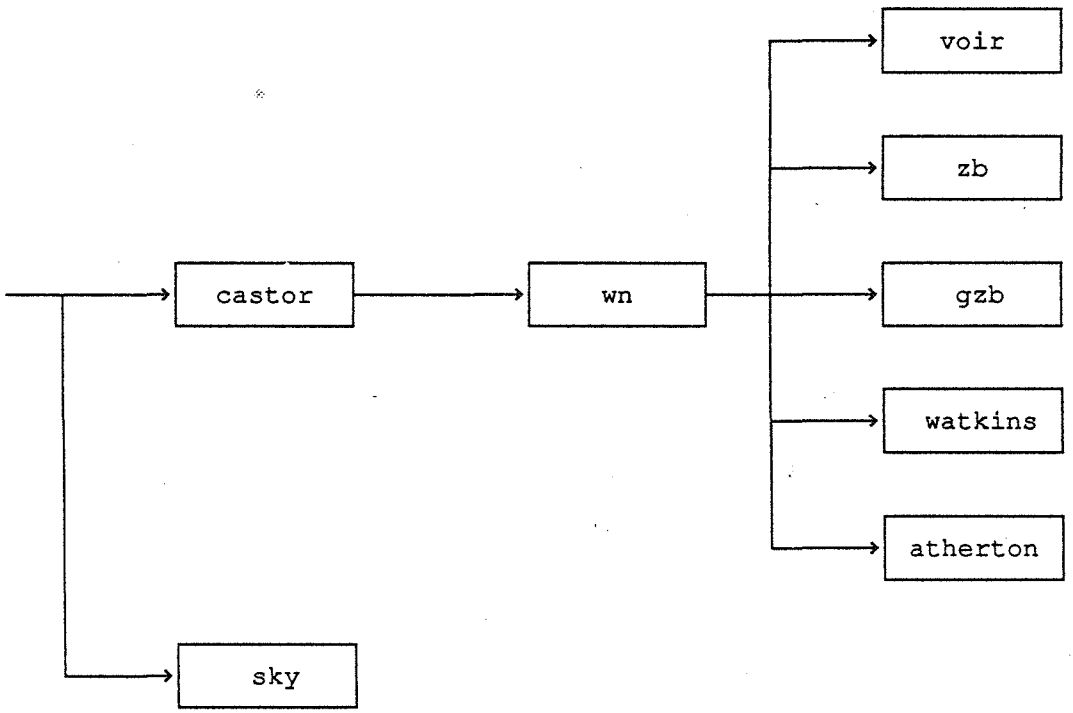


figure 4.37

Chacune des boîtes symbolise un programme :

castor est l'interprète du langage de modélisation CASTOR, son entrée est un fichier écrit selon la syntaxe de ce langage et sa sortie est

un fichier de polygones ;

wn est le programme effectuant la transformation perspective et le fenêtrage des polygones ;

voir implante l'algorithme d'élimination des parties cachées de Michelucci [MICH 87a] ; il a été écrit par Dominique Michelucci ;

zb envoie les polygones vers la mémoire d'images Lexidata Solidview dans laquelle l'élimination des parties cachées est faite grâce à la méthode du z-buffer ;

gzb implante l'algorithme d'élimination des parties cachées avec antialiassage du "gz-buffer" que nous avons décrit en 4.5.1.1 ;

watkins implante l'algorithme d'élimination des parties cachées de Watkins ; il a été écrit par Jean-Michel Moreau ;

atherton implante l'algorithme d'élimination des parties cachées pour la modélisation CSG d'Atherton [ATHE 83] ; il a été écrit par Dominique Michelucci et Didier Tallot ;

sky est un programme de synthèse complet utilisant la méthode du tracé de rayon ; il a été écrit par Jacqueline Argence.

Chacune des flèches indique une possibilité de communication de données. Cette communication s'effectue typiquement par l'intermédiaire de tubes de UNIX. Nous avons déjà indiqué dans le deuxième chapitre de ce rapport les influences que ce système d'exploitation avait eues sur notre implantation du langage de modélisation **CASTOR**. Cette même influence se poursuit ici.

castor et *sky* utilisent tous les deux en entrée un fichier de description CASTOR. Tous les autres échanges symbolisés sur cette figure concernent des polygones, aussi nous allons décrire le format employé pour ceux-ci dans le paragraphe suivant.

4.7.2 Le format de communication de polygones

Les programmes *voir* et *atherton* ont besoin de toute l'information booléenne contenue dans un arbre de description CSG pour pouvoir faire simultanément l'élimination des parties cachées et la conversion en représentation par frontières des solides. Aussi, nous transmettons cette information au moyen d'une structure parenthésée identique à celle du langage CASTOR. Pour décrire complètement la syntaxe de ces fichiers, il nous reste à donner la syntaxe de description d'un polygone :

<polygone> ::= P ([<liste d'attributs>] ({ <contour> }))

<liste d'attributs> ::= <normale> | <couleur> | <type d'ombrage>
<normale> ::= N <composantes>
<couleur> ::= C <composantes>
<type d'ombrage> ::= P | S

<contour> ::= (<sommet> { , <sommet> })

<sommet> ::= <coordonnées> [<liste d'attributs de sommet>]

<liste d'attributs de sommet> ::= <normale> | <couleur>

<composantes> ::= <nombre réel> <nombre réel> <nombre réel>
<coordonnées> ::= <nombre réel> <nombre réel> <nombre réel>

Un polygone est donc décrit par un ensemble d'attributs et une liste de contours, ce qui permet de décrire sans problème les polygones troués ou non connexes. La liste d'attributs permet de donner la normale au polygone telle qu'elle peut être calculée grâce à un produit vectoriel au moyen des coordonnées des sommets d'un contour. Remarquons que si cette information est optionnelle en entrée de *wn*, celui-ci la met toujours en sortie. Pour que ce calcul de normale soit possible, il faut être sûr que les sommets d'un contour sont donnés dans un certain ordre par rapport à l'intérieur du solide. Dans notre cas, nous les donnons dans l'ordre direct si l'on regarde le polygone de face depuis l'extérieur du solide. La couleur est donnée dans l'espace Rouge-Vert-Bleu. Le type d'ombrage permet de choisir entre une couleur unique pour le remplissage de la projection du polygone (option P) et un lissage de Gouraud (option S). Toutefois, pour que cette deuxième option puisse effectivement être prise en compte, il faut qu'une couleur ou une normale soit disponible pour chaque sommet. La normale en chaque sommet est définie comme la normale à la surface originelle en ce point. Cette possibilité permet de faire des lissages du type de celui de Gouraud ou de Phong sans avoir à trouver une normale approximative en chaque sommet en moyennant les normales des polygones partageant ce sommet.

4.8 COMPOSITION : L'EXTENSION DE L'IMPLANTATION, DEVELOPPEMENTS FUTURS

4.8.1 La composition au niveau du pixel

Les programmes actuels permettent une composition au niveau de la primitive polygone. Toutefois, la composition au niveau pixel me paraît intéressante à plusieurs points de vue :

- toutes les primitives ne se convertissent pas aisément en polygones (ou alors il en faut un très grand nombre, ce qui diminue les performances des algorithmes d'élimination de parties cachées tout en compliquant les traitements d'antialiassage) ;

- cela permet d'utiliser plusieurs algorithmes d'élimination des parties cachées, chacun d'entre eux pouvant être adapté à certaines primitives et par là être plus efficace ;

- de ce fait chacun des programmes gérant une primitive reçoit beaucoup moins de primitives ;

- cela permet d'appliquer les principes d'une programmation modulaire à un niveau supérieur en faisant des modules adaptés aux primitives plutôt qu'un énorme programme, difficile à maintenir et à étendre, qui sache gérer toutes les primitives présentes et à venir ;

- c'est enfin préparer une parallélisation du processus complet de synthèse en permettant de distribuer les différents processus spécialisés pour chaque type de primitives ainsi que le processus de composition sur plusieurs processeurs.

Comme on l'a vu, le principal problème dans une approche de composition au niveau du pixel est celui de l'aliassage. Cependant, celui-ci peut en grande partie être éliminé grâce à une méthode comme celle de [DUFF 85] ou celle du gz-buffer, puisque les cas où ils échouent (petits objets et corrélation des objets intersectant un même pixel) doivent normalement être largement éliminés. En effet, si chacune des images à composer est engendrée par un processus adapté aux primitives qui y interviennent, on ne doit pas avoir à découper ces primitives en petits objets. D'autre part, une découpe "naturelle" en objets ne doit pas conduire à séparer les objets "voisins" dont les projections seront corrélées.

4.8.2 Le rendu

Un des problèmes dans notre système de synthèse est la faible pénétration des attributs d'aspect et d'éclairement, ce qui conduit à re-générer complètement une image simplement parce que la couleur d'un des objets n'est pas tout à fait celle qui était voulue. L'approche de la composition au niveau du pixel peut être un début de solution à un tel problème si l'on peut se contenter de générer simplement une image ne contenant que l'objet avec sa nouvelle couleur et de composer cette nouvelle image avec la précédente. Cela ne résoud toutefois aucun problème pour la pénétration des attributs d'éclairement. Aussi les idées développées dans [COOK 84], puis dans [PERL 85] me paraissent très séduisantes.

L'idée principale de [COOK 84] est de permettre la description facile de la formule qui va servir à calculer le rendu de chaque point d'une primitive (c'est-à-dire en la spécifiant hors du contexte des algorithmes de conversion des primitives en points de l'image et d'élimination des parties cachées qui l'encadrent normalement). Cela évite de devoir faire entrer toutes les instanciations de primitives dans le même schéma d'éclairement et cette idée rejoint donc celle de modularité nécessaire à un système universel de synthèse d'images.

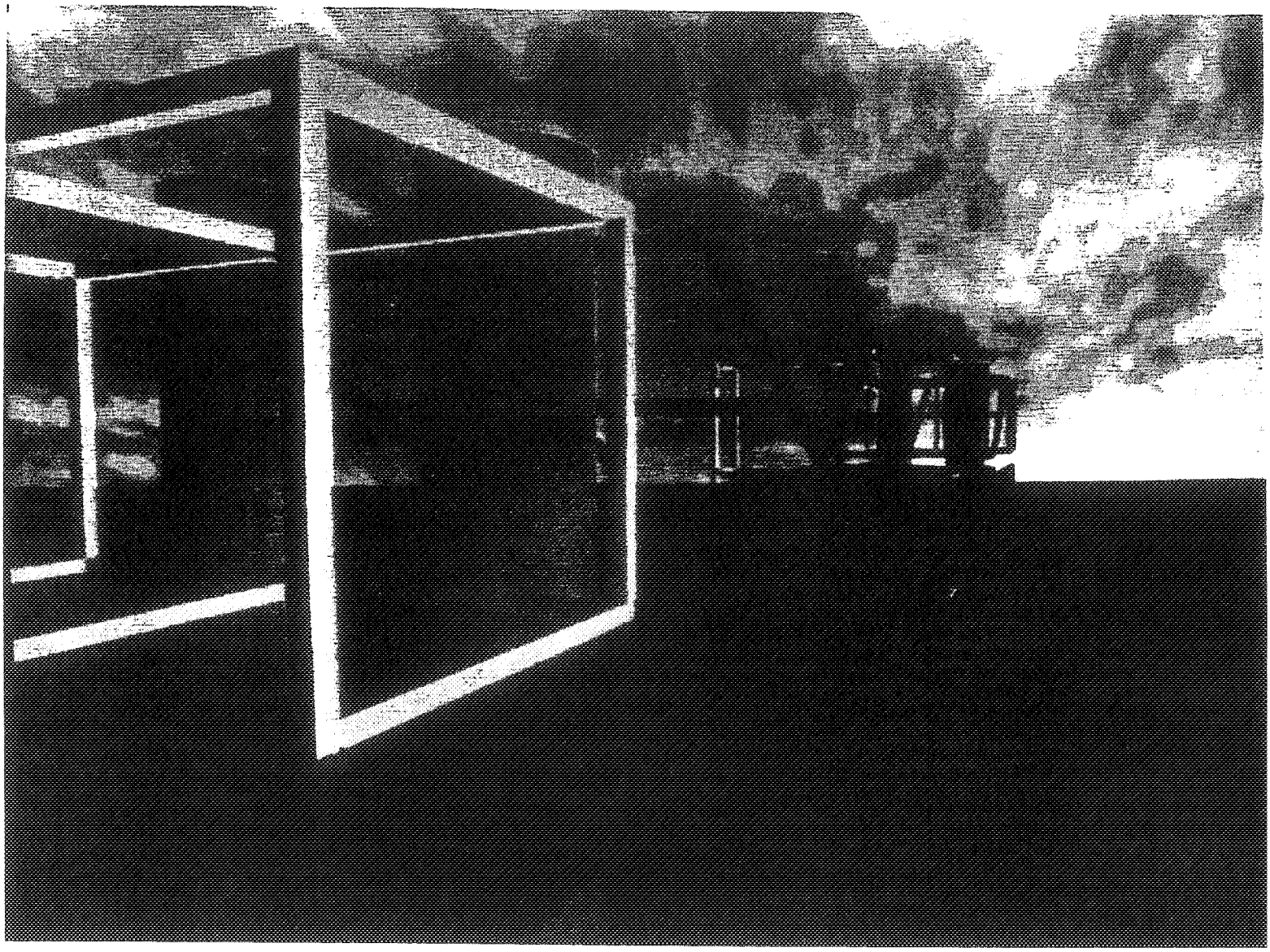
[PERL 85] ramène ces formules au niveau de l'image (ou, ce qui revient au même, au niveau des pixels). Il redéfinit d'ailleurs ces deux termes ; ainsi, pour toute liste de variables (par exemple [red green blue]), un *pixel* est une liste de valeurs correspondant à ces variables (par exemple [0.5 0.3 0.7]). Une *image* est une liste de noms de variables associée à un vecteur bidimensionnel de *pixel*s. Cette formulation permet de conserver pour chaque pixel autant d'informations qu'il peut être nécessaire, en particulier des composantes de normales, des identificateurs de primitives,... qui permettent de repousser aussi loin que voulu les opérations créant l'image définitive, ceci grâce à un langage manipulant ces images. Les opérations

décrites dans [PORT 84] et [DUFF 85] peuvent être vues comme des cas particuliers d'opérations disponibles dans le langage de [PERL 85].

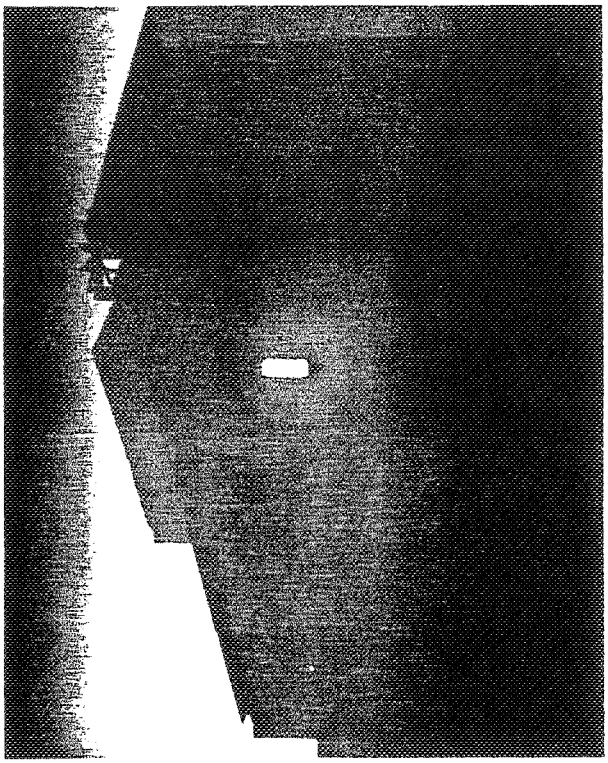
Aussi je verrai bien comme extension de notre système les étapes suivantes :

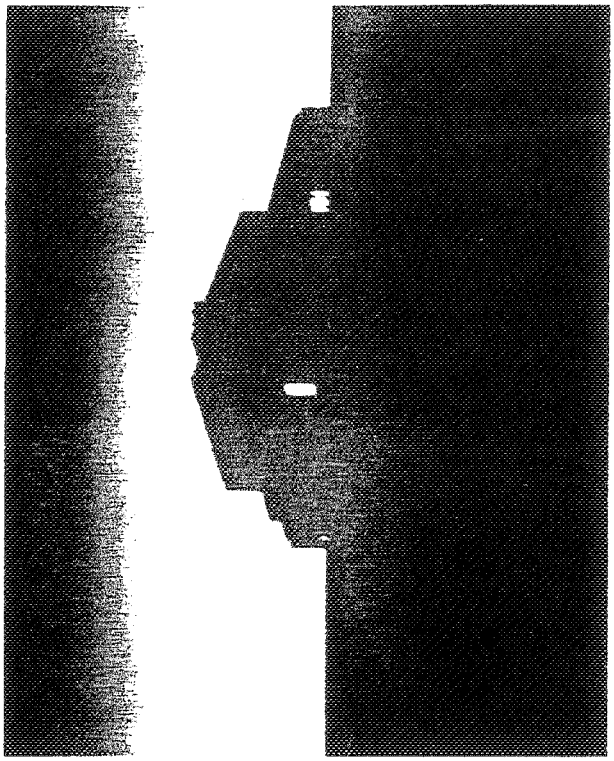
- la définition d'un format d'image permettant de garder toutes sortes d'informations,
- la modification des algorithmes cités en 4.7.1 pour qu'ils génèrent des images à ce format avec la possibilité de demander la sortie d'attributs qui, actuellement, sont des variables internes à ces algorithmes et utilisés dans l'étape de calcul d'éclairement de ces algorithmes (comme la normale par exemple) ;
- la définition d'opérations sur ces images combinant les techniques de rendu de [COOK 84] et d'antialiassage de [DUFF 85].

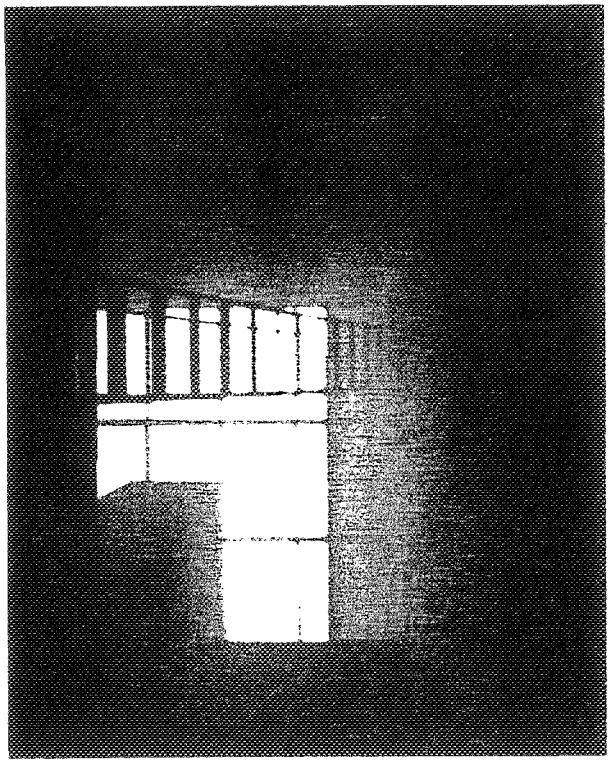
PHOTOGRAPHIES

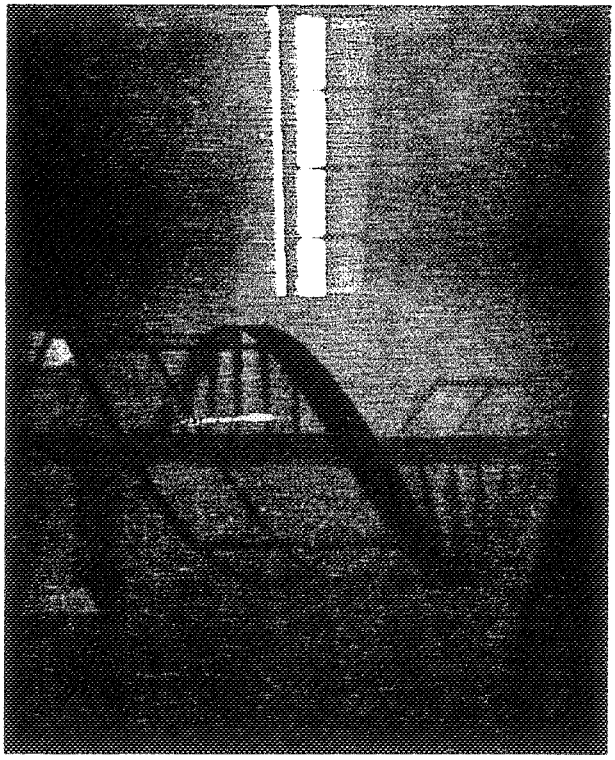


photographie 2.1

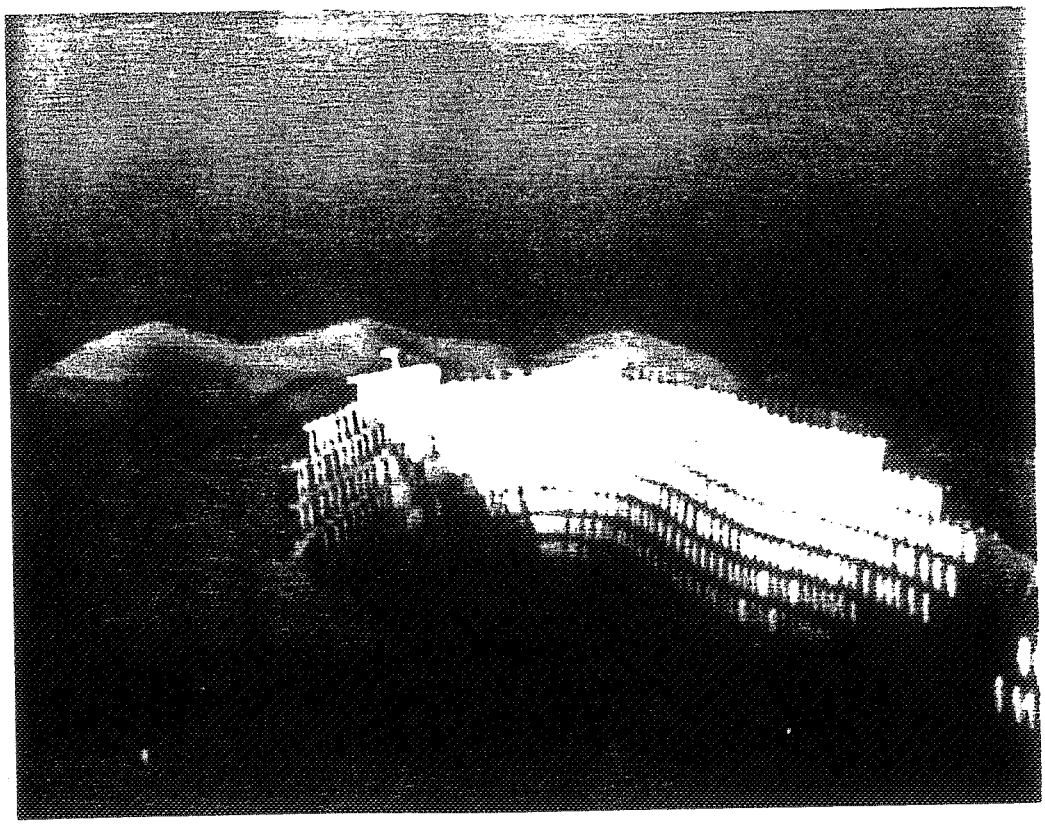


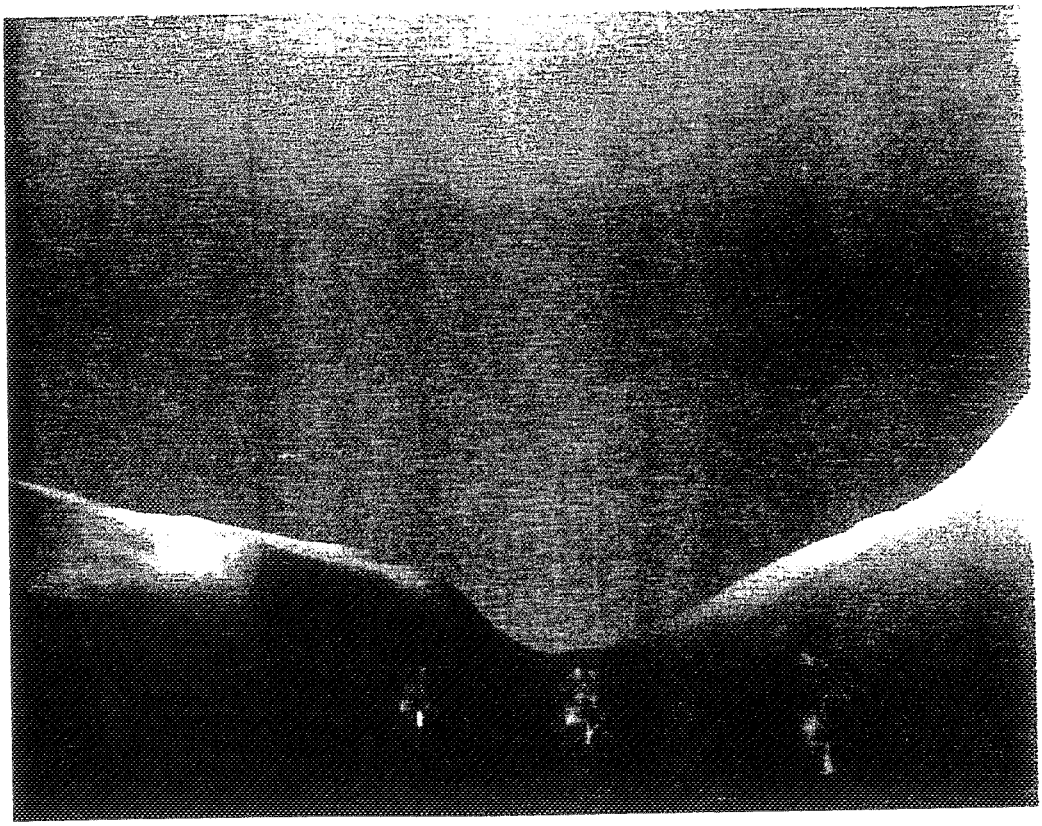




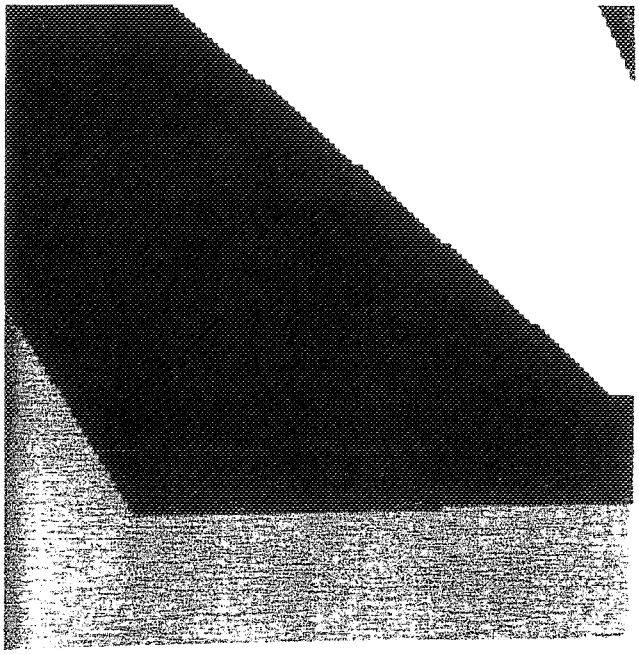


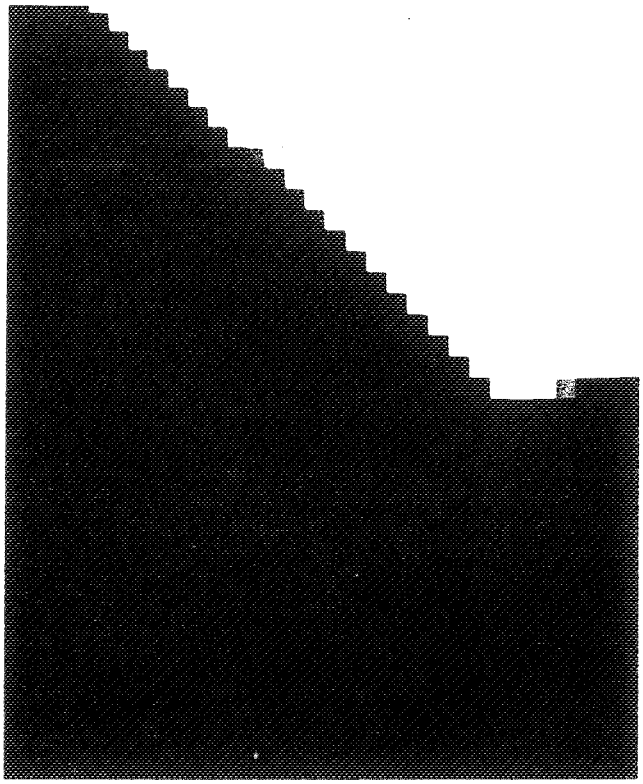
photographies 2.3, 2.4, 2.5 et 2.6

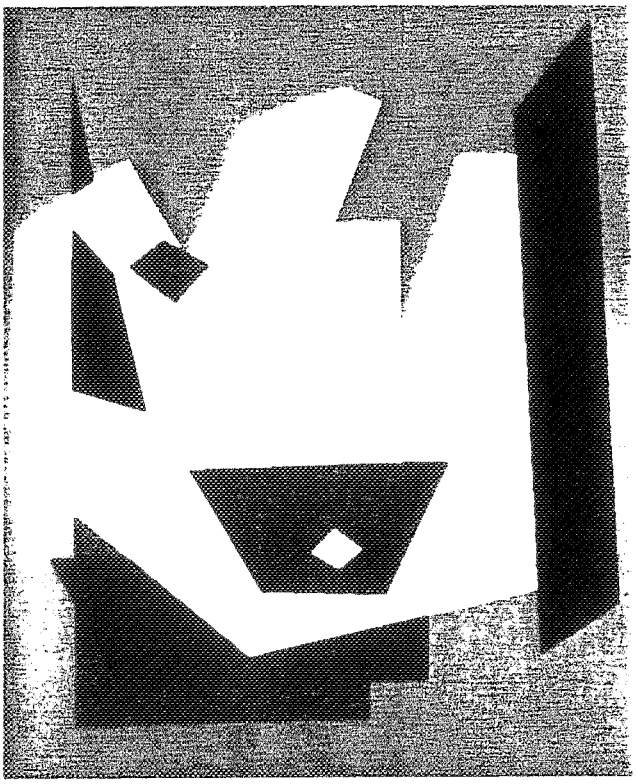


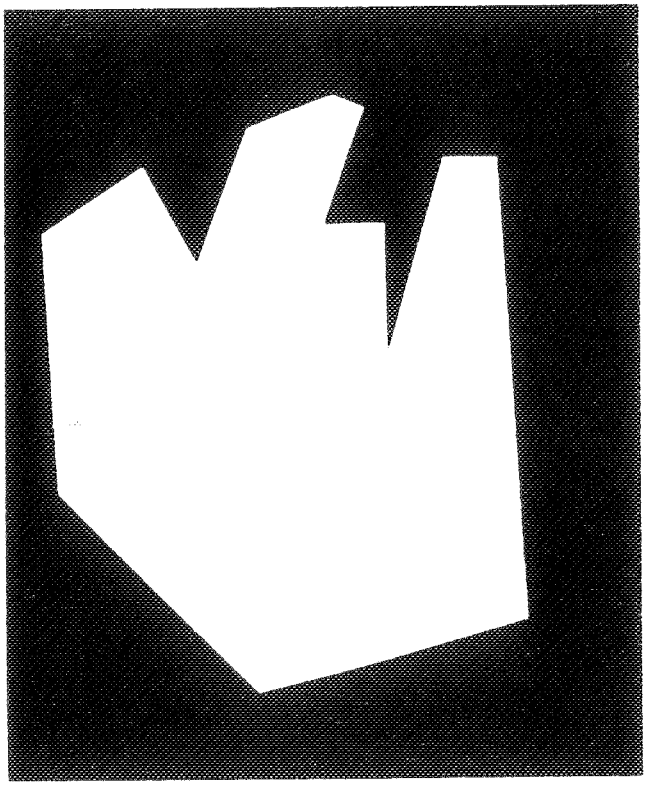


photographies 3.1 et 3.2

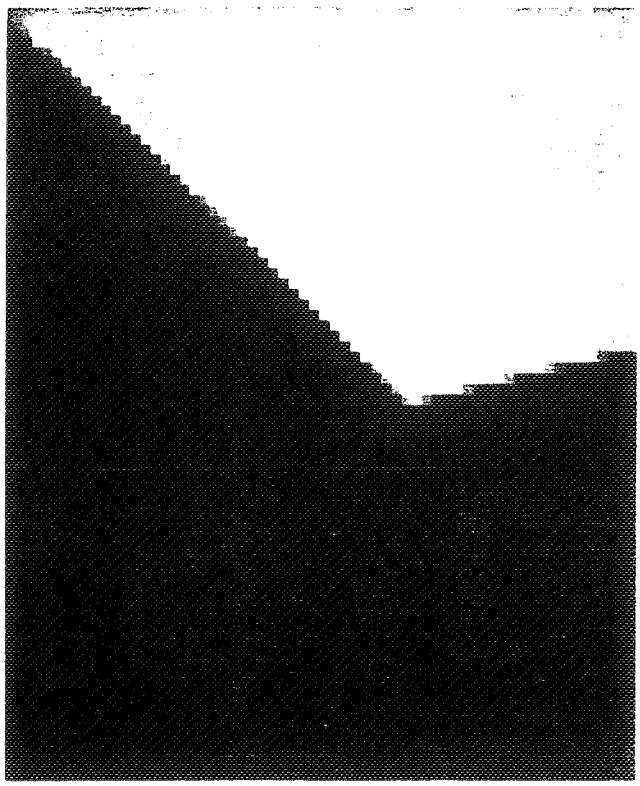


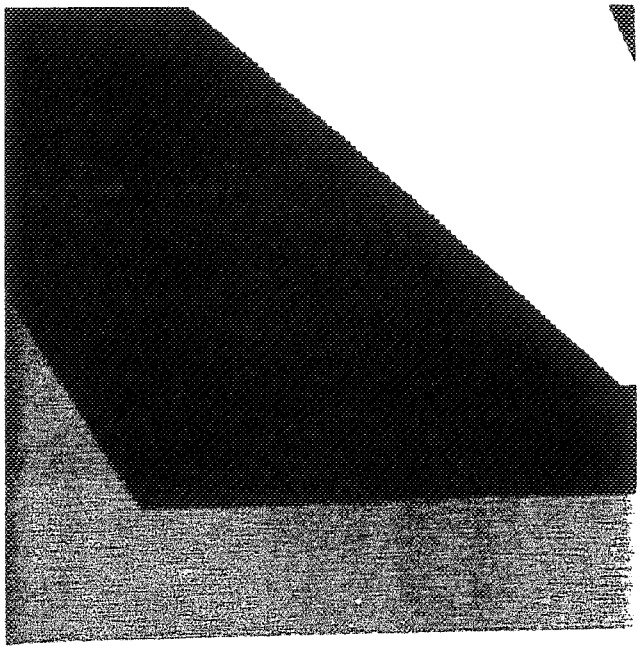


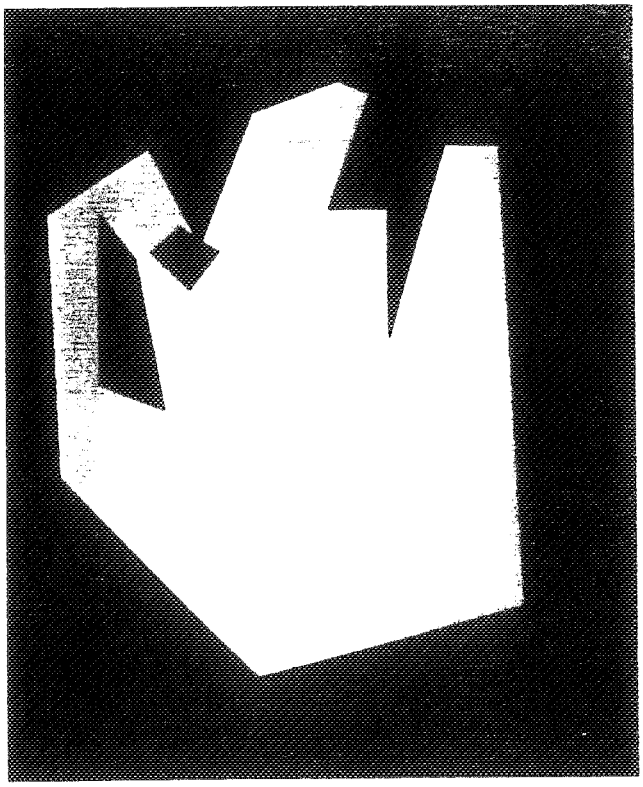


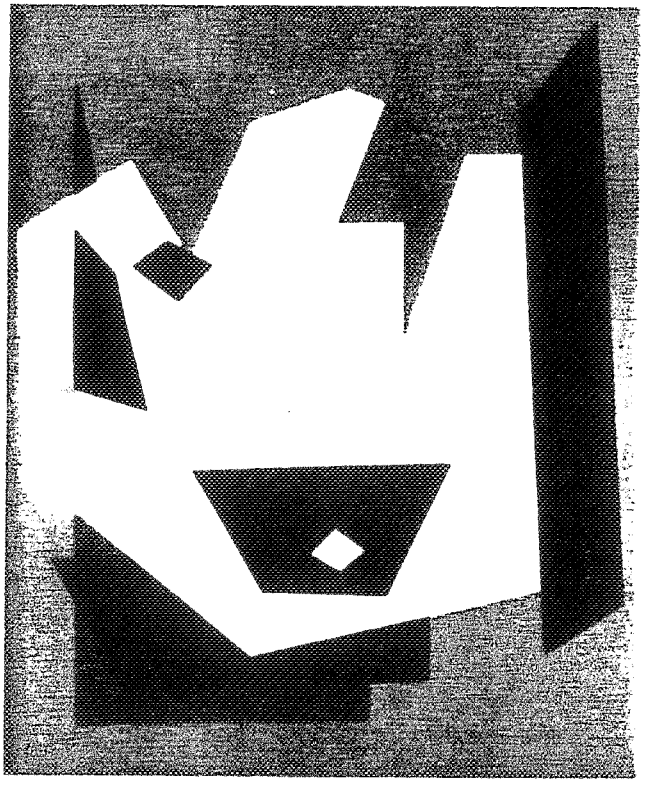


photographies 4.1, 4.2, 4.3 et 4.4









photographies 4.5, 4.6, 4.7 et 4.8

REFERENCES BIBLIOGRAPHIQUES

[ANDE 82] **anderson d.p.,** "hidden line elimination in projected grid surfaces", *ACM Trans. on Graph., Vol 1, Number 4, oct 1982.*

[AONO 84] **aono m., kunii t.l.,** "botanical tree image generation", *IEEE CG&A, may 1984,* pp. 10-34.

[ARGE 85] **argence j.,** "synthèse d'éléments naturels: arbres", *rapport de DEA informatique, Ecole des Mines de Saint-Etienne, 1985.*

[ATHE 83] **atherton p.a.,** "a scan-line hidden surface removal procedure for constructive solid geometry", **SIGGRAPH'83,** *Computer Graphics, Vol 17, Number 3,* pp. 73-82.

[BAUM 75] **baumgart b.g.,** "a polyhedron representation for computer vision", **AFIPS'75** conference proceedings, *1975, Vol 44,* pp. 589-596.

[BEIG 86a] **beigbeder m., ghazanfarpour d., peroche b.,** "un algorithme d'anti-aliasing avec "z-buffer"", *in* deuxieme colloque image, semaine internationale de l'image électronique **CESTA,** *avril 1986,* Nice, CESTA, 1986, pp. 817-822.

[BEIG 86b] **beigbeder m.,** "castor: un modeleur 3d", *in* CAO et robotique en architecture et B.T.P., actes des journées internationales, Marseille, Juin 1986 *Paris, Hermès, 1986,* pp. 349-358.

[BENT 79] **bentley j.l., ottmann t.a.,** "algorithms for reporting and counting geometric intersections", *IEEE Trans. on Comput., Vol C-28, Number 9, 1979.*

[BISH 86] **bishop g., weimer d.m.,** "fast phong shading", **SIGGRAPH'86,** *Computer Graphics, Vol 20, Number 4,* pp. 103-106.

[BLOO 85] **bloomenthal j.,** "modeling the mighty maple", **SIGGRAPH'85,** *Computer Graphics, Vol 19, Number 3,* pp. 305-311.

[BOUR 83] **bourne s.r.,** "the UNIX system", *Reading, Addison-Wesley, 1983.*

[BROW 82] **brown c.m.,** "padl-2: a technical summary", *IEEE CG&A, mar 1982,* pp. 69-84.

[CARP 84] **carpenter l.,** "the a-buffer, an antialiased hidden surface method", **SIGGRAPH'84,** *Computer Graphics, Vol 18, Number 3,* pp. 103-108.

[CATM 74] **catmull e.,** "a subdivision algorithm for computer display of curved surfaces", *PhD University of Utah, dec 1974.*

[CATM 78] **catmull e.,** "a hidden-surface algorithm with anti-aliasing", **SIGGRAPH'78,** *Computer Graphics, Vol 12, Number 3, 1978,* pp. 6-11.

[COOK 84] **cook r.l.,** "shade trees", **SIGGRAPH'84,** *Computer Graphics, Vol 18, Number 3,* pp. 223-230.

[COOK 87] **cook r.l., carpenter l., catmull e.,** "the reyes image rendering architecture", **SIGGRAPH'87,** *Computer Graphics, Vol 21, Number 4,* pp. 95-102.

[COQU 84a] **coquillart s.,** "shaded display of digital maps", *IEEE CG&A jul 1984,* pp. .

[COQU 84b] **coquillart s.,** "représentation de paysages et tracé de rayon", *thèse 3ème cycle, Ecole des Mines de Saint-Etienne, Université de Grenoble, décembre 1984.*

[CROW 82] **crow f.c.,** "a more flexible image generation environment", **SIGGRAPH'82,** *Computer Graphics, Vol 16, Number 3,* pp. 9-18.

[DOCT 81] **doctor l.j., torborg j.g.,** "display techniques for octree-encoded objects", *IEEE CG&A, jul 1981,* pp. 29-38.

[DUCH 87] **duchêne v.,** "introduction des déformations FFD dans CASTOR", *rapport de TFE informatique, Ecole des Mines de Saint-Etienne, 1987.*

[DUFF 79] **duff t.,** "smoothly shaded renderings of polyhedral objects on raster displays", **SIGGRAPH'79,** *Computer Graphics, Vol 13, Number 2,* pp. 270-274.

[DUFF 85] **duff t.,** "compositing 3d rendered images", **SIGGRAPH'85,** *Computer Graphics, Vol 19, Number 3,* pp. 41-44.

[FISH 80] **fishman b., schachter b.,** "computer displays of height fields", **computer & graphics,** *1980,* Vol 5, pp. 53-60.

[FOLE 82] **foley j.d., van dam a.,** "fundamentals of interactive computer graphics", *Reading, Addison-Wesley, 1982.*

[FUJI 83] **fujimoto a., iwata k.,** "jag-free images on raster displays", *IEEE CG&A, dec 1983,* pp. 26-34.

[FUJI 84] **fujimoto a., perrott c.g., iwata k.,** "a 3d graphics display system with depth buffer and pipeline processor", *IEEE CG&A, jun 1984,* pp. 11-23.

[GARD 84] **gardner g.y.,** "simulation of natural scenes using textured quadric surfaces", **SIGGRAPH'84**, *Computer Graphics, Vol 18, Number 3,* pp. 11-19.

[GARD 85] **gardner g.,** "visual simulation of clouds", **SIGGRAPH'85**, *Computer Graphics, Vol 19, Number 3,* pp. 297-303.

[GHAZ 85] **ghazanfarpour-kholendjany d.,** "synthèse d'images et antialiassage", *thèse de docteur-ingénieur, Ecole des Mines de Saint-Etienne, dec 1985.*

[GOAT 80] **goates g.b., griss m.l., herron g.j.,** "picturebalm: a lisp-based graphics language system with flexible syntax and hierarchical data structure", **SIGGRAPH'80**, *Computer Graphics, Vol 14, Number 3,* pp. 93-99.

[GOLD 86] **goldfeather j., hultquist j.p.m., fuchs h.,** "fast constructive solid geometry display in the pixel-powers graphics system", **SIGGRAPH'86**, *Computer Graphics, Vol 20, Number 4,* pp. 107-116.

[GOUR 71] **gouraud h.,** "computer display of curved surfaces", *IEEE Trans., Vol C-20, juin 1971,* pp. 623-629.

[HOOK 86] **van hook t.,** "real-time shaded nc milling display", **SIGGRAPH'86**, *Computer Graphics, Vol 20, Number 4,* pp. 15-20.

[HUGL 80] **hugli h.,** "de la synthèse d'images appliquée aux maquettes de terrains numériques", *Institut de physique technique, Ecole Polytechnique Fedérale de Zurich, 1980.*

[JANS 85] **jansen f.w.,** "a csg list priority hidden surface algorithm", **Eurographics'85**, *Amsterdam, North-Holland, 1985,* pp. 51-62.

[KAJI 82] **kajiya j.t.,** "ray tracing parametric patches", **SIGGRAPH'82**, *Computer Graphics, Vol 16, Number 3,* pp. 245-254.

[KAUF 85] **kaufman a., bakalash r.,** "a 3d cellular frame buffer", **Eurographics'85**, *Amsterdam, North-Holland, 1985,* pp. 215-

220.

[KAWA 82] **kawaguchi y.,** "a morphological study of the form of nature", **SIGGRAPH'82,** *Computer Graphics, Vol 16, Number 3,* pp. 223-232.

[LAID 86] **laidlaw d.h., trumbore w.b., hughes j.f.,** "constructive solid geometry for polyhedral objects", **SIGGRAPH'86,** *Computer Graphics, Vol 20, Number 4,* pp. 161-170.

[LANE 79] **lane j.,** "a generalized scan line algorithm for the computer display of parametrically defined surfaces", *computer graphics and image processing, 1979,* Vol 11, pp. 290-297.

[LANE 80] **lane j.m., carpenter l.c., whitted t., blinn j.f.,** "scan line methods for displaying parametrically defined surfaces", *CACM, jan 1980, Vol 23, Number 1,* pp. 23-34.

[LEE 82] **lee y.t., requicha a.a.g.,** "algorithms for computing the volume and other integral properties of solids II", *CACM, sep 1982, Vol 25, Number 9,* pp. 642-650.

[LEVI 76] **levin j.,** "a parametric algorithm for drawing pictures of solid objetcs composed of quadric surfaces", *CACM, oct 1976, Vol 19, Number 10,* pp. 555-563.

[LEVI 80] **levin j.z.,** "quadril: a computer language for the description of quadric-surface bodies", **SIGGRAPH'80,** *Computer Graphics, Vol 14, Number 3,* pp. 86-92.

[LIEN 87] **lienhardt p., françon j.,** "synthèse d'images de feuilles végétales", *in* image électronique Paris 1987, Paris, CESTA, *1987,* pp. 213-218.

[MAGN 85] **magnenat-thalmann n., thalmann d.,** "computer animation, theory and practice", *Tokyo, Springer-Verlag, 1985.*

[MAHL 72] **mahl r.,** "visible surface algorithms for quadric patches", *IEEE transactions on computers, jan 1982, Vol c21, Number 1,* pp. 1-4.

[MANT 83] **mantyla m., tamminen m.,** "localized set operations for solid modeling", **SIGGRAPH'83,** *Computer Graphics, Vol 17, Number 3,* pp. 279-288.

[MARS 80] **marshall r., wilson r., carlson w.,** "procedure models for generating 3d terrain", **SIGGRAPH'80,** *Computer Graphics, Vol 14, Number 3,* pp. 154-162.

[MART 84] **martinez f.,** "la synthèse d'image: concepts, matériels et logiciels", *Paris, Editests, 1984.*

[MAX 81] **max n.l.,** "vectorized procedural models for natural terrain: waves and islands in the sunset", **SIGGRAPH'81,** *Computer Graphics, Vol 15, Number 3,* pp. 317-324.

[MEAG 82] **meagher d.,** "geometric modeling using octree encoding", *computer graphics and image processing, 1982, Vol 19,* pp. 129-147.

[MICH 84] **michelucci d., gangnet m.,** "saisie de plans à partir de tracés à main levée", **MICAD'84,** *3ème conférence européenne sur la CFAO et l'infographie, Paris, Hermès, 1984,* pp. 95-110.

[MICH 87a] **michelucci d., peroche b.,** "représentation interactive d'arbres CSG", **MICAD'87,** *6ème conférence européenne sur la CFAO et l'infographie, Paris, Hermès, 1987,* pp. 535-548.

[MICH 87b] **michelucci d.,** "les représentations par frontières: quelques constructions; difficultés rencontrées", *thèse de doctorat, Ecole des Mines de Saint-Etienne, Novembre 1987.*

[NADA 87] **nadas t., fournier a.,** "GRAPE: an environment to build display processes", **SIGGRAPH'87,** *Computer Graphics, Vol 21, Number 4,* pp. 75-84.

[ODON 81] **o'donnel t.j., olson a.j.,** "gramps - a graphics language interpreter for real-time interactive 3d picture editing and animation", **SIGGRAPH'81,** *Computer Graphics, Vol 15, Number 3,* pp. 133,142.

[OHAS 85] **ohashi t., uchiki t., tokoro m.,** "a three-dimensionnal shaded display method for voxel-based representation", **Eurographics'85,** *Amsterdam, North-Holland, 1985,* pp. 221-232.

[OKIN 84] **okino n., kakazu y., morimoto m.,** "extended depth-buffer algorithms for hidden-surface visualization", *IEEE CG&A, may 1984,* pp. 79-88.

[PEAC 85] **peachey d.r.,** "solid texturing of complex surfaces", **SIGGRAPH'85,** *Computer Graphics, Vol 19, Number 3,* pp. 279-286.

[PERL 85] **perlin k.,** "an image synthesizer", **SIGGRAPH'85,** *Computer Graphics, Vol 19, Number 3,* pp. 287-296.

[PETR 87] **petrie g., kennie t.j.m.,** "terrain modelling in surveying and civil engineering", *CAD, may 1987, Vol 19, Number 4,* pp. 171-187.

[PHON 75] **phong b.t.,** "illumination for computer generated pictures", *CACM, jun 1975, Vol 18, Number 6,* pp. 311-317.

[PITT 80] **pitteway m.l.v., watkinson d.j.,** "bresenham's algorithm with grey scale", *CACM, nov 1980, Vol 23, Number 11,* pp. 625-626.

[PORT 84] **porter t., duff t.,** "compositing digital images", SIGGRAPH'84, *Computer Graphics, Vol 18, Number 3,* pp. 253-259.

[POTM 87] **potmesil m., offert e.m.,** "FRAMES: software tools for modeling, rendering and animation of 3d scenes", SIGGRAPH'87, *Computer Graphics, Vol 21, Number 4,* pp. 85-93.

[REQU 80] **requicha a.a.g.,** "representations for rigid solids: theory, methods, and systems", *computing surveys, dec 1980, Vol 12, Number 4,* pp. 437-464.

[REQU 85] **requicha a.a.g., voelcker h.b.,** "boolean operations in solid modeling:boundary evaluation and merging algorithm", *proceedings of the IEEE, jan 1985, Vol 73, Number 1,* pp. 30-44.

[ROSS 86] **rossignac j.r., requicha a.a.g.,** "depth-buffering display techniques for constructive solid geometry", *IEEE CG&A, sep 1986,* pp. 29-39.

[ROTH 82] **roth s.d.,** "ray casting for modeling solid", *computer graphics and image processing, 1982, Vol 18,* pp. 109-144.

[SCHA 80] **schachter b.j.,** "long-crested wave models", *computer graphics and image processing, Vol 12, 1980,* pp. 187-201.

[SCHA 83] **schachter b.j.,** "computer image generation", *New-York, Wiley, 1983.*

[SCHW 82] **schweitzer d., cobb e.s.,** "scanline rendering of parametric surfaces", SIGGRAPH'82, *Computer Graphics, Vol 16, Number 3,* pp. 265-271.

[SEDE 86] **sederberg t.w., parry s.r.,** "free-form deformation of polygonal data", *in* deuxieme colloque image semaine internationale de l'image electronique, CESTA, *avril 1986,* Nice, pp. 634-639.

[SMIT 84] **smith a.r.,** "plants, fractals and formal languages", SIGGRAPH'84, *Computer Graphics, Vol 18, Number 3,* pp. 1-10.

[SUTH 74] **sutherland i.e., sproull r.f., schumacker r.a.,** "a characterization of ten hidden-surface algorithms", *computing surveys, mar 1974, Vol 6, Number 1,* pp. 1-55.

[TAMM 84] **tamminen m., samet h.,** "efficient octree conversion by connectivity labeling", **SIGGRAPH'84**, *Computer Graphics, Vol 18, Number 3,* pp. 43-51.

[TILO 80] **tilove r.b.,** "set membership classification : a unified approach to geometric intersection problems", *IEEE transactions on computers, oct 1980, Vol 29, Number 10,* pp. 874-883.

[TURK 86] **turkowski k.,** "anti-aliasing in topological color spaces", **SIGGRAPH'86**, *Computer Graphics, Vol 20, Number 4,* pp. 307-314.

[VOEL 78] **voelker h., requicha a., hartquist e., fisher w., metzger j., tilove r., birrel n., hunt w., armstrong,** "the padl-1.0/2 system for defining and displaying solid objects", **SIGGRAPH'78**, *Computer Graphics, Vol 12, Number 3,* pp. 257-263.

[VOSS 85] **vossler d.l.,** "sweep-to-csg conversion using pattern recognition techniques", *IEEE CG&A, aug 1985,* pp. 61-68.

[WEIL 77] **weiler k., atherton p.,** "hidden surface removal using polygon area sorting", **SIGGRAPH'77**, Computer Graphics, Vol 11, Number 2, pp. 214-222.

[WHIT 81] **whitted t., weimer d.m.,** "a software test-bed for the development of 3d graphics system ", **SIGGRAPH'81**, *Computer Graphics, Vol 15, Number 3,* pp. 271-277.

[WILL 85] **willis p.j.,** "a review of recent hidden surface removal techniques", *Displays, Vol 6, Number 1, 1985,* pp. 11-20.

[YAMA 84] **yamaguchi f., tokieda t.,** "a unified algorithm for boolean shape operations", *IEEE CG&A, jun 1984,* pp. 24-37.

ANNEXE

Les programmes décrits dans ce rapport ont été réalisés en langage C sur le matériel de l'équipe *Communications Visuelles* de l'Ecole des Mines de Saint-Etienne.

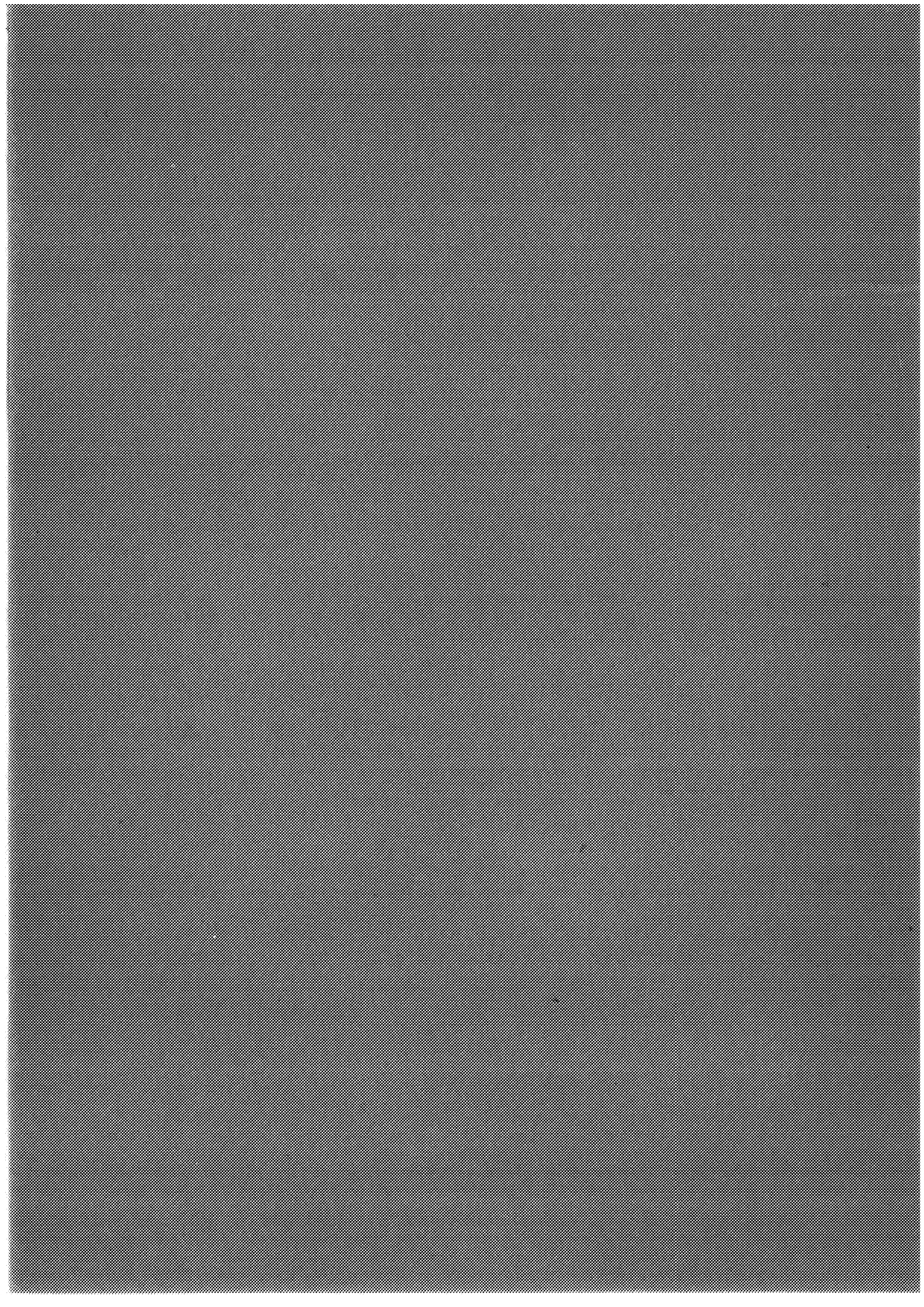
L'ordinateur hôte est un hp 9000 série 500 de Hewlett Packard, fonctionnant sous un système d'exploitation de la famille UNIX : HP-UX. Il est équipé avec :

- un processeur 32 bits,
- 2.5 méga-octets de mémoire centrale,
- un disque de 120 méga-octets.

La mémoire d'images est un SolidView de LEXIDATA dont les caractéristiques principales sont :

- une définition de 640 x 512 points,
- une table de couleurs de 12 bits en entrée et 3 x 8 bits en sortie,
- 12 plans mémoire pour l'image, ces plans adressent la table de couleur et permettent donc d'obtenir simultanément 4096 couleurs que l'on peut choisir dans une palette de 16 millions,
- 12 plans mémoire pour le tampon de profondeur,
- 4 plans pour la superposition ("*overlay*").

Les animations ont été enregistrées sur un magnétoscope SONY VO-5850P permettant l'enregistrement image par image. Toutefois, pour utiliser ce mode de fonctionnement, il ne faut pas le faire attendre plus de cinq minutes entre l'enregistrement de deux images (ceci à cause d'un problème d'usure de la bande magnétique). Aussi comme nous ne pouvions pas garder plus de trois ou quatre images sur le disque de l'ordinateur à cause de sa petite taille et de son encombrement, toutes les images des animations devaient pouvoir être calculé en moins de cinq minutes, ce qui nous imposait donc d'utiliser la méthode du tampon de profondeur implanté sur le LEXIDATA, ce qui explique qu'elles ne sont pas de très grande qualité.



Résumé : *La génération d'images de synthèse nécessite deux étapes : l'une pour fabriquer une maquette géométrique tridimensionnelle d'une scène (modélisation), l'autre pour prendre une "photographie" de cette maquette (visualisation). Pour chacune de ces deux étapes, de nombreuses méthodes ont été proposées. Cette thèse en présente un tour d'horizon et une classification des algorithmes d'élimination des parties cachées. Les problèmes de visualisation engendrés par le grand nombre de modélisations possibles sont abordés.*

Pour la modélisation, s'appuyant sur les conclusions du tour d'horizon, un langage permettant la description de scène est présenté ainsi que son intégration sous UNIX.

Enfin, une technique d'antialiassage originale avec la méthode d'élimination des parties cachées du tampon de profondeur est présentée.

Mots clés : *synthèse d'images, modélisation des solides, langage de représentation, élimination des parties cachées, antialiassage.*