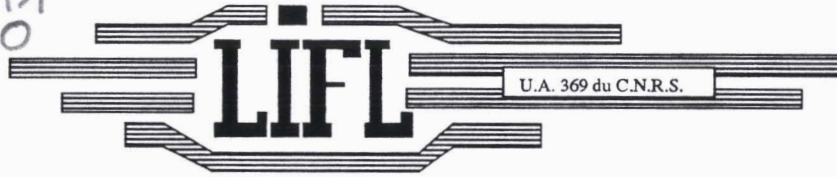


50376
1991
20

66357

50376
1991
20

USTL
FLANDRES ARTOIS



CU

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

Année 1991

Numéro d'ordre: 672

THESE

présentée à

l'Université des Sciences et Techniques de LILLE FLANDRES ARTOIS

Pour Obtenir le titre de

Docteur en Informatique



Titre:

**Etude d'un processeur de visualisation
d'images de synthèse en temps réel
exploitant un parallélisme massif objet:
le projet I.M.O.G.E.N.E**

Par Christophe CHAILLOU

soutenue le 16 Janvier 1991 devant la commission d'Examen

Membres du Jury:

Vincent CORDONNIER	Président
Michel LUCAS	Rapporteur
Jean Paul SANSONNET	Rapporteur
Ute CLAUSSEN	Examineur
Michel MERIAUX	Directeur de Thèse
Bernard PEROCHE	Examineur

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE
FLANDRES ARTOIS

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M.H. LEFEBVRE, M. PARREAU.

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT
ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER,
DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF,
LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL,
PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PAREAU, J. LOMBARD, M. MIGEON, J. CORTOIS.

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES
DE LILLE FLANDRES ARTOIS

M. A. DUBRULLE.

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	Electronique
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie Appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du Solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean-Pierre	Chimie-Physique
M. BOSCOQ Denis	Probabilités
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BREZINSKI Claude	Analyse Numérique

M. BRIDOUX Michel	Chimie-Physique
M. CELET Paul	Géologie Générale
M. CHAMLEY Hervé	Géotechnique
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. DAUCHET Max	Informatique
M. DEBOURSE Jean-Pierre	Gestion des Entreprises
M. DHAINAUT André	Biologie Animale
M. DOUKHAN Jean-Claude	Physique du Solide
M. DYMENT Arthur	Mécanique
M. ESCAIG Bertrand	Physique du Solide
M. FAURE Robert	Mécanique
M. FOCT Jacques	Métallurgie
M. FRONTIER Serge	Ecologie Numérique
M. GRANELLE Jean-Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. LABLACHE-COMBIER Alain	Chimie Organique
M. LACOSTE Louis	Biologie Végétale
M. LAVEINE Jean-Pierre	Paléontologie
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique Atomique et Moléculaire
M. LEROY Jean-Marie	Spectrochimie
M. LHOMME Jean	Chimie Organique Biologique
M. LOMBARD Jacques	Sociologie
M. LOUCHEUX Claude	Chimie Physique
M. LUCQUIN Michel	Chimie Physique
M. MACKE Bruno	Physique Moléculaire et Rayonnements Atmosph.
M. MIGEON Michel	E.U.D.I.L.
M. PAQUET Jacques	Géologie Générale
M. PETIT Francis	Chimie Organique
M. POUZET Pierre	Modélisation - calcul Scientifique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. SALMER Georges	Electronique
M. SCHAMPS Joel	Spectroscopie Moléculaire
M. SEQUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. TILLIEU Jacques	Physique Théorique
M. TOULOTTE Jean-Marc	Automatique
M. VIDAL Pierre	Automatique
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ALLAMANDO Etienne	Composants Electroniques
M. ANDRIES Jean-Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BART André	Biologie animale
M. BASSERY Louis	Génie des Procédés et Réactions Chimiques

Mme BATTIAU Yvonne
M. BEGUIN Paul
M. BELLET Jean
M. BERTRAND Hugues
M. BERZIN Robert
M. BKOUICHE Rudolphe
M. BODARD Marcel
M. BOIS Pierre
M. BOISSIER Daniel
M. BOIVIN Jean-Claude
M. BOUQUELET Stéphane
M. BOUQUIN Henri
M. BRASSELET Jean-Paul
M. BRUYELLE Pierre
M. CAPURON Alfred
M. CATTEAU Jean-Pierre
M. CAYATTE Jean-Louis
M. CHAPOTON Alain
M. CHARET Pierre
M. CHIVE Maurice
M. COMYN Gérard
M. COQUERY Jean-Marie
M. CORIAT Benjamin
Mme CORSIN Paule
M. CORTOIS Jean
M. COUTURIER Daniel
M. CRAMPON Norbert
M. CROSNIER Yves
M. CURGY Jean-Jacques
Melle DACHARRY Monique
M. DEBRABANT Pierre
M. DEGAUQUE Pierre
M. DEJAEGER Roger
M. DELAHAYE Jean-Paul
M. DELORME Pierre
M. DELORME Robert
M. DEMUNTER Paul
M. DENEL Jacques
M. DE PARIS Jean Claude
M. DEPREZ Gilbert
M. DERIEUX Jean-Claude
Melle DESSAUX Odile
M. DEVRAINNE Pierre
Mme DHAINAUT Nicole
M. DHAMELINCOURT Paul
M. DORMARD Serge
M. DUBOIS Henri
M. DUBRULLE Alain
M. DUBUS Jean-Paul
M. DUPONT Christophe
Mme EVRARD Micheline
M. FAKIR Sabah
M. FAUQUAMBERGUE Renaud

Géographie
Mécanique
Physique Atomique et Moléculaire
Sciences Economiques et Sociales
Analyse
Algèbre
Biologie Végétale
Mécanique
Génie Civil
Spectroscopie
Biologie Appliquée aux enzymes
Gestion
Géométrie et Topologie
Géographie
Biologie Animale
Chimie Organique
Sciences Economiques
Electronique
Biochimie Structurale
Composants Electroniques Optiques
Informatique Théorique
Psychophysiologie
Sciences Economiques et Sociales
Paléontologie
Physique Nucléaire et Corpusculaire
Chimie Organique
Tectonique Géodynamique
Electronique
Biologie
Géographie
Géologie Appliquée
Electronique
Electrochimie et Cinétique
Informatique
Physiologie Animale
Sciences Economiques
Sociologie
Informatique
Analyse
Physique du Solide - Cristallographie
Microbiologie
Spectroscopie de la réactivité Chimique
Chimie Minérale
Biologie Animale
Chimie Physique
Sciences Economiques
Spectroscopie Hertzienne
Spectroscopie Hertzienne
Spectrométrie des Solides
Vie de la firme (I.A.E.)
Génie des procédés et réactions chimiques
Algèbre
Composants électroniques

M. FONTAINE Hubert	Dynamique des cristaux
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie Sturcturale
M. GAMBLIN André	Géographie urbaine, industrielle et démog.
M. GLORIEUX Pierre	Physique moléculaire et rayonnements Atmos.
M. GOBLOT Rémi	Algèbre
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie Physique
M. GOURIEROUX Christian	Probabilités et Statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean-Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLLOT Jean	Chimie Organique
M. GUILBAULT Pierre	Physiologie animale
M. HENRY Jean-Pierre	Génie Mécanique
M. HERMAN Maurice	Physique spatiale
M. HOUDART René	Physique atomique
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et Statistiques
M. Jean Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme (I.A.E.)
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KREMBEL Jean	Biochimie
M. LANGRAND Claude	Probabilités et statistiques
M. LATTEUX Michel	Informatique
Mme LECLERCQ Ginette	Catalyse
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
Melle LEGRAND Solange	Algèbre
M. LEGRAND Pierre	Chimie
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne
M. LE MAROIS Henri	Vie de la firme (I.A.E.)
M. LEROY Yves	Composants électroniques
M. LESENNE Jacques	Systèmes électroniques
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique théorique
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU Jean-Marie	Optique-Physique atomique
M. MAIZIERES Christian	Automatique
M. MAURISSON Patrick	Sciences Economiques et Sociales
M. MESMACQUE Gérard	Génie Mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie Organique
M. MORTREUX André	Chimie Organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
Mme MOUYART-TASSIN Annie Françoise	Informatique
M. NICOLE Jacques	Spectrochimie
M. NOTELET Francis	Systèmes électroniques
M. PARSY Fernand	Mécanique

M. PECQUE Marcel
M. PERROT Pierre
M. STEEN Jean-Pierre

Chimie organique
Chimie appliquée
Informatique

Remerciements

Je remercie Monsieur le Professeur Vincent Cordonnier de me faire l'honneur de présider ce jury et d'avoir été l'initiateur de ce projet en me proposant un sujet de thèse.

Je remercie Messieurs Michel Lucas, Professeur à l'E.N.S.M. Nantes, et Jean-Paul Sansonnet, Directeur de Recherche CNRS au L.R.I., d'avoir porté une attention particulière à ce travail en acceptant d'en être rapporteurs.

Je remercie Monsieur Bernard Peroche, Professeur à l'école des Mines de Saint-Etienne, et Mademoiselle Ute Claussen, de la société AITEC, de leur participation au jury.

Je remercie chaleureusement Monsieur Michel Mériaux, Professeur à l'EUDIL, qui a encadré la majeure partie de ce travail. Je lui suis particulièrement gré d'avoir accepté d'être dérangé, jusqu'à plusieurs fois par jour, pour discuter et répondre à mes questions.

Je remercie toutes les personnes ayant travaillé sur I.M.O.G.E.N.E:

- Sylvain Karpf qui a réalisé le premier prototype dans le cadre de son D.E.A. et avec qui j'ai discuté de très nombreux après-midis pour définir I.M.O.G.E.N.E.

- Eric Nyiri qui réalise un excellent simulateur d'I.M.O.G.E.N.E.

- Samuel Degrande pour avoir conçu le premier circuit intégré.

- Isabelle Bourrust, Patrice Normand, Zhigang Nie, Vincent Lefèvre de leur contribution.

- Dominique Gonzalez sans qui ce projet ne serait pas I.M.O.G.E.N.E.

Je remercie tous les membres de l'Equipe Graphique, pour les échanges que nous avons eus, ainsi que toutes les personnes qui font du L.I.F.L un lieu de travail efficace et agréable.

Je remercie Monsieur Henri Glanc qui a reproduit, avec soin et célérité, les différentes versions de cette thèse.

Merci aussi à Bénédicte et Clémence.

SOMMAIRE

INTRODUCTION

I. PREMIERS ELEMENTS	6
Le sujet	7
Réaliser un processeur de visualisation d'images de synthèse	7
Le temps réel	7
Le Processus de Visualisation	8
Schéma global	8
Exclusion des méthodes de lancer de rayons et de radiosit�	9
Organisation classique des processeurs de visualisation	11
L'affichage, les �crans	12
Le Parall�lisme en synth�se d'images	14
Le Parall�lisme	14
Parall�lisme et Processus de Visualisation	14
Vocabulaire	16
II. PRESENTATION ANALYTIQUE DES MACHINES DE RENDU	18
Les Machines	19
Geometry System	19
Le circuit int�gr� PRC	21
Le circuit int�gr� SAGE	22
La machine Silicon Graphics 4D/240GTX	24
La machine Stellar GS1000	27
Autres machines commerciales	29
La Pixel Machine	30
CAP	32
Experts	33
La Machine de Torborg	35
GSP-NVS	38
Pixel-planes 4	41
Pixel-planes 5	45
Les Tableaux et Enseignements	48
Organisation des machines pr�sent�es	50
Parall�lismes et effets pipe-lines exploit�s	53
Algorithmes utilis�s	55
Le processus de conversion en pixels	57
Performances annonc�es	59
Conclusion	59

SOMMAIRE

III. VERS UN RENDU TEMPS REEL, ALGORITHMES ET ARCHITECTURE ..61

Organiser le processus de visualisation: Algorithmes	62
Les Opérations inter-objets	63
les Eclairments	76
Organisation globale d' une machine d'affichage en temps réel	80
La mémoire de trame	81
Les Phénomènes d'aliassage	82
La modélisation, les primitives visualisables	83
Les solutions matérielles pour réaliser un processeur d'affichage temps réel	84
Parallélisme Massif et Réalisation Matérielle	84
L'Approche Pixels	85
L'Approche Objets	89
Comparaisons des deux approches	91
Notre choix: L'Approche Objets	94
Le module pilote	95
Le module graphique	95
Conclusion du chapitre	95

IV. I.M.O.G.E.N.E96

Les Processeurs Elémentaires	97
Introduction	97
Définition du Processeur Elémentaire	97
Le Processeur Elémentaire de premier ordre	97
Le Processeur Elémentaire de second ordre	99
Solution matérielle	101
Choix du nombre de bits	102
Le Processeur Elémentaire programmable	102
Un autre Processeur Elémentaire	103
Réalisation	105
Les Processeurs Objets	106
Définition	106
Le Processeur Facette plane	106
Le processeur Polyèdre	107
Le Processeur Sphère	108
Le Processeur Cylindre	111
Autres Processeurs Objets	113
Réalisation, contraintes physiques	113
Le Décideur	115
Présentation	115
Elimination des parties cachées	115
Traitement des objets transparents	116

SOMMAIRE

Traitement des ombres portées	117
Visualisation directe d'arbre CSG	119
Problèmes d'ordre matériel	121
Le module Graphique	122
Schéma d'ensemble	122
Le Processeur d'Eclairage	122
L'Antialiassage	123
L'Unité de Commande	123
La liaison Module Pilote / Module Graphique	123
Schéma d'implantation	124
La mémoire de trame	125
Performances	126
Le Module Pilote	127
Présentation	127
L'unité de configuration	127
L'unité d'animation	128
L'unité de calculs	128
Parallélisation du Module Hôte	129
CONCLUSION.....	130
V. COMPLEMENTS	132
Index des Machines citées.....	133
Index des Algorithmes décrits	134
Eclairage	134
Elimination parties cachées	134
Ombres portées	134
Visualisation directe d'arbres C.S.G.....	134
Index des mots-clés	135
BIBLIOGRAPHIE	136

INTRODUCTION

Ce travail est l'étude de l'architecture d'une machine destinée à afficher des images de synthèse en temps réel. Ce projet se nomme I.M.O.G.E.N.E: Images au Moyen d'Objets Générés par Expressions NumériquEs.

L'énorme volume de calculs nécessaire pour visualiser des images de synthèse incite, depuis déjà longtemps, à réaliser des processeurs (souvent appelés modules graphiques) spécifiques à cette fonction. Les modules graphiques ont fait d'énormes progrès ces dernières années, principalement grâce aux avancées technologiques qui permettent d'augmenter les fréquences, l'intégration et par conséquent les performances.

Actuellement sont définis de nouveaux concepts architecturaux pour réaliser des modules graphiques dont les performances devraient dépasser le million de facettes par seconde. La machine I.M.O.G.E.N.E s'inscrit dans ce cadre avec la contrainte stricte suivante: recalculer intégralement l'image à chaque affichage, c'est-à-dire tous les vingt-cinquièmes de seconde pour visualiser des images animées.

Contrairement à ce qu'on pourrait croire, définir des modules graphiques très performants ne permet pas forcément d'obtenir le temps réel. Nous exposons les différentes solutions actuellement étudiées en mettant l'accent sur ces problèmes temporels.

Pour atteindre notre objectif nous proposons de définir un module graphique comprenant un processeur (appelé processeur objet) par primitive graphique. Le parallélisme massif, que nous préconisons, nous impose de faire une étude des algorithmes du processus de visualisation pour déterminer ceux qui autorisent un tel parallélisme. Nous en déduisons une organisation du processus de visualisation adapté à nos contraintes.

Afin de limiter le nombre de processeurs nécessaire pour visualiser une scène et d'augmenter la qualité des images, nous définissons des processeurs objets capables de traiter des primitives plus complexes que les facettes triangulaires utilisées par tous les modules graphiques existants.

Ce document comporte quatre parties:

Un premier chapitre nommé "Premiers Eléments" délimite le sujet de ce travail et fixe les bases sur lesquelles nous travaillons.

Le second chapitre est une présentation, suivie d'une comparaison des principales machines d'affichage définies ces dernières années pour visualiser efficacement des images de synthèse.

Le troisième chapitre présente une analyse des algorithmes du processus de visualisation. Cette analyse nous conduit à proposer différentes architectures massivement parallèles permettant d'afficher en temps réel.

Le dernier chapitre est une présentation de la machine I.M.O.G.E.N.E.

I PREMIERS ELEMENTS

L'objet de ce premier chapitre est de délimiter le sujet de ce travail; en particulier nous indiquons quels sont nos objectifs. Le choix de ces objectifs nous impose des limites que nous cernons avec précision. Nous donnons ensuite quelques généralités qui servent de base à cette thèse.

I.1 Le sujet

I.1.1 Réaliser un processeur de visualisation d'images de synthèse

Le travail que nous présentons dans cette thèse porte sur la conception d'un module graphique permettant de visualiser des images de synthèse en temps réel. Notre objectif n'est pas de réaliser une machine exploitable commercialement dans un proche avenir mais de proposer une solution architecturale qui permettra à terme d'afficher des images complexes en temps réel. Les processeurs actuellement commercialisés sont encore très loin d'avoir de telles performances et les concepts qu'ils utilisent ne permettront jamais d'obtenir le temps réel.

De très nombreux travaux sont actuellement faits pour exploiter le parallélisme dans les modules graphiques. L'approche que nous préconisons est d'utiliser un nombre très important de processeurs simples et identiques. Exploiter un parallélisme massif entraîne de nouvelles contraintes que nous tentons de répertorier. Envisager un parallélisme massif n'est raisonnable que depuis la commercialisation de logiciel de CAO permettant de définir des circuits intégrés spécifiques à une application.

Définir de nouveaux algorithmes pour le processus de visualisation n'est pas notre but. Nous déterminons parmi les algorithmes existants ceux qui se prêtent à une parallélisation massive et nous en déduisons une nouvelle organisation pour le processus de visualisation.

Notre objectif principal est d'obtenir une machine "temps réel vrai".

I.1.2 Le temps réel

Nous appelons "temps réel vrai" le fait que l'image soit intégralement redéfinie à chaque affichage, un nouvel affichage étant nécessaire tous les vingt-cinquièmes de seconde pour visualiser des objets animés.

Si nous utilisons un balayage entrelacé (voir I.2.4.2) à une fréquence de 50 Mhz, une image complète est effectivement affichée en un vingt-cinquième de seconde. En utilisant un balayage non entrelacé l'image est affichée cinquante fois par seconde. Le module graphique travaille alors deux fois plus vite. Dans ce travail nous fixons à 25 images par seconde l'animation temps réel. C'est le nombre d'images par seconde utilisé par les caméras. Si le module graphique travaille en non entrelacé, il est possible d'afficher deux fois chaque image.

La solution que nous préconisons est de n'utiliser aucune mémorisation dans le module graphique et de recalculer l'image complète à chaque balayage écran. Cette approche très stricte a les conséquences suivantes:

- La mémoire de trame utilisée par tous les modules graphiques actuels est abandonnée.
- Une solution ne sera choisie que si nous avons la certitude qu'elle pourra afficher des images animées.

Définir l'architecture d'une machine avec la contrainte "temps réel vrai" nous conduit à proposer une structure matérielle originale.

I.2 Le Processus de Visualisation

I.2.1 Schéma global

Le processus de visualisation d'images de synthèse peut se découper en cinq parties:

1) DEFINITION

Cette partie recouvre la création, la modification ou la suppression d'objets graphiques. Une fois définis, les objets sont placés dans la structure de données graphique.

2) PREPARATION

Pour préparer une image, les objets graphiques à afficher sont extraits de la base de données et fournis au(x) processeur(s) qui vont les traiter.

La préparation d'une image recouvre aussi la mise en place des différentes sources lumineuses, des objets et de l'observateur.

3) TRANSFORMATION

Les transformations géométriques sont l'ensemble des opérations à appliquer à chaque objet pour permettre sa visualisation. Ces opérations sont au minimum un changement de repère, du repère absolu dans celui de l'observateur, le découpage de l'objet dans le polyèdre de vision et la projection dans le repère écran.

4) CONVERSION

Cette partie réalise la transformation des objets graphiques en pixels (PIcture ELeMent). En général cela est fait en déterminant les pixels de contour de l'objet et en effectuant un remplissage de la zone délimitée.

Les valeurs ainsi déterminées sont stockées dans une mémoire appelée mémoire de trame.

5) AFFICHAGE

L'affichage consiste à envoyer les valeurs calculées en tous les pixels vers l'écran graphique.

De nombreuses tâches essentielles pour la visualisation ne figurent pas dans ce découpage. En effet elles peuvent être effectuées à différents stades du processus. Ces tâches sont, au minimum, l'élimination des parties cachées et les calculs d'éclairage. D'autres opérations inter-objets (traitement des ombres portées, visualisation d'objets partiellement transparents) sont aussi envisageables. Dans le chapitre II nous décrivons comment sont organisées les machines actuellement définies pour implanter de façon performante les cinq parties du processus de visualisation et nous précisons où sont réalisées les différentes tâches n'apparaissant pas dans notre découpage.

Pour atteindre une animation temps réel les quatre dernières parties du processus de visualisation doivent être réalisables dans le temps imparti à l'affichage d'une trame. L'objet du chapitre III sera de proposer une implantation matérielle du processus de visualisation permettant l'affichage en temps réel.

1.2.2 Exclusion des méthodes de lancer de rayons et de radiosité

1.2.2.1 Le lancer de rayons

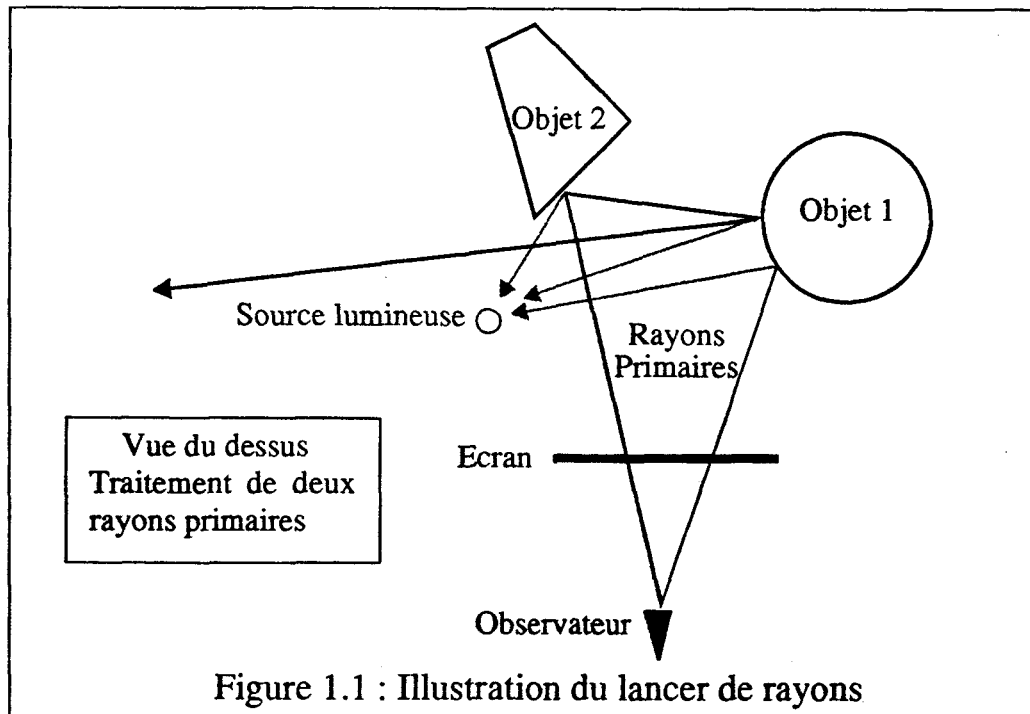
Cette méthode a été popularisée au début des années 80 par Whitted [White80][Roth82]. La méthode de lancer de rayons se présente comme suit:

1) De l'observateur on envoie un rayon primaire par pixel (voir Figure 1.1). Pour chacun de ces rayons on détermine ses points d'intersection avec tous les objets de la scène, ceci pour trouver l'objet le plus proche de l'observateur. Cette première phase revient à déterminer l'objet visible en chaque pixel c'est-à-dire à éliminer les parties cachées.

2) Dans un second temps on envoie de chaque point précédemment calculé plusieurs rayons secondaires:

- Un vers chaque source lumineuse qui, s'il ne rencontre aucun objet jusqu'à celle-ci permet de préciser si l'objet est éclairé par la source. Ces rayons permettent de prendre en compte les ombres portées.
- Un dans la direction symétrique de celle du rayon primaire par rapport à la normale à l'objet pour calculer les réflexions spéculaires. Ces rayons sont placés sur la figure 1.1.
- Eventuellement un dans la direction de réfraction pour les objets transparents.

3) On peut poursuivre le suivi des rayons de réflexion et de réfraction pour prendre en compte les objets éclairés indirectement par d'autres objets. Un exemple de rayon de ce type est représenté Figure 1.1.



Les calculs pour afficher une scène, même de complexité raisonnable, sont considérables. Les temps pour une image se comptent en minutes, voire en heures.

Les calculs les plus nombreux sont des calculs d'intersection de droites (les rayons) avec les objets généralement définis par des équations. Pour diminuer le nombre de calculs complexes, on évalue l'intersection des rayons avec des englobants simples des objets (sphères, parallélépipèdes).

De très nombreux travaux sont en cours pour réaliser des machines parallèles de lancer de rayons.

I.2.2.2 Paralléliser le lancer de rayon

La plupart des machines parallèles pour le lancer de rayon utilisent le procédé suivant: la scène tridimensionnelle englobant l'ensemble des objets est découpée en plusieurs parties gérées chacune par un processeur. Ces parties sont de taille fixe ou variable pour équilibrer la charge en objets sur les différents processeurs. Au cours du traitement les rayons transitent dans le réseau de processeurs. Chaque processeur traite les rayons atteignant des objets dans sa zone, et envoie les rayons engendrés qu'il ne peut traiter aux processeurs voisins.

Citons quelques machines de ce type :

La machine CM² [Cleary86] de l'université de Calgary est constituée d'un réseau matriciel de 10*10 processeurs Motorola 68000.

La machine Voxar [PitoCD89][CaubDG88] utilise un réseau de Transputers.

Une implémentation d'un algorithme de lancer de rayons est développée sur un Hypercube d'Intel [BouaPr88][Priol89].

→ Cette parallélisation par découpage de la scène nécessite d'avoir des liaisons entre processeurs ayant des débits considérables pour les transferts de rayons. Plus le découpage est fin, moins chaque processeur traite d'objets et donc plus les communications entre processeurs sont importantes. Les performances maximales pour ce genre de découpage sont atteintes en utilisant une centaine de processeurs. Le gain en temps est du même ordre de grandeur. Les temps de calcul sont de l'ordre de quelques secondes par image pour les machines que nous avons citées.

L'Equipe de l'I.R.I.S.A propose une autre solution permettant de limiter les transferts entre processeurs, en faisant transiter les objets de la scène et non les rayons. Ils utilisent un mécanisme d'adressage par pages où la mémoire répartie est considérée comme une virtuelle mémoire partagée [BadoPr91].

Le lancer de rayons est une méthode qui permet d'obtenir des images de très bonne qualité au prix d'un grand nombre de calculs. Il n'est pas envisageable d'afficher des images en temps réel avec cette technique dans un avenir proche.

Actuellement des études sont menées pour afficher rapidement, avec cette méthode, en utilisant un processeur par pixel (le Projet RC² [AtamML91]).

I.2.2.3 Méthodes de radiosité

La technique de radiosité [GoraTG84] considère que toutes les surfaces sont modélisées par de petits polygones plans. Le flux émis en tout point d'un polygone est constant.

On appelle radiosité d'une surface la quantité de lumière quittant cette surface. Elle s'exprime pour un polygone par la formule:

$$R_i = E_i + \rho_i * \sum_{j=1}^N F_{ij} * R_j \quad [1]$$

E_i est la quantité de lumière émise par la surface i .

ρ_i le coefficient de réflexion de la surface.

F_{ij} facteur de forme géométrique (fraction du flux émis par le polygone i reçue par le polygone j). Les facteurs de formes sont des intégrales doubles sur les surfaces et sont fonction des angles entre les facettes.

Le calcul d'éclairement pour une scène donnée se fait en deux temps:

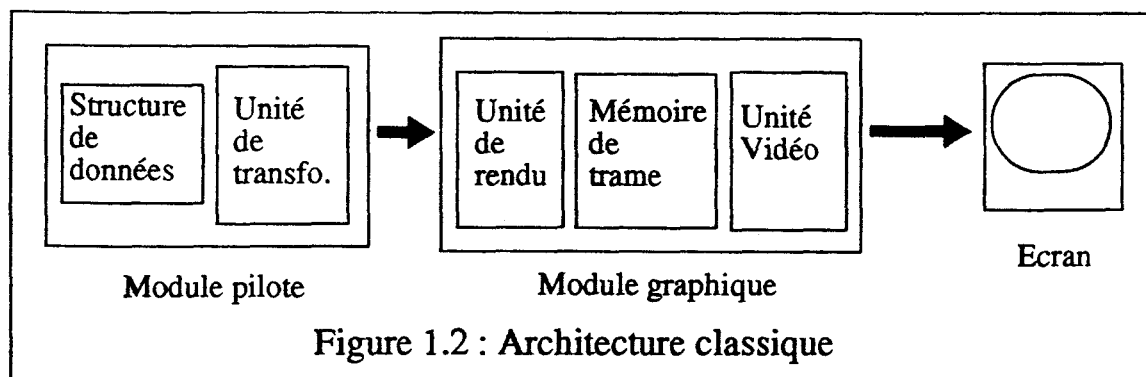
1) calcul de tous les facteurs de forme. Cohen [CohGr85] a introduit une méthode permettant de déterminer ces coefficients sans calculer les intégrales.

2) résolution du système linéaire formé par les N équations de la forme de l'expression [1], N étant le nombre de facettes de la scène. Ceci pour déterminer les radiosités R_i de chaque surface polygonale.

Pour une scène de N facettes il faut calculer N^2 facteurs de forme et résoudre un système linéaire de N équations à N inconnues. Un tel procédé, bien que parallélisable, n'est pas actuellement utilisable en temps réel.

I.2.3 Organisation classique des processeurs de visualisation

La figure 1.2 représente l'architecture des machines de visualisation actuellement commercialisées [Martin86]:



L'unité de transformation gère la préparation à l'affichage et les transformations géométriques. L'unité de rendu s'occupe de la conversion des objets en pixels et de toutes les opérations de rendu. Le plus souvent les calculs d'éclairement sont effectués par le module pilote, l'unité de rendu faisant de l'interpolation sur ces valeurs exactes.

Le module pilote est en général un logiciel sur un processeur d'usage général. Le module graphique est réalisé par automates câblés ou VLSI spécialisés dans les machines les plus performantes.

I.2.4 L'affichage, les écrans

I.2.4.1 le principe des écrans TRC

Les écrans TRC, Tubes à Rayons Cathodiques, sont composés d'un écran de luminophores (anode), d'un canon à électrons (cathode) et d'un système de déviation verticale et horizontale des électrons. Les faisceaux envoyés par le canon à électrons excitent les luminophores voulus à l'aide du système de déviation (voir Figure 1.3).

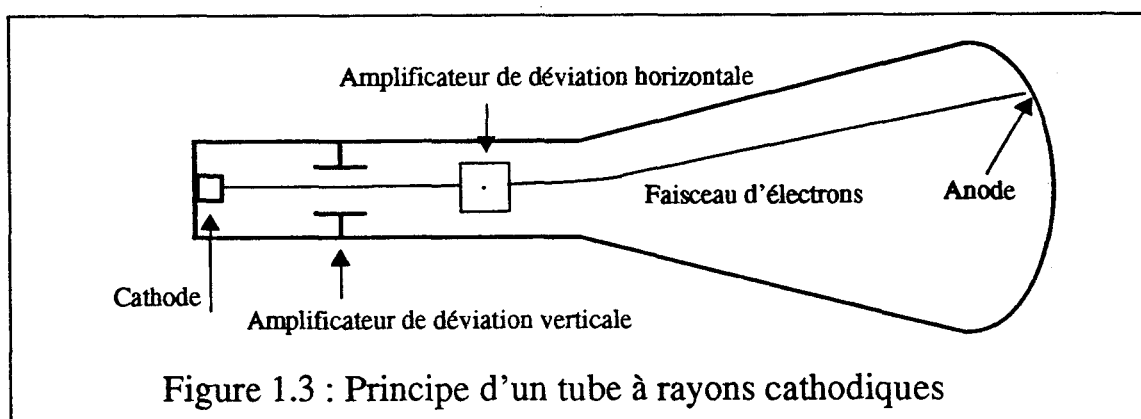


Figure 1.3 : Principe d'un tube à rayons cathodiques

I.2.4.2 Le principe du balayage vidéo

L'écran est balayé en continu ligne par ligne et est rafraîchi systématiquement à une fréquence compatible avec la persistance rétinienne (voir Figure 1.4).

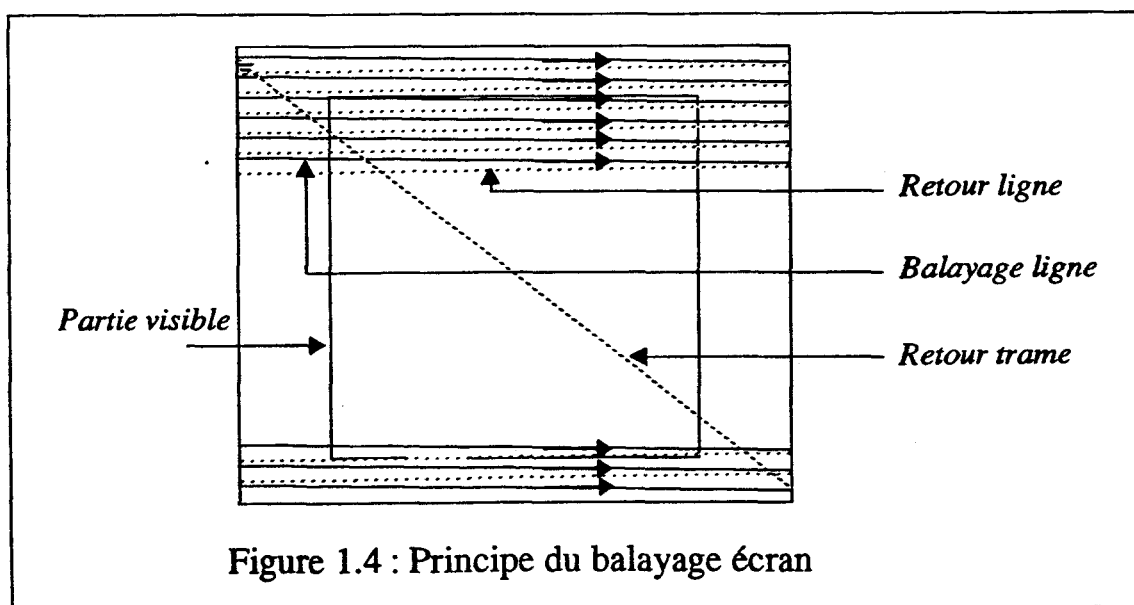


Figure 1.4 : Principe du balayage écran

Avec une fréquence de 50 Hz et en utilisant une zone visible 512x512, il faut afficher une ligne tous les 38 micro-secondes et donc afficher un point tous les 60 nano-secondes environ. Ces valeurs sont imposées par l'utilisation d'un retour ligne de 7 micro-secondes et d'un retour trame de 600 micro-secondes.

Une solution pour diminuer la fréquence est d'utiliser un balayage entrelacé, c'est à dire qu'à chaque trame seulement une ligne sur deux est traitée. Valoriser totalement l'écran nécessite deux balayages de trame. C'est cette solution qui est utilisée par les écrans de type télévision.

Le tableau 1.1 récapitule les fréquences nécessaires pour des écrans 512x512 et 1024x1024 avec un balayage entrelacé ou non [Degran89].

	entrelacé	non entrelacé
512x512	Pixel: 134 ns soit 7,4 MHz Ligne: 75,8 μ s soit 13,2 kHz	Pixel: 60 ns soit 16,6 MHz Ligne: 38 μ s soit 26,4 kHz
1024x1024	Pixel: 30 ns soit 33,1 MHz Ligne: 38 μ s soit 26,4 kHz	Pixel: 11,5 ns soit 85,7 MHz Ligne: 19 μ s soit 52,8 kHz

Tableau 1.1: Fréquence pixel

I.2.4.3 Les Autres solutions

Il existe d'autres types d'écrans que les tubes cathodiques. Les principaux sont les systèmes dits à écran plat. Ces écrans sont soit émetteurs comme les écrans à plasma ou à diodes électroluminescentes, soit non-émetteurs comme les écrans à cristaux liquides.

Avec ces solutions il sera possible d'envisager d'autres méthodes d'affichage que le balayage vidéo. Ces écrans ont toutefois de nombreux défauts:

- Il est difficile de réaliser des écrans de grande taille.
- Les coûts sont très élevés.
- Les couleurs obtenues sont peu satisfaisantes.

Il est donc raisonnable de penser que les tubes cathodiques et le balayage de trame resteront hégémoniques sur le marché des écrans encore au moins une dizaine d'années.

I.3 Le Parallélisme en synthèse d'images

I.3.1 Le Parallélisme

Dès que plusieurs processeurs traitent en commun un processus, il y a parallélisme. Cette notion recouvre des organisations très différentes que nous divisons en trois catégories pour un processus P s'exécutant sur un ensemble de données (di).

1) Le Pipe-line

Le processus est divisé en une chaîne de sous-processus P1, ... Pn à réaliser séquentiellement. Le flot des données est traité dans une chaîne de processeurs M1,... Mn. Chaque processeur Mi exécute le processus Pi. Les données progressent de processeur en processeur en cours de traitement.

Cette solution n'est envisageable que si le processus initial se divise bien en un sous-ensemble de processus dont les temps de traitement sont voisins. Le rythme de progression est imposé par le processeur le plus lent. Des mémoires tampons peuvent être intercalées entre les processeurs pour compenser ponctuellement les différences de vitesses d'exécution.

2) Le Parallélisme sur les données

Le processus P est intégralement implanté sur un ensemble de processeurs Mi. Les données à traiter sont réparties sur les différents processeurs. Il est alors indispensable d'utiliser un mécanisme de répartition des données aux processeurs.

Cette solution devient difficilement utilisable si les traitements des données ne sont pas disjoints, c'est-à-dire nécessitent des communications et des synchronisations entre processeurs.

3) La Parallélisation du processus

Le processus P est divisé en un ensemble de sous-processus fonctionnant en parallèle sur différents processeurs. Cette technique de parallélisation est délicate à mettre en oeuvre car elle remet en cause la structure séquentielle des algorithmes. De plus elle nécessite des liens de communication entre tous les processeurs, ceci pose d'énormes problèmes matériels quand le nombre de processeurs devient grand.

Cette solution de parallélisation est intéressante quand le traitement d'une donnée par le processus est très long.

I.3.2 Parallélisme et Processus de Visualisation

Analyser les parallélismes exploités par une implantation donnée du processus de visualisation est fort difficile. Les raisons en sont multiples, les principales sont:

- Les données sont de deux types, les objets graphiques et les pixels de l'écran.

Le processus de visualisation est composé de nombreux sous-processus que nous avons choisis de regrouper en cinq phases dans le paragraphe I.2.1. Les différentes phases n'offrent pas les mêmes possibilités de parallélisation. De plus, pour chacune de ces phases du processus de visualisation, de nombreux algorithmes existent, certains aisément

parallélisables, d'autres absolument pas.

Nous classons en quatre catégories les différentes propositions de parallélisation du processus de visualisation.

1) Le pipe-line objets

Le processus est implanté en un pipe-line de processeurs dans lequel progressent les objets graphiques.

2) Le parallélisme objets

Le processus est implanté totalement sur un ensemble de processeurs traitant chacun une partie des objets.

3) Le pipe-line pixels

Le processus est implanté en un pipe-line de processeurs dans lequel progressent les pixels.

4) Le parallélisme pixels

Le processus est implanté complètement sur un ensemble de processeurs qui traitent chacun une partie de l'écran.

C'est à partir de cette classification que nous allons construire notre travail d'analyse.

Il est malheureusement impossible de classer directement les propositions pour l'intégralité du processus de visualisation avec les catégories que nous venons d'établir. Toute solution utilise plusieurs de ces parallélismes à différentes phases du processus de visualisation. Dans l'analyse, il faudra être précis sur la partie du processus de visualisation qui est parallélisée.

Un autre critère important pour analyser le parallélisme dans le processus de visualisation est l'ordre de grandeur du parallélisme. Celui-ci recouvre aussi bien une machine bi-processeurs qu'une machine comprenant un millier de processeurs.

Nous n'avons pas fait figurer dans cette classification la "parallélisation du processus". En effet l'utilisation du parallélisme en synthèse d'image s'impose non parce que l'exécution du processus pour une donnée est trop complexe, mais parce que le nombre de données à traiter est considérable. Une scène peut contenir plus de 10 000 objets graphiques, un écran 1000*1000 contient un million de pixels.

Il existe bien évidemment des algorithmes de rendu se prêtant à une "parallélisation du processus", par exemple l'algorithme d'élimination des parties cachées de Warnock (présenté au paragraphe III.1.1.1.1). Dans cet algorithme, l'angle d'étude n'est ni orienté pixels ni orienté objets mais repose sur un découpage récursif du volume englobant la scène à visualiser, ce qui impose de considérer que la donnée est une seule entité comprenant l'ensemble des objets. Une implémentation parallèle de cet algorithme serait à classer dans la catégorie: "Parallélisation du processus".

Toutes les propositions de réalisations matérielles utilisent des algorithmes exploitant des parallélismes ou effets pipe-line sur les objets ou sur les pixels, la parallélisation du processus n'est alors qu'une conséquence. Nous reviendrons en détail sur ce problème dans le chapitre III.

I.3.3 Vocabulaire

I.3.3.1 Granularité

Le degré de parallélisme, parfois appelé granularité, des machines peut être classé en trois catégories:

- Faible si le nombre de processeurs est inférieur à une cinquantaine.
- Moyen si le nombre de processeurs est compris entre cinquante et deux cent.
- Massif si le nombre de processeurs est supérieur à deux cent.

I.3.3.2 SIMD, MIMD

Les organisations parallèles sont caractérisées comme SIMD ou MIMD [Flynn72].

SIMD Single Instruction stream, Multiple Data stream : ce sont des machines ayant plusieurs unités de traitement contrôlés par la même unité de commande. Les différentes unités exécutent le même flot d'instructions au même rythme sur des données distinctes.

MIMD Multiple Instruction stream, Multiple Data stream : les différentes unités de traitement possèdent chacune une unité de commande. Les différentes unités peuvent traiter des flots d'instructions distincts sur des données distinctes.

I.3.3.3 Efficacité, Accélération

Un processeur donné met un temps t pour effectuer une tâche. Si N processeur mettent un temps t' pour réaliser la même tâche on peut définir l'efficacité et l'accélération comme suit:

$$\text{Efficacité} = t / (t' * N)$$

$$\text{Accélération} = N * \text{Efficacité} = t / t'$$

Illustrons ce vocabulaire sur un exemple (voir Figure 1.5)

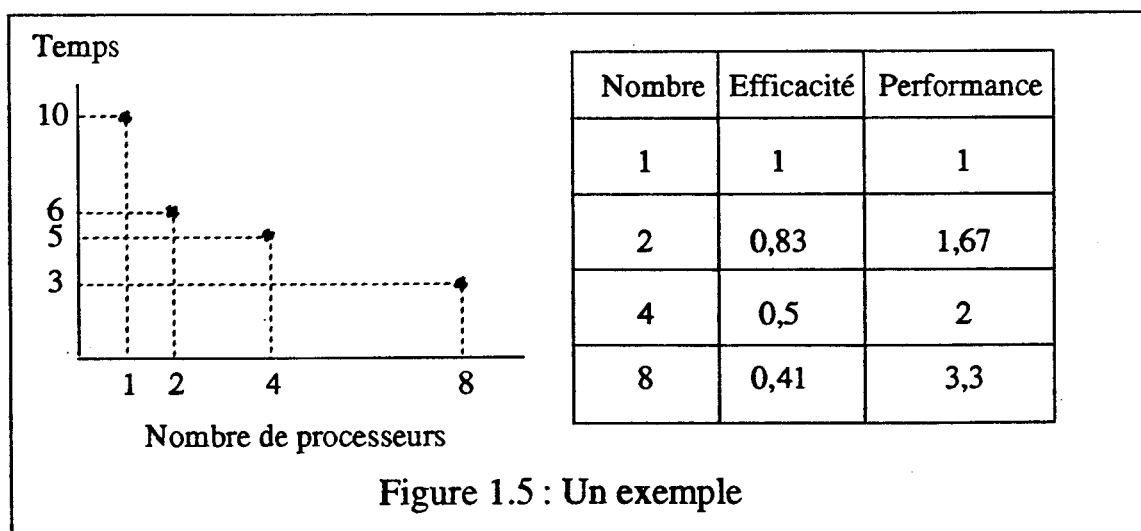


Figure 1.5 : Un exemple

Sur cet exemple l'efficacité diminue quand le nombre de processeurs croît, toutefois

les Accélérations augmentent.

Pour obtenir une machine d'affichage temps réel, seule l'Accélération est à prendre en compte, l'Efficacité n'est pas, dans ce cas, un critère essentiel.

Pour évaluer a priori l'Efficacité et les Accélérations des algorithmes et des machines d'affichage nous allons exprimer l'Efficacité comme le produit de deux valeurs qu'il est possible d'estimer de façon théorique.

Efficacité = Rendement * Ratio-coopération

Le Rendement est le quotient du nombre d'opérations utiles sur le nombre total d'opérations.

Le Ratio-coopération est le rapport du temps de travail effectif sur le temps total d'exécution, c'est à dire en ajoutant les temps de communication entre processeurs, le temps d'attente et le temps passé à des tâches n'existant pas avec un processeur unique mais qui s'imposent en utilisant plusieurs processeurs.

Illustrons ces notions sur deux exemples.

Exemple 1:

Un processeur élémentaire de tracé de segment traite un pixel toutes les 10 ns. Pour augmenter la vitesse de traitement on utilise une matrice de 16 processeurs pour traiter des blocs de 4*4 pixels.

Un préprocesseur détermine les blocs 4*4 de la mémoire d'image à traiter pour un segment donné. Il met 5 ns pour choisir un bloc. Il n'y a aucune communication entre les différents processeurs élémentaires. Le temps total de traitement pour un bloc de 16 pixels est de 10+5 = 15 ns.

Le Ratio-coopération est donc sur cet exemple de 10 / 15.

On peut par ailleurs estimer que sur un bloc carré de 16 pixels, en moyenne 6 appartiennent au segment à tracer. Le Rendement est donc de 6 / 16.

$$\text{Efficacité} = \frac{10}{15} * \frac{6}{16} = 0,25$$

L'Accélération de cette machine utilisant 16 processeurs en parallèle est donc de 4.

Exemple 2 :

Exprimons avec le vocabulaire que nous venons de définir les résultats sur les machines parallèles de lancer de rayons par division de la scène tridimensionnelle.

Pour ces machines le Rendement est égal à 1, toutes les opérations effectuées sont utiles. Par contre le Ratio-coopération diminue extrêmement vite quand le découpage devient fin. Chaque rayon transite par de plus en plus de processeurs avant de rencontrer un objet. Ceci à tel point que l'Accélération décroît à partir d'un certain nombre de processeurs.

II PRESENTATION ANALYTIQUE DES MACHINES DE RENDU

Nous présentons dans ce chapitre différentes machines conçues pour visualiser des images de synthèse. Nous dressons ensuite des tableaux permettant d'analyser et de comparer ces machines. Parallèlement à ces tableaux, nous mettons en évidence certaines contraintes qui nous permettent de mieux cerner les solutions envisageables pour définir un module d'affichage temps réel.

II.1 Les Machines

Le plan que nous allons suivre maintenant pour décrire chacune des machines est le suivant:

i) Une description globale de l'architecture de la machine illustrée par un schéma. On précisera les algorithmes utilisables par cette machine. On insistera sur la méthode utilisée pour générer les pixels.

ii) Une analyse des formes de parallélisme utilisées en reprenant la terminologie définie dans le premier chapitre. On mettra l'accent sur le degré de parallélisme utilisé.

iii) Des précisions sur les éventuels circuits intégrés utilisés.

iv) Un résumé des principales caractéristiques et une mise en évidence de l'originalité de la proposition.

Nous allons présenter 12 machines qui sont représentatives des différentes solutions architecturales envisagées à ce jour. Nous pouvons regrouper ces machines en 4 classes:

A) les processeurs dédiés réalisés en VLSI

1 Geometry System

2 Le circuit intégré PRC

3 Le circuit intégré SAGE

B) les machines commerciales de haut de gamme

4 La Silicon Graphics 4D/240GTX

5 La Stellar GS1000

6 Autres machines

C) les machines utilisant des processeurs généraux en parallèle

7 La Pixel Machine

8 CAP

D) les machines prototypes

9 Experts

10 La machine de Torborg

11 La machine GSP-NVS

12 Pixel-Planes 4

13 Pixel-Planes 5

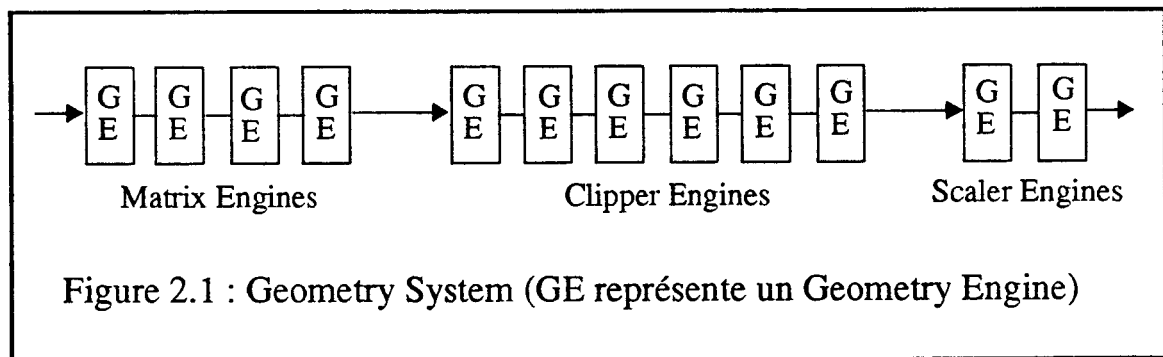
Tous les algorithmes cités dans ce chapitre, généralement sous le nom de leur concepteur, seront développés dans les chapitres suivants. Une liste de ces algorithmes figure à la fin de ce document avec les références bibliographiques et le numéro du paragraphe où ils sont explicités.

II.1.1 Geometry System

Le circuit intégré "Geometry Engine" [Clark82] a été développé pour accélérer la phase de transformation du processus de visualisation.

Ce circuit intégré effectue des calculs flottants en parallèle sur les 4 composantes d'un segment (ou vecteur) en coordonnées homogènes. Clark appelle "Geometry System" (voir Figure 2.1) un pipe-line de 12 circuits intégrés "Geometry Engine" qui effectue l'ensemble des transformations indispensables pour chaque segment. Les tâches sont découpées comme suit:

- Les 4 premiers circuits intégrés "Matrix Engines" réalisent le changement de repère du repère de la scène dans celui de l'observateur par calculs matriciels.
- Les 6 suivants "Clipper Engines" effectuent le découpage des segments dans le polyèdre de vision. Un circuit intégré réalise le découpage pour un coté du polyèdre.
- Les deux derniers "Scaler Engines" effectuent la projection dans le repère deux dimensions de l'écran.



Les parallélismes utilisés par cette machine sont les suivants:

- i) Un parallélisme objets de très bas niveau, les 4 coordonnées d'un segment sont traitées en parallèle.
- ii) Un pipe-line objets de 12 étages en utilisant plusieurs circuits intégrés pour effectuer la totalité des opérations de transformation.

Le circuit intégré "Geometry Engine" est un circuit ayant 40 broches, composé de 4 blocs identiques traitant chacun une composante.

Ce circuit intégré est la seule proposition que nous étudierons s'intéressant à la partie transformation du processus de visualisation. Il permet d'obtenir de bonnes performances: 100000 segments traités par seconde pour un circuit intégré et 4 fois plus en utilisant le pipe-line de 12 circuits intégrés. Il est actuellement utilisé sur certaines stations graphiques, par exemple la station IRIS de Silicon Graphics. Vue la rapide augmentation des performances des processeurs généraux de calculs (processeur de signal, processeur vectoriel), les circuits intégrés de ce type deviendront rapidement obsolètes.

II.1.2 Le circuit intégré PRC

PRC (Polygon Rendering Chip) [SwanTh86] est un circuit permettant d'afficher rapidement des polygones 3D avec ombrage.

Les entrées de ce circuit intégré sont les valeurs X, Y, Z, R, V et B en tous les sommets du polygone à visualiser. A partir de ces valeurs le circuit intégré fournit en chaque pixel où le polygone est présent, la profondeur de celui-ci et les valeurs R, V et B à afficher. Ces calculs sont faits par interpolations linéaires sur les six variables. Cela revient pour l'ombrage à utiliser la méthode de Gouraud.

Ce circuit intégré est composé de 4 unités fonctionnant en pipe-line (voir Figure 2.2):

- L'unité "Edge" qui détermine les extrémités des segments horizontaux composant le polygone.
- L'unité "Setup" qui calcule les paramètres d'interpolations.
- L'unité "Interpolator" qui effectue tous les calculs d'interpolations.
- L'unité "Fill" qui génère tous les pixels.

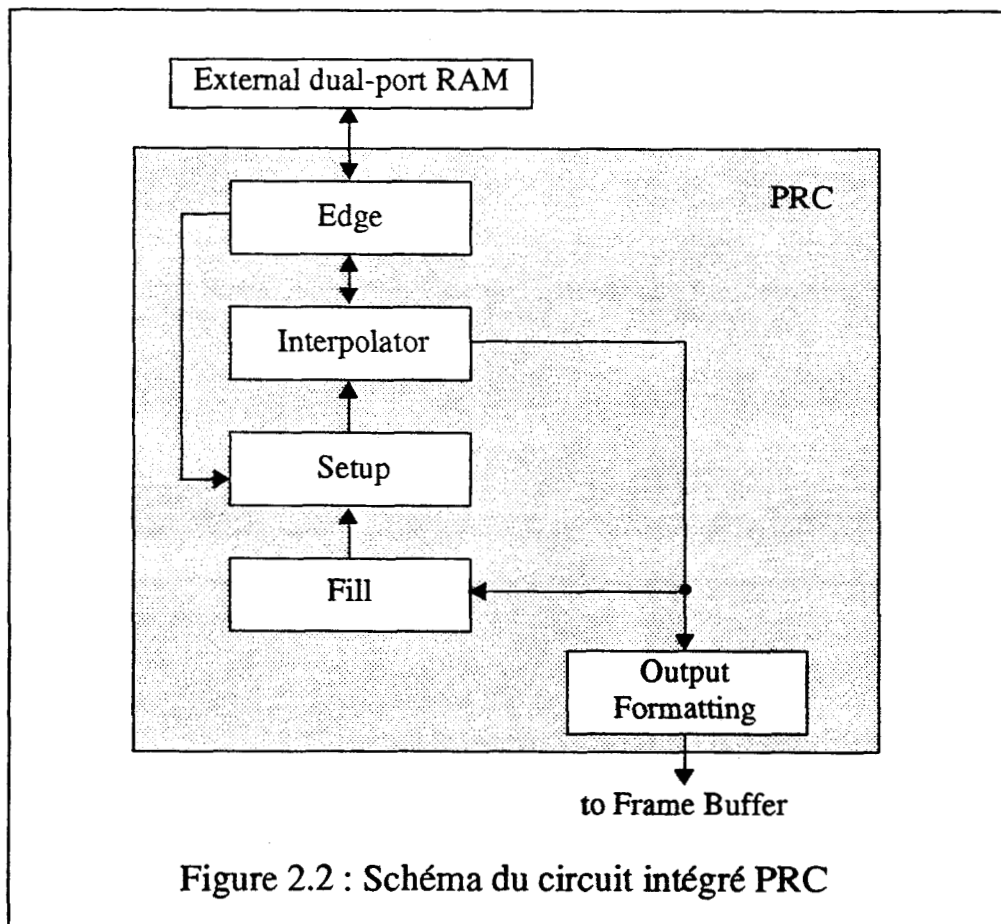


Figure 2.2 : Schéma du circuit intégré PRC

Ce circuit intégré utilise le parallélisme suivant:

Un pipe-line objets entre les différents segments horizontaux d'un objet.

Ce processeur est composé de 150.000 transistors environ et travaille à la fréquence de 20 Mhz. Il est implanté dans les stations de CAO Hewlett-Packard.

Ce circuit intégré utilise peu le parallélisme, mais son organisation interne et sa réalisation VLSI lui permettent de générer les pixels au rythme d'un toutes les 50 ns. Il met environ 90 microsecondes pour engendrer un triangle de surface 1.000 pixels.

Il serait tout à fait possible de faire travailler plusieurs de ces circuits intégrés en parallèle. La vitesse serait multipliée par le nombre de circuits intégrés. Mais se produiraient des problèmes de conflit d'accès à la mémoire de trame.

II.1.3 Le circuit intégré SAGE

SAGE [GharGH88] est un circuit intégré CMOS contenant un million de transistors implémentant un pipe-line de 256 processeurs élémentaires "Pixel Processor" gérant chacun un pixel (voir Figure 2.3). Un circuit intégré s'occupe d'une partie d'une ligne écran.

Chaque "Pixel Processor" incrémente linéairement des valeurs X, Z, R, G et B, ce qui permet de tracer des segments horizontaux en trois dimensions et d'effectuer un ombrage suivant la méthode de Gouraud. De plus ces processeurs effectuent l'élimination des parties cachées en utilisant un Z-buffer par ligne.

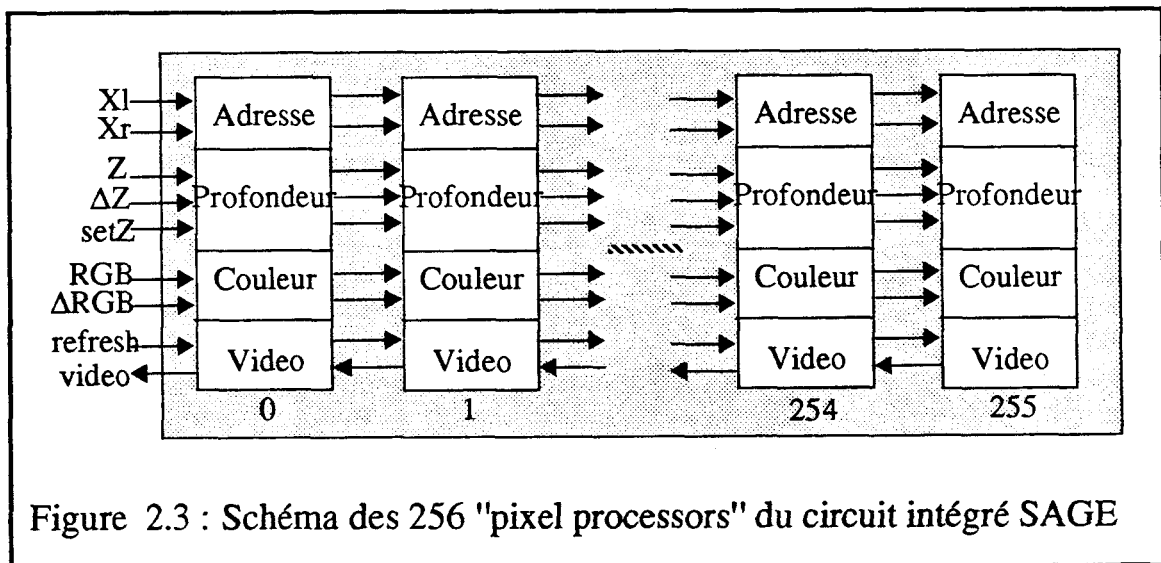


Figure 2.3 : Schéma des 256 "pixel processors" du circuit intégré SAGE

Les concepteurs de ce circuit intégré proposent d'en utiliser 4 pour réaliser une unité complète de génération des pixels. Deux configurations sont alors envisageables: les utiliser en série ou en parallèle (voir Figure 2.4).

4 circuits intégrés SAGE peuvent traiter 256 segments horizontaux au rythme du balayage écran pour un écran 1024*1024 en configuration parallèle. S'il y a davantage de segments sur une des lignes, il est indispensable d'utiliser une mémoire de trame.

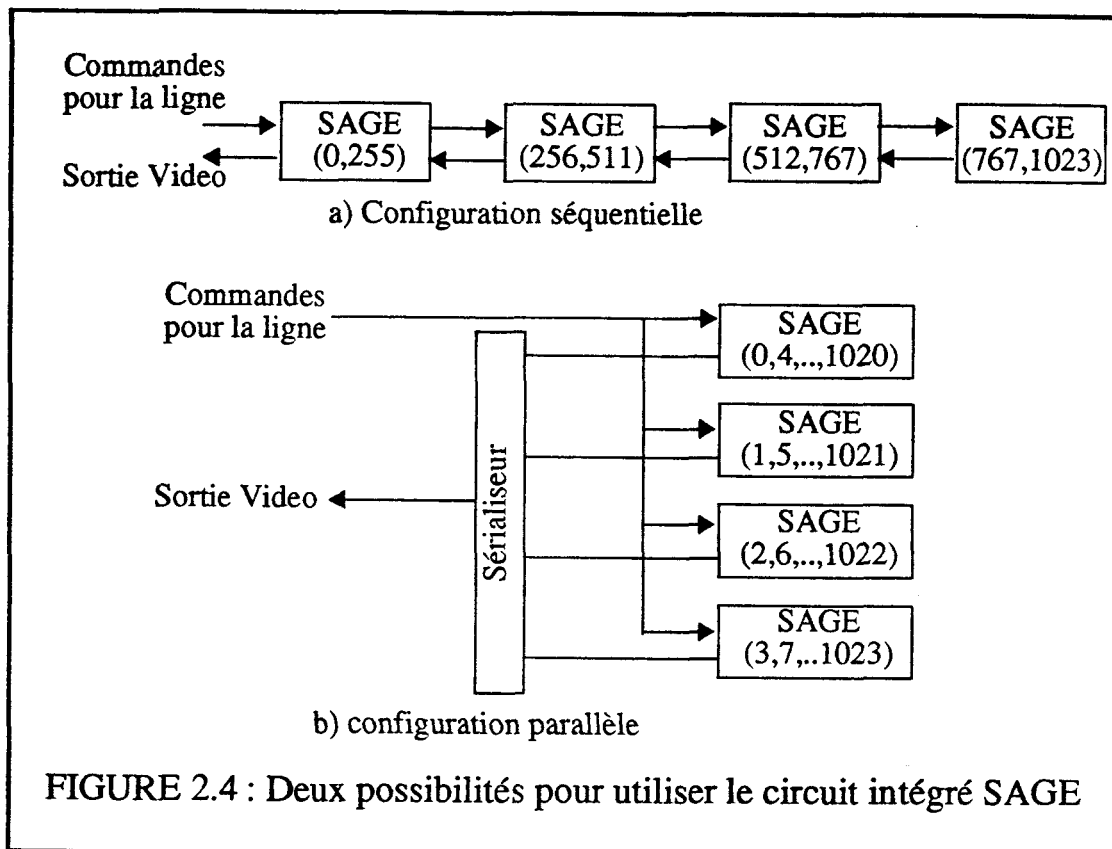
Pour visualiser des faces planes 3D, on utilise 6 processeurs d'interpolation verticale qui calculent les valeurs Xl, Xr, Z, R, G et B pour chacun des segments horizontaux de chaque face afin d'alimenter les circuits intégrés SAGE pour exploiter au maximum leurs possibilités. La mémoire interne de ces processeurs d'interpolation est de 512 polygones.

Ces circuits intégrés permettent d'utiliser les parallélismes suivants:

i) Un pipe-line objets de 256 étages pour générer les pixels des segments horizontaux, effectuer l'élimination des parties cachées et calculer l'ombrage.

ii) Un parallélisme pixels entre les pixels des différents circuits intégrés quand ceux-ci sont utilisés en parallèle.

iii) L'utilisation de tels circuits intégrés permet une parallélisation des opérations situées au niveau du pixel.



Pour des scènes comprenant un grand nombre d'objets il n'est pas évident que les processeurs d'interpolation verticale puissent alimenter les circuits intégrés SAGE en les exploitant au maximum.

Le principal intérêt de cette réalisation est la très importante intégration en VLSI qui permet de générer les pixels à une vitesse remarquable. Pour des segments horizontaux de longueur moyenne 40 bits, le temps pour traiter un objet en un pixel est de 1 ns.

Les concepteurs de ce circuit intégré pensent pouvoir afficher 1 000 000 de polygones de 1000 pixels de surface avec ombrage de Gouraud et Z-buffer par seconde en utilisant la configuration 4 circuits intégrés en parallèle.

II.1.4 La machine Silicon Graphics 4D/240GTX

Cette machine [Akeley89][AkelJe88] est l'une des plus performantes actuellement commercialisée.

Elle est composée d'une mémoire centrale, d'un module graphique, d'un module d'entrées/sorties et de 4 processeurs (voir Figure 2.5) reliés entre eux par deux bus, "Sync bus" et "MPlink bus".

Le "Sync bus" permet de synchroniser les 4 processeurs pour les utiliser au mieux en parallèle par adressage direct dans les mémoires des processeurs. Le "MPlink bus" supporte les transferts de données entre les 4 processeurs, le module graphique, la mémoire et les entrées/sorties. Des caches sont utilisés pour optimiser l'utilisation de ce bus.

Chaque processeur est composé d'une unité centrale RISC, d'un coprocesseur de calculs flottants, d'une mémoire et de caches. Ces unités sont reliées entre elles par un bus interne au processeur. L'unité centrale est un processeur 32 bits à 25 MHz.

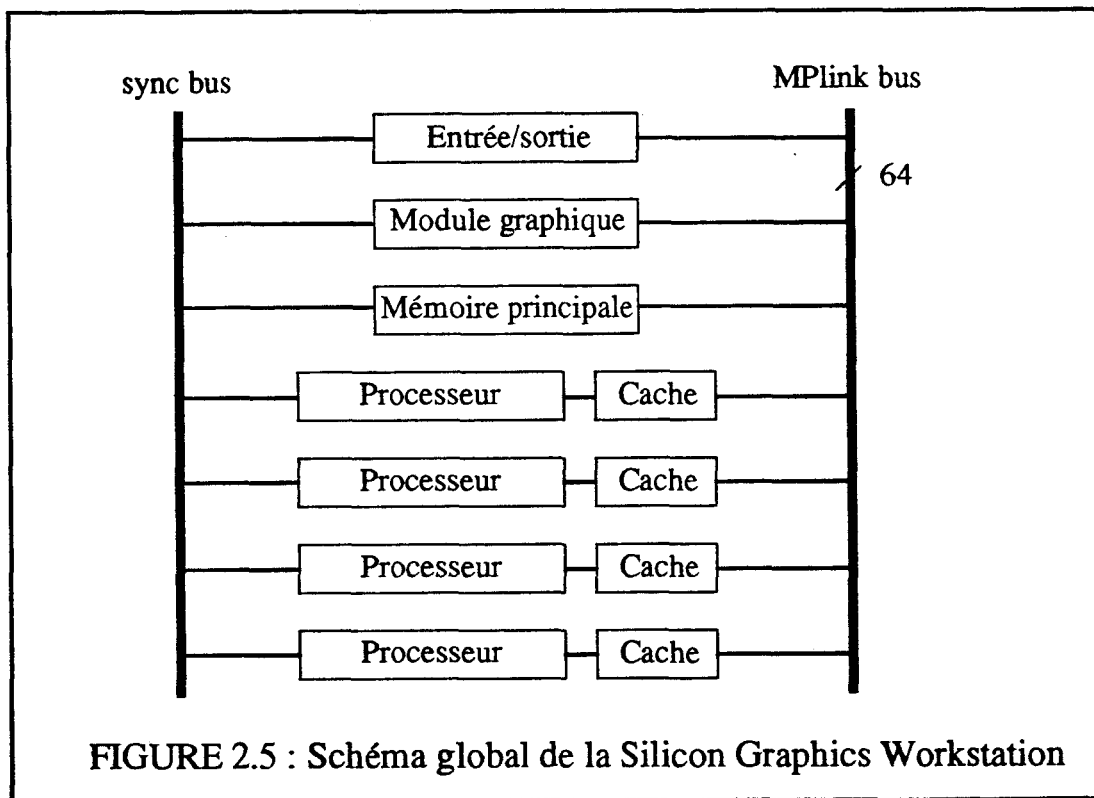


FIGURE 2.5 : Schéma global de la Silicon Graphics Workstation

Le module graphique de la Silicon Graphics Workstation est composé de quatre unités différentes fonctionnant en pipe-line (voir Figure 2.6) :

- Une unité de géométrie.

Les calculs géométriques sur les sommets des polygones sont effectués en coordonnées homogènes par cinq composants "Geometry Engine".

- Une unité de conversion.

Cette unité décompose en plusieurs étapes les polygones en segments verticaux

(spans). Le "Polygon processor" décompose les polygones en trapèzes et triangles. Ensuite le "Edge processor" est un interpolateur qui détermine les sommets des différents segments verticaux composant chaque trapèze. Les cinq "Span processors" génèrent l'ensemble des pixels pour chaque segment vertical.

- Une unité de génération des pixels.

La génération des pixels dans la mémoire de trame est réalisée par 20 processeurs "Image Engine" gérant chacun un vingtième de la mémoire de trame. Ces processeurs éliminent les parties cachées en utilisant l'algorithme du Z-buffer. Ils peuvent aussi faire une combinaison linéaire de la couleur présente avec la nouvelle, ce qui permet de traiter les transparences en profondeur.

- Une unité d'affichage.

Cette unité composée de 5 "multimode graphics processors" affiche les informations fournies par le module de génération via une table de fausses couleurs.

L'éclairage est réalisé en utilisant la méthode d'interpolation de Gouraud.

Cette machine exploite de nombreux parallélismes:

- i) Un pipe-line objets pour les 4 unités du module graphique.
- ii) Un parallélisme pixels, les pixels étant traités en parallèle par les 20 "Image Engines" de l'unité de génération des pixels.
- iii) Il y a donc parallélisation du remplissage des objets et des opérations situées au niveau du pixel (par exemple l'élimination des parties cachées).
- iv) Un parallélisme algorithmique entre la génération des objets et la préparation à la visualisation d'une scène en utilisant les 4 processeurs RISC.

L'intégralité du module graphique est réalisé matériellement, et comprend en particulier 5 circuits intégrés Geometry Engine décrits précédemment. Le bus qui relie les 4 processeurs au module graphique détermine la capacité globale de la machine.

La principale caractéristique de cette machine est d'exploiter au mieux les possibilités de parallélisation qu'offrent le processus de visualisation tout en utilisant un processeur dédié d'une complexité matérielle raisonnable.

Les performances de cette machine sont les suivantes:

- 400000 segments de 10 pixels de long par seconde.
- 137000 triangles de surface moyenne 50 pixels avec ombrage de Gouraud et Z-buffer par seconde.
- 100000 polygones de surface moyenne 100 pixels avec ombrage de Gouraud et Z-buffer par seconde.

Les performances de cette machine reflètent non seulement celles de l'unité de rendu mais aussi celles du calcul des transformations géométriques.

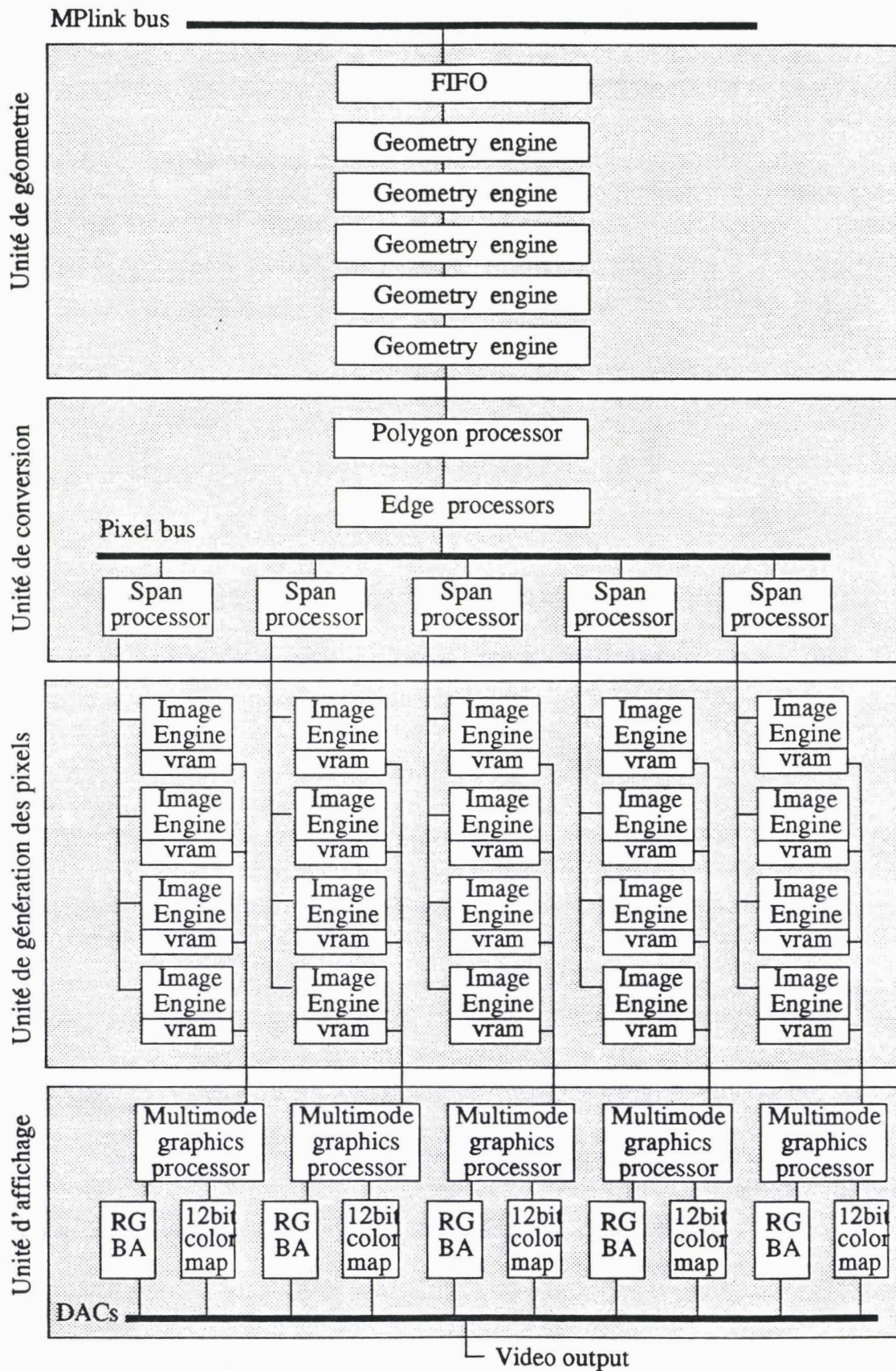


Figure 2.6 : Le module graphique de la Silicon Graphics Workstation

II.1.5 La machine Stellar GS1000

Cette machine [AppaBM89] a pour coeur un réseau de connexions "Data Path" très performant reliant entre elles les différentes unités de la machine.

Les principales unités de cette architecture sont les suivantes (voir Figure 2.7) :

- Un processeur "Multi-Stream Processor" qui exécute simultanément 4 flots indépendants d'instructions. Ce processeur est le module pilote de cette machine.
- Un processeur vectoriel de calcul "Vector Floating Point Processor", relié au réseau de connexions par un bus 512 bits, sur lequel sont exécutées les transformations géométriques. Ce processeur alimente directement par une liaison 32 bits le processeur de rendu.
- Des mémoires de 3 types: une mémoire cache avec une unité TLB "Translation Look-aside Buffer" qui permet aux processeurs de travailler avec des adresses virtuelles, une mémoire principale où est stockée la base de données images et une mémoire vidéo. La mémoire vidéo est directement reliée à l'écran TRC. Elle est organisée en blocs de 16 pixels pour augmenter la vitesse d'accès. Les mémoires principale et vidéo se partagent un bus de 512 bits pour accéder au réseau de connexions.
- Un processeur de rendu "Rendering Processor" qui pour chaque objet graphique génère les pixels le composant. Il réalise en particulier les calculs d'éclaircissement et de profondeur pour éliminer les parties cachées avec l'algorithme du Z-buffer. Les objets visualisables par ce processeur sont les segments, les triangles 3D et les sphères.

Les logiciels développés sur cette machine permettent d'utiliser pour l'éclaircissement les méthodes d'interpolation de Gouraud et de Phong (approximation de Bishop et Weiner) avec réflexions diffuse et spéculaire. Les sources lumineuses peuvent être à l'infini ou ponctuelles à distance finie.

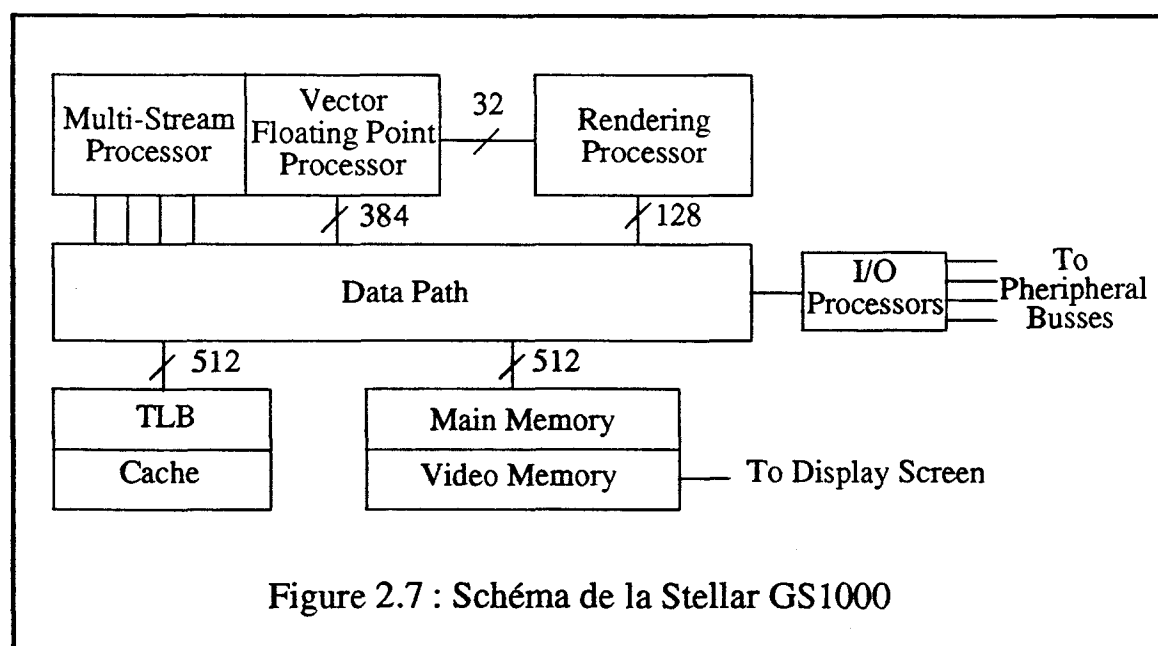


Figure 2.7 : Schéma de la Stellar GS1000

Le processeur de rendu de la machine est composé de trois unités fonctionnant en pipe-line (voir Figure 2.8) :

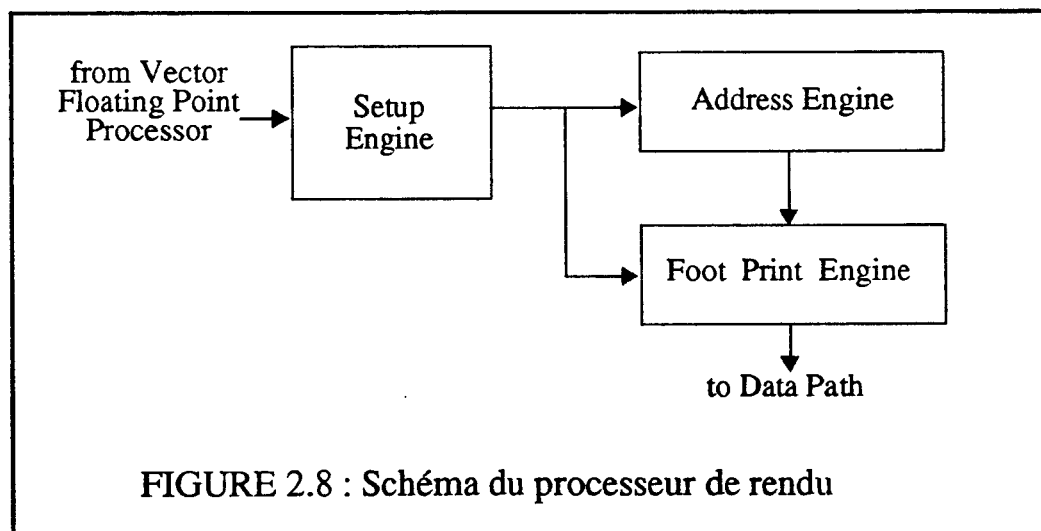
1) Une unité de préparation "Setup Engine" qui détermine les bornes et les coefficients des expressions qui sont utilisés pour faire le rendu. Elle fournit ces valeurs aux deux autres unités.

2) Une unité d'adressage qui détermine les blocs de 4*4 pixels de la mémoire vidéo ayant une intersection avec l'objet en cours de traitement et fournit leur adresse virtuelle à l'unité de génération.

Les deux unités que nous venons de décrire sont réalisées autour d'un circuit intégré microprogrammé "Setup-Address" comprenant, comme éléments principaux, une ALU 32 bits et un multiplieur 16 bits. Ce circuit est un boîtier ayant 299 broches et environ 40000 portes logiques.

3) Une unité de génération "Foot Print Engine" qui traite les 16 pixels d'un bloc en parallèle. Elle est composée de 16 circuits intégrés "Toes" traitant chacun un pixel. Chacun de ces circuits contient une ALU 32 bits et un multiplieur de deux nombres 16 bits. Un accumulateur permet de réaliser des produits 32*32 bits. Ce circuit est un boîtier contenant 25000 portes et ayant 155 broches.

A chaque cycle, cette unité de génération calcule en tout pixel du bloc 4*4 la profondeur et l'attribut à afficher, acquiert via le réseau de connexions le contenu de la mémoire vidéo pour ce bloc, élimine les parties cachées et replace les valeurs dans la mémoire.



Les parallélismes exploités par cette solution architecturale sont les suivants:

- i) Un parallélisme pixels, avec l'utilisation de 16 circuits intégrés "Toes" en parallèle.
- ii) Un pipe line objets de quelques étages entre le processeur vectoriel et les trois unités du processeur de rendu.
- iii) Cette architecture a pour caractéristique de paralléliser toutes les opérations

situées au niveau du pixel, c'est-à-dire pour cette machine les calculs de profondeur et d'éclairage ainsi que l'élimination des parties cachées.

iv) Un parallélisme objets faible en utilisant un processeur multi-flots.

Cette machine utilise dans son module de rendu 18 circuits intégrés: 2 circuits intégrés "Setup-Address" et 16 circuits intégrés "Toes" que nous avons décrits précédemment. C'est cependant toujours cette unité qui est la plus lente de la machine.

La principale caractéristique de cette machine est d'obtenir d'excellentes performances tout en proposant de nombreuses possibilités pour les objets élémentaires directement visualisables (segments, triangles 3D et sphères) et pour les modèles d'éclairage. Ceci est rendu possible par le fait que le processeur de rendu de cette machine est partiellement programmable et que les processeurs travaillant sur chaque pixel peuvent faire des calculs assez complexes, par exemple des multiplications. Par ailleurs, la mémoire vidéo n'est pas liée au processeur de rendu, ce qui permet une utilisation plus souple. Il est par exemple possible d'effectuer un post-traitement sur l'image dans la mémoire de trame.

Les performances (en nombre de triangles par seconde) du logiciel effectuant les calculs de transformations sur le processeur multi-flots et le processeur vectoriel sont les suivantes:

- 143000 triangles de surface moyenne 100 pixels avec ombrage de Gouraud et Z-buffer pour une source lumineuse en utilisant une table de fausses couleurs. (134000 avec deux sources et 120000 avec trois sources)
- 40000 triangles de surface moyenne 100 pixels avec ombrage de Phong et Z-buffer pour une source lumineuse en prenant en compte l'éclairage diffus avec une table de fausses couleurs.
- 25000 triangles de surface moyenne 100 pixels avec ombrage de Phong et Z-buffer pour une source lumineuse en prenant en compte l'éclairage spéculaire et en utilisant une table de fausses couleurs.

Les performances (en nombre de triangles par seconde) du processeur de rendu sont:

- 155000 triangles de surface moyenne 100 pixels avec ombrage de Gouraud et Z-buffer en utilisant une table de fausses couleurs
- 100000 triangles de surface moyenne 100 pixels avec ombrage de Gouraud et Z-buffer en utilisant de vraies couleurs.
- 40000 triangles de surface moyenne 100 pixels avec ombrage de Phong et Z-buffer pour une source lumineuse en utilisant une table de fausses couleurs.

II.1.6 Autres machines commerciales

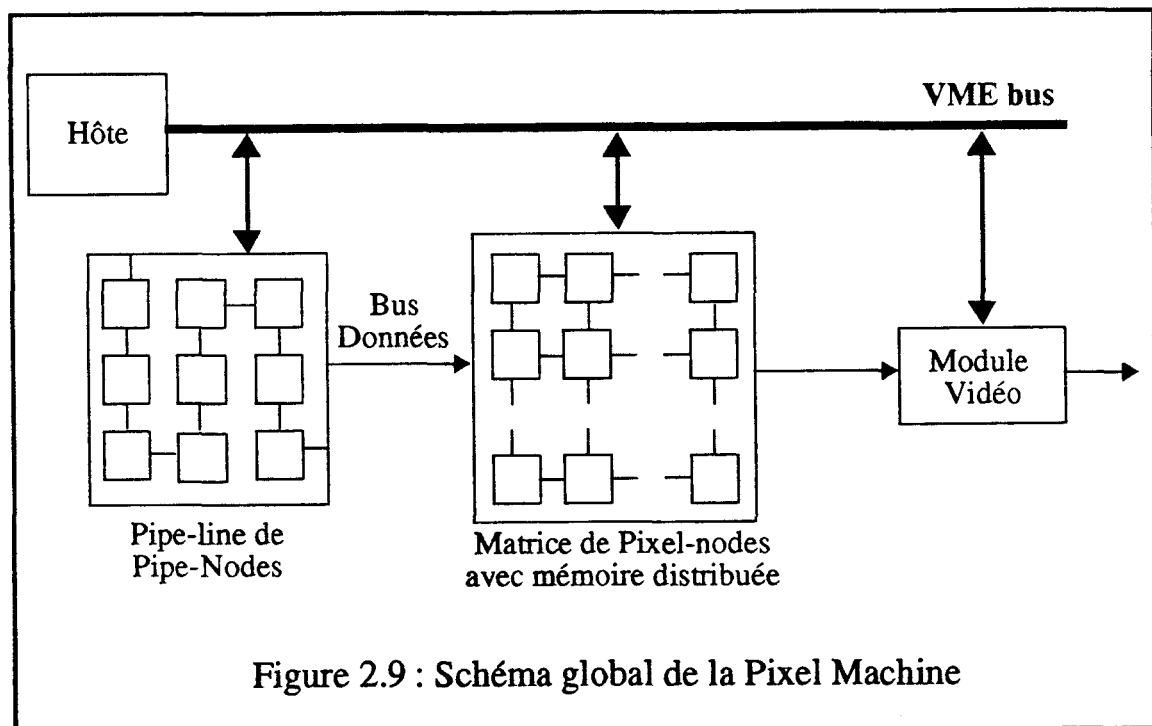
Il existe bien évidemment d'autres machines de haut de gamme ayant des performances du même ordre de grandeur que celles des deux machines présentées. Citons par exemple les SRX de Hewlett Packard, la DN10000 du constructeur Apollo [KirkVo90][Voorhi89] ou Titan de Ardent [Borden89].

Cette dernière a pour particularité de n'utiliser aucun processeur dédié réalisé en VLSI afin d'être le plus souple possible d'utilisation. Elle utilise en parallèle des processeurs

comprenant un CPU RISC et un processeur vectoriel sur lequel sont réalisées en pipe-line les transformations et la conversion des objets en pixels. Le pipe-line doit être réparti sur les différents processeurs en fonction du nombre de processeurs disponibles, ce qui est une tâche difficile à réaliser efficacement. Cette machine a par ailleurs les mêmes défauts que les machines réalisées matériellement: si un étage du pipe-line nécessite beaucoup de temps, il ralentit tout le processus. Pour afficher à des vitesses importantes, un accélérateur graphique, faisant la conversion des objets en pixels, est ajouté à la machine. Cet accélérateur est alors considéré comme l'un des processeurs fonctionnant en parallèle.

II.1.7 La Pixel Machine

Cette machine [PotmHo89][PotmMH89] a pour caractéristique principale de posséder une unité graphique sans aucun processeur spécialisé, mais est composée de processeurs de signaux utilisés en parallèle.



L'unité graphique de la Pixel Machine est composée de trois blocs (voir Figure 2.9) :

- Un (ou deux) pipe-line de 9 "Pipe-Nodes" calculant les transformations sur les objets graphiques (calcul matriciel, découpage dans le polyèdre de vision, projection...).

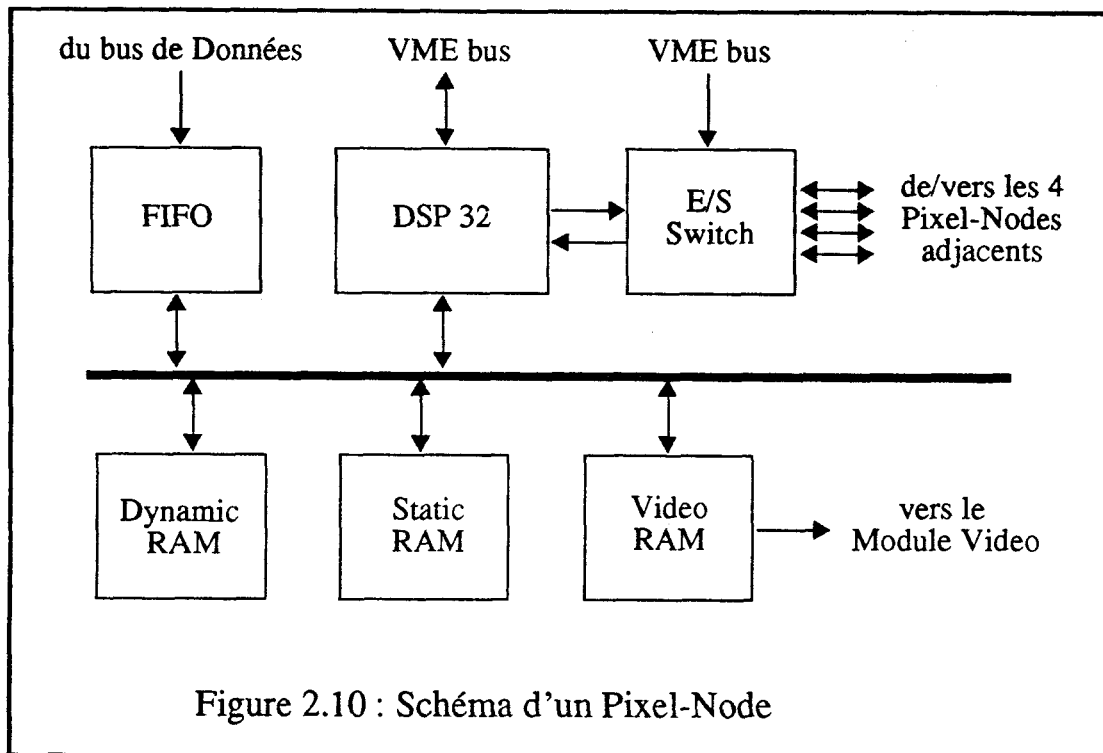
Chaque "Pipe-Node" du pipe-line est composé d'une liste d'attente, d'une mémoire statique et d'un processeur de signal DSP32. Il reçoit les objets graphiques de son prédécesseur dans le pipe-line et les envoie à son successeur après traitement. Le dernier processeur du pipe-line alimente la matrice de "Pixel-Nodes" via le bus de données.

- Une matrice comprenant de 16 à 64 processeurs "Pixel-Nodes" travaillant chacun sur une fraction de la mémoire de trame.

Un "Pixel-Node" a pour composants principaux (voir Figure 2.10): un processeur de signal DSP32 comme le "Pipe-Node", des mémoires statiques et dynamiques, une mémoire vidéo reliée au module vidéo, un contrôleur de synchronisation, une file d'attente pour recevoir les objets graphiques via le bus de données et des primitives d'entrées/sorties lui permettant de communiquer avec les 4 "Pixel-Nodes" adjacents.

- Un module Vidéo composé de deux parties: la première "Pixel Funnel" regroupe et synchronise les pixels distribués dans la mémoire vidéo des différents "Pixel-Nodes", la seconde contrôle l'écran et réalise la conversion digital/analogique.

Le module d'affichage a une capacité de calcul de 820 Mégaflops.



Les parallélismes exploités par cette machine sont faciles à déterminer:

- Un pipe-line objets pour traiter les transformations géométriques de 9 ou 18 étages dans le pipe-line de "Pipe-Nodes"
- Un parallélisme pixels égal au nombre de "Pixel-Nodes" travaillant en parallèle sur la mémoire de trame.
- L'utilisation de plusieurs processeurs de signaux travaillant en parallèle sur la mémoire de trame permet une parallélisation de toutes les opérations situées au niveau du pixel.

Cette machine ne permet probablement pas une visualisation très rapide (malheureusement aucun chiffre de performance n'est donné dans les présentations de cette machine). Mais, étant réalisée avec des processeurs généraux, elle est programmable: tous les algorithmes de visualisation peuvent donc être implantés, en particulier des algorithmes

exploitant le lancer de rayons. Quand elle est utilisée pour faire du rendu géométrique classique elle réalise l'antialiasage par suréchantillonnage.

II.1.8 CAP

Comme celle que nous venons de présenter, la machine CAP (Cellular Array Processor) [SatoIS85] est réalisée autour d'une matrice de processeurs à usage général (voir Figure 2.11). Pour cette machine ce sont des i80186 couplés avec des i8087. Chacun de ces processeurs, appelé cellule (cell), gère une partie de la mémoire de trame. La répartition de la mémoire de trame aux différentes cellules est programmable par bandes, par blocs, ou point par point, afin d'adapter au mieux la machine à l'algorithme utilisé.

Cette matrice de cellules effectue la conversion des objets en pixels. Les objets visualisables par cette machine sont les facettes polygonales.

La préparation d'une image et les transformations géométriques sont réalisées sur le processeur hôte, un Apollo Domain DN640.

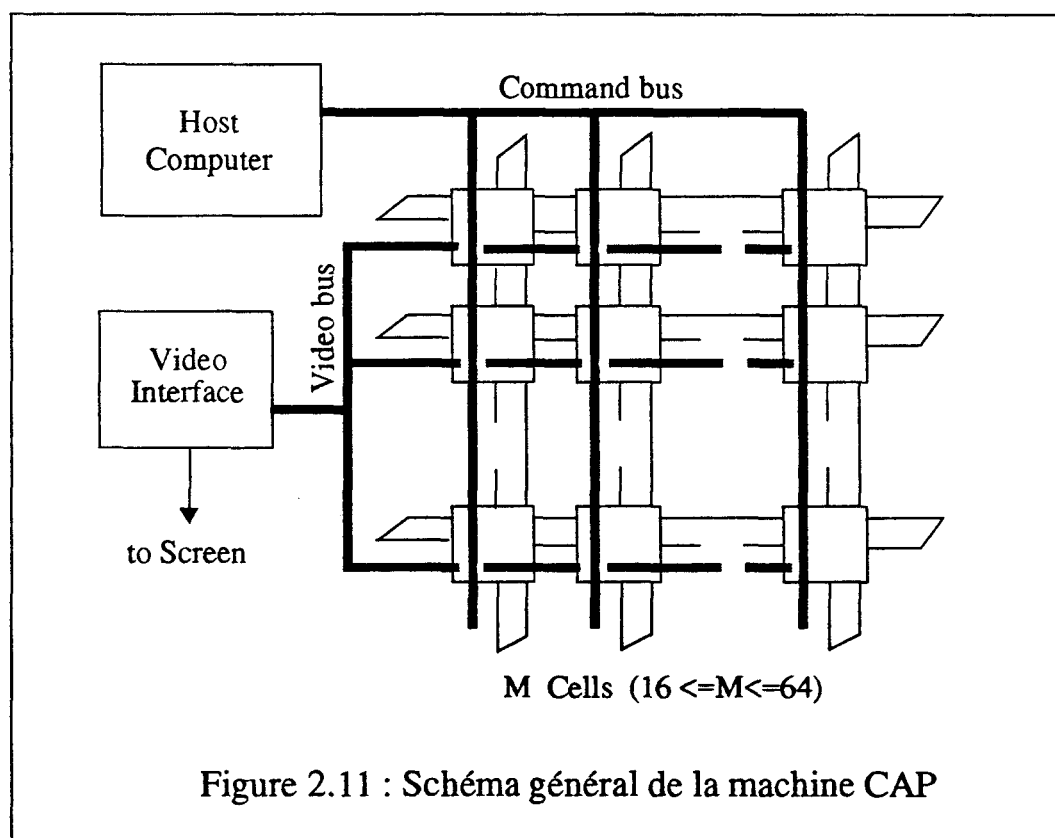


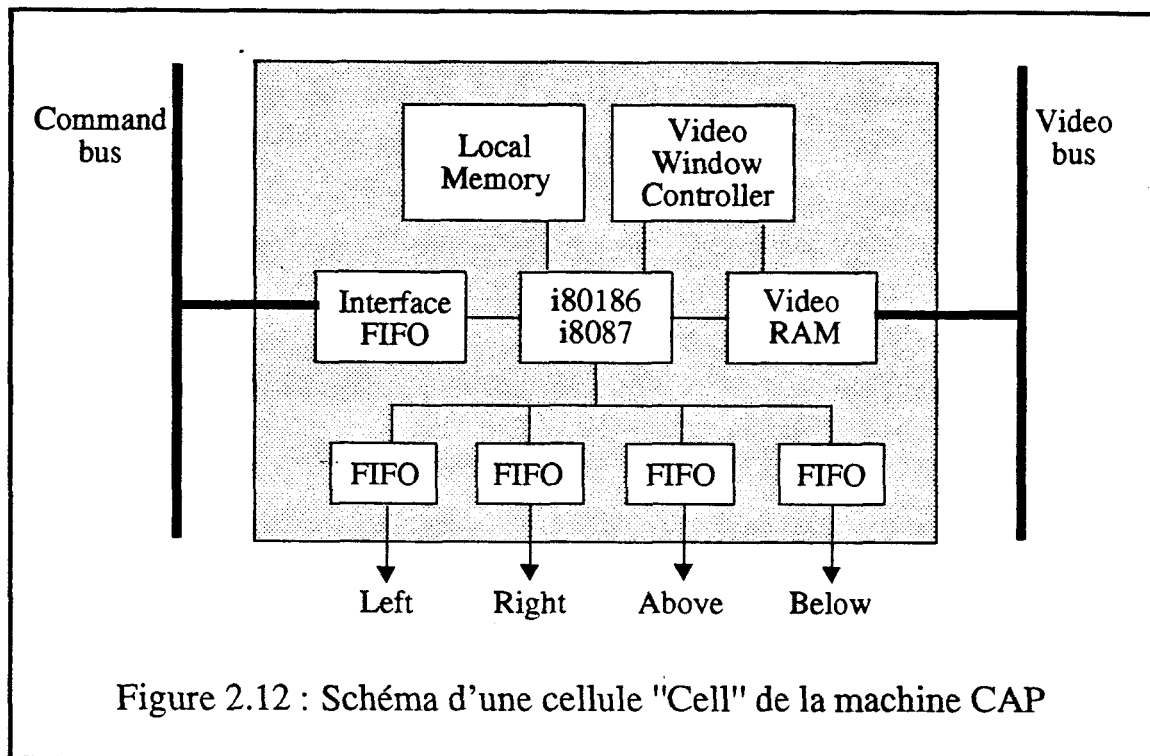
Figure 2.11 : Schéma général de la machine CAP

Chaque processeur "Cell" est composé des unités suivantes (voir Figure 2.12) :

- Une unité centrale comprenant un processeur i80186 et un coprocesseur arithmétique i8087.
- Des mémoires RAM, ROM et Video RAM.
- Une unité de répartition de la mémoire de trame entre cellules "Video Window

Controller".

- Une interface avec le bus de commande.
- Une voie de communication avec les 4 cellules voisines.



Les concepteurs de cette machine ont développé un logiciel implémentant l'algorithme d'Atherton pour visualiser directement un objet défini par un arbre CSG sans précalculer les intersections des objets sur le processeur hôte. Cela permet d'augmenter considérablement la vitesse d'affichage des objets définis par construction CSG.

Le parallélisme utilisé par cette machine est:

un parallélisme pixels entre les pixels traités par des cellules distinctes de la matrice.

Cette machine possède des caractéristiques architecturales assez similaires à la Pixel Machine. Cependant, comme elle est plus ancienne, ses performances sont moindres.

Cette machine met 8 secondes pour afficher une scène comprenant 5320 polygones avec ombrage de Gouraud et Z-buffer dans une configuration de 64 processeurs "Cell". Les performances sont plus faibles en visualisation directe d'objets définis par des arbres CSG.

II.1.9 Experts

La machine Experts (EXpandable Parallel processor Enhancing Real Time Scan conversion) [NiimIM84] s'appuie sur le fait que l'algorithme d'élimination des parties cachées du Z-buffer par ligne est aisément parallélisable: les traitements des différentes lignes de l'écran sont indépendants les uns des autres.

Cette machine est un multiprocesseur exploitant un découpage de l'écran à deux niveaux (voir Figure 2.13):

- Un ensemble de M processeurs "SLP" (Scan Line Processor) traitant chacun une bande horizontale de l'écran. Chaque processeur décompose les objets graphiques fournis par le hôte en segments horizontaux pour les lignes écran dont il s'occupe. Pour chaque ligne il établit une liste de segments.

- Un ensemble de N processeurs "PXP" (PiXel Processor) associé à chaque processeur "SLP" traitant chacun une partie d'une ligne écran. Les différentes lignes écran traitées par un même "SLP" le sont séquentiellement. Ces processeurs exécutent l'élimination des parties cachées, effectuent les calculs d'ombrage suivant la méthode de Gouraud et effectuent un antialiasage aux extrémités des segments.

Les nombres N et M de processeurs sont bornés par 16, c'est-à-dire au maximum 16 processeurs "SLP" et 16*16 processeurs "PXP".

M unités "Merger" fusionnent les résultats des différents "PXP" liés à un "SLP" et place les résultats dans la mémoire de trame. Celle-ci est lue par l'unité vidéo "Video Output Control Unit".

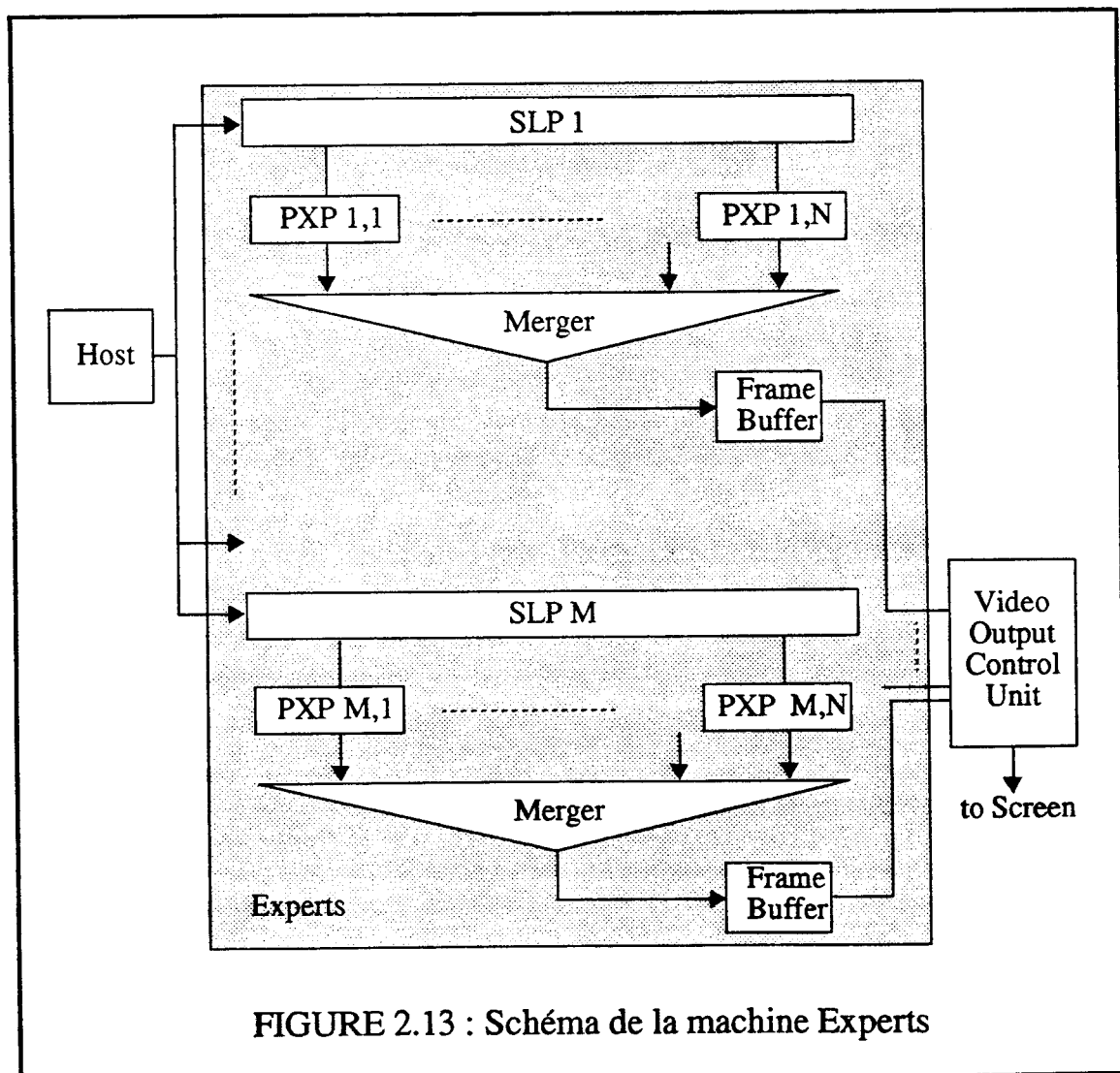


FIGURE 2.13 : Schéma de la machine Experts

Cette machine exploite les parallélismes suivants:

i) Un parallélisme objets entre les polygones traités par des processeurs "SLP" distincts.

ii) Un parallélisme objets de plus bas niveau entre les segments horizontaux traités par des processeurs "PXP" distincts d'un même processeur "SLP".

iii) Un parallélisme pixels. En effet les pixels sont générés en parallèle par N*M processeurs "PXP".

Il faut, cependant, bien cerner les deux restrictions suivantes:

- Aucun effet pipe-line n'est utilisable, les processeurs "PXP" ne pouvant traiter une ligne écran que lorsque tous les objets de la scène ont été traités par le processeur "SLP" dont ils sont esclaves.

- Chaque processeur "SLP" doit considérer tous les objets graphiques, même ceux absents de la partie de l'écran dont il s'occupe pour créer les listes de segments pour chaque ligne. De même, les processeurs "PXP" doivent considérer tous les segments présents sur la ligne.

Ces deux limites restreignent considérablement les possibilités de travail en parallèle.

Les processeurs "SLP" et "PXP" sont des circuits réalisés spécifiquement pour cette machine. Ils sont microprogrammables.

Cette machine a des performances modestes, vu les structures matérielles développées. Ceci est en partie dû au fait que cette proposition a quelques années; avec les mêmes concepts, on pourrait envisager aujourd'hui de meilleures performances.

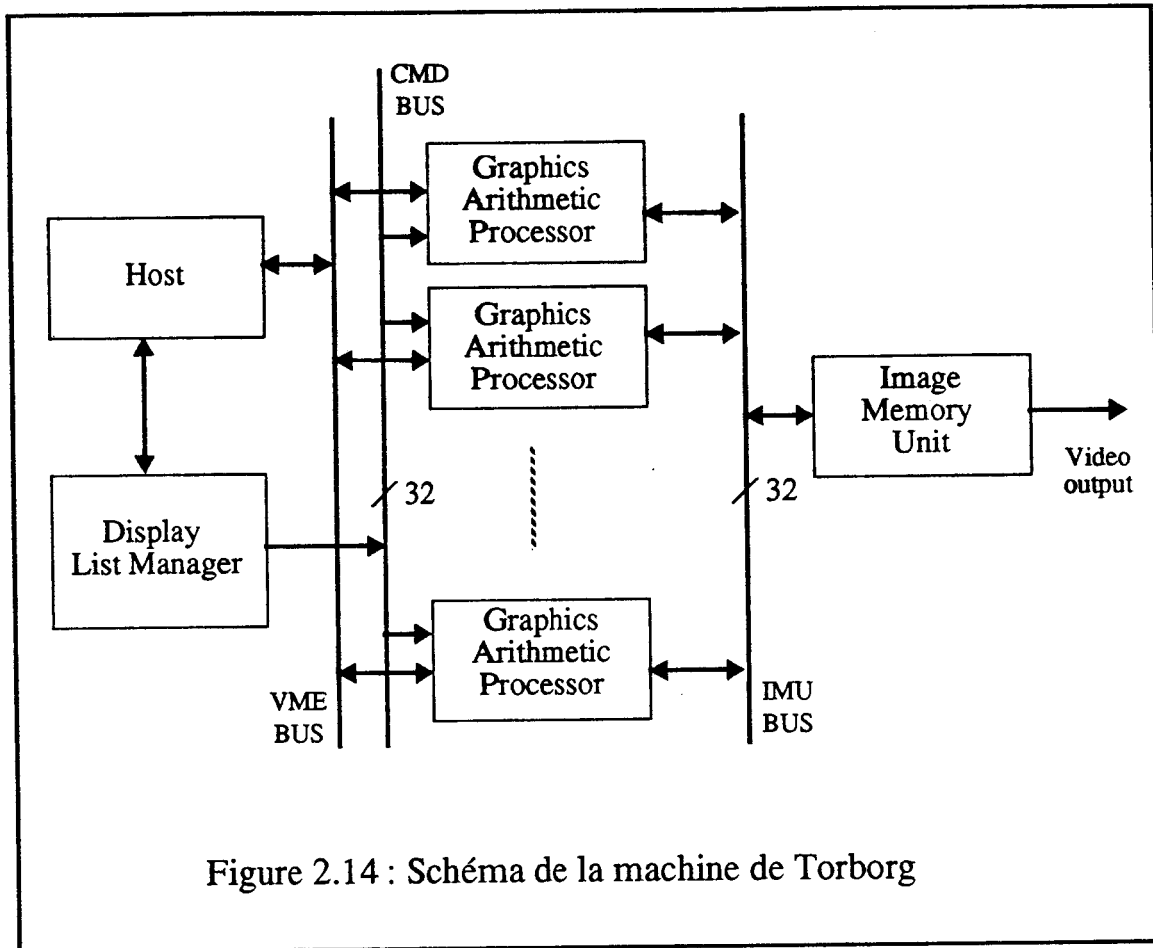
Cette machine dans une configuration comprenant 8 processeurs "SLP" et 8*8 processeurs "PXP" peut afficher environ 200 polygones avec ombrage de Gouraud et Z-buffer en un quinzième de seconde.

II.1.10 La Machine de Torborg

[DenaRT88][Torbor87]

L'unité graphique de cette machine est composée (voir Figure 2.14) d'une structure de donnée "DLM" (Display List Manager) de plusieurs processeurs objets "GAP" (Graphics Arithmetic Processor) travaillant en parallèle (au maximum 8) et d'une unité de mémoire de trame "IMU" (Image Memory Unit). L'unité "DLM" contient les outils permettant la répartition des objets aux processeurs "GAP".

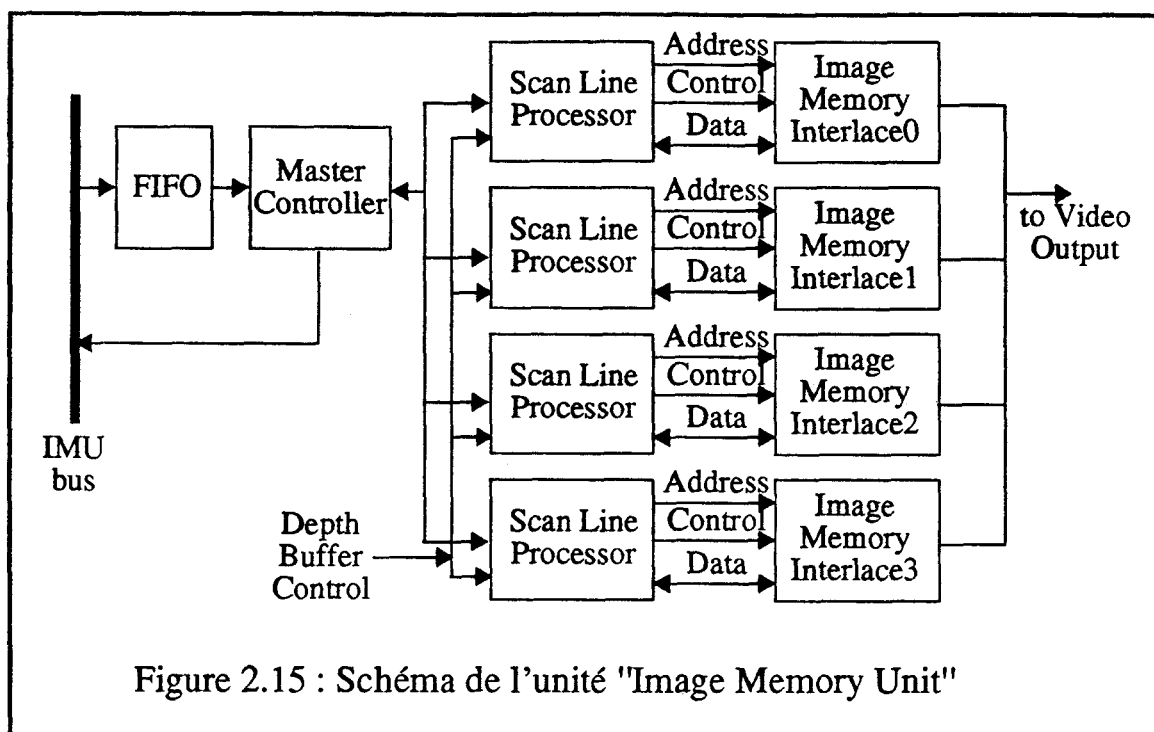
Les processeurs objets sont de puissants processeurs directement contrôlés par l'application via le "VME bus". A partir des objets de la structure de données à laquelle ils sont reliés via le "CMD bus", les processeurs objets produisent des objets de bas niveau (points, segments, rectangles, triangles...). Un second bus "IMU bus" relie les processeurs objets à l'unité de mémoire d'image.



L'unité "IMU" traite des triangles 3D par interpolations linéaires à partir des valeurs X, Y, Z, R, V et B aux trois sommets du triangle. Elle est composée d'un contrôleur "Master Controller" qui prétraite les objets et de quatre processeurs "Scan Line Processor" gérant chacun un quart des lignes (voir Figure 2.15).

Le "Master Controller" détermine les extrémités des différents segments composant chaque triangle. Les unités "Scan Line Processor" génèrent les valeurs Z, R, V et B en tous les pixels de chacun des segments et effectuent l'élimination des parties cachées par l'algorithme du Z-buffer.

Le "Master Controller" et les "Scan Line Processor" sont des circuits intégrés VLSI. Le "Master Controller" contient environ 8700 portes et le "Scan Line Processor" 15000 portes. Les principaux éléments de ces circuits intégrés sont des mémoires, des interpolateurs et des générateurs d'adresses.



L'unité de mémoire d'image est contrôlée par l'ordinateur hôte via les processeurs objets. Une commande séquentielle graphique n'est exécutable que lorsque les différents processeurs objets ont terminé le traitement en cours. Cette machine ne peut donc être utilisée pour afficher une image en un vingt-cinquième de seconde.

Les parallélismes utilisés sont les suivants:

i) Un parallélisme objets entre les objets traités par des processeurs "GAP" différents pour les traitements géométriques.

ii) Un parallélisme objets de bas niveau entre les segments d'un même triangle traités par des "Scan Line Processors" distincts.

iii) Un parallélisme pixels dans l'unité "IMU". Cette unité génère les pixels dans la mémoire de trame au rythme de 20 à chaque cycle.

Seul le processeur "IMU" est réalisé avec des composants VLSI. Ce processeur permet de générer 60 millions de pixels par seconde.

Indépendamment de la vitesse de l'unité "IMU", la possibilité de multiplier le nombre de processeurs "GAP" est limité par la capacité du "IMU bus".

Une des principales originalités de cette machine est de proposer une architecture parallélisant la partie transformation du processus de visualisation. Un second point fort est son unité "Image Memory Unit" très performante. La stratégie utilisée pour la répartition des objets aux différents processeurs "GAP" n'est pas évoquée.

Une scène de 65000 triangles de taille moyenne 19 pixels est affichée par cette machine en 720 ms avec ombrage de Gouraud et Z-buffer en utilisant des éclaircissements diffus et spéculaires et 8 processeurs "Graphics Arithmetic Processor". Le temps pour la même scène en utilisant un seul processeur "GAP" est de 5,7 s.

II.1.11 GSP-NVS

La machine GSP-NVS (Graphical Signal Processor with Normal Vector Shading) [DeerWS88] propose une structure originale pour le traitement de triangles 3D obtenus par découpage des polygones de la structure de données. Cette machine reprend la structure orientée objets présentée pour la première fois par Weinberg [Weinbe81].

Son organisation est la suivante (voir Figure 2.16):

L'unité "Display List Runner" recueille dans la mémoire de l'ordinateur hôte les informations géométriques et les commandes d'affichage. Elle effectue certains calculs de préparation, en particulier le découpage en triangles.

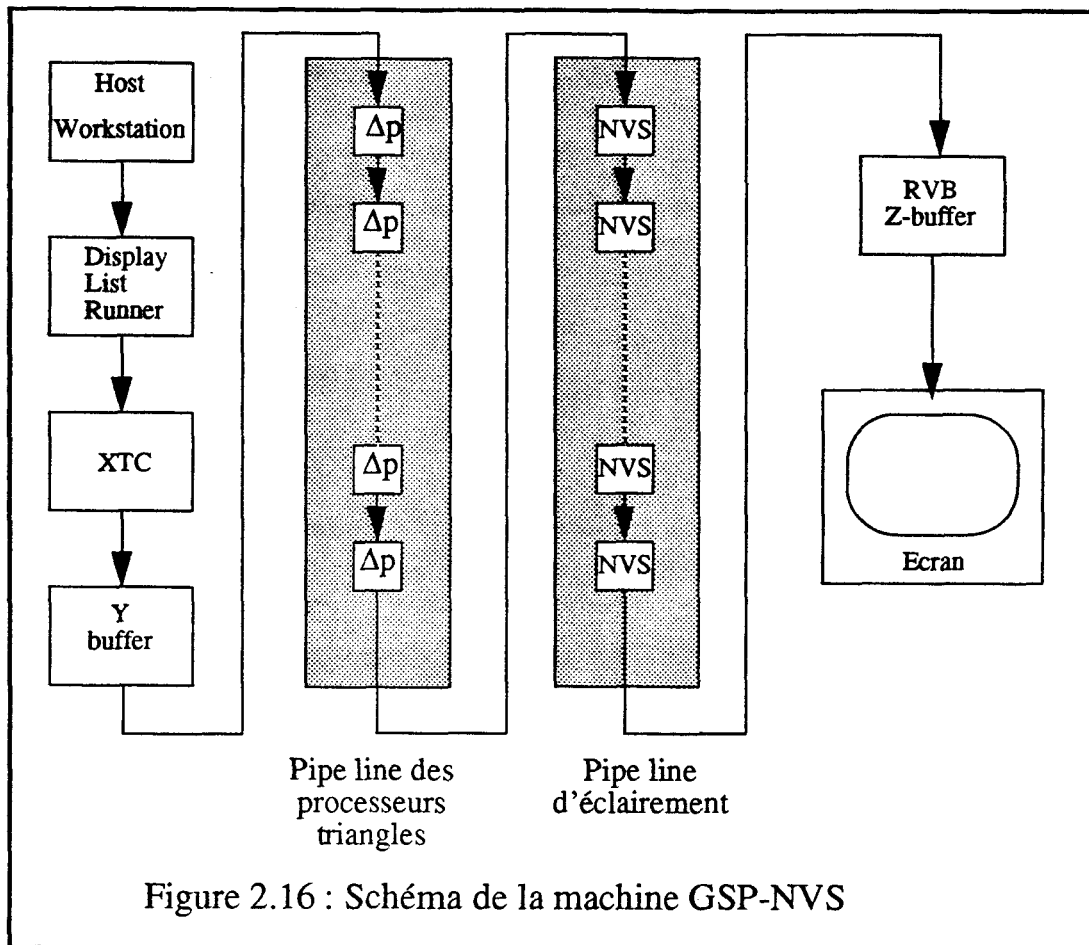
L'unité "XTC" effectue les transformations géométriques et le découpage dans le polyèdre de vision.

Le processeur "Y buffer" effectue un tri des triangles, en 1024 listes, suivant leur point d'ordonnée maximale pour les fournir aux différents processeurs " Δp " du pipe-line au début du traitement de la ligne-écran où ils apparaissent.

Ensuite un ensemble de processeurs " Δp " fonctionnant en pipe-line détermine en tout pixel et le triangle visible et la normale de celui-ci en ce pixel. Chaque processeur " Δp ", réalisé en VLSI, s'occupe d'un et un seul triangle. A partir des valeurs de profondeur et de normale aux trois sommets fournies par le "Y buffer", le processeur effectue des interpolations bilinéaires. En tout pixel, il teste si son triangle est visible en fonction des valeurs qu'il reçoit de son prédécesseur dans le pipe-line. Si son triangle est devant il envoie ses propres valeurs sinon il fait suivre les valeurs reçues. L'alimentation des processeurs en triangles se fait avant le traitement de chaque ligne de la mémoire de trames. Un processeur peut ainsi traiter plusieurs triangles horizontalement disjoints d'une même image.

Le second pipe-line de 16 processeurs "NVS" calcule l'ombrage avec la méthode de Phong pour 5 sources lumineuses en fonction de la normale en chaque pixel. Ces processeurs normalisent le vecteur normal N et le vecteur de position de l'observateur V puis calculent le vecteur de réflexion R avec l'expression $R=2(N.V).N-V$. Ils calculent ensuite les valeurs R , V , et B pour chaque source lumineuse en fonction de R , N , L_i la direction de la source, L_{ci} sa couleur, L_{ai} la lumière ambiante et les caractéristiques intrinsèques de l'objet. Le circuit intégré "NVS" est un processeur spécialisé pour calculer des racines carrées et des puissances en utilisant des ROM spécifiques.

Les deux pipe-lines de cette machine fonctionnent au rythme du balayage de la mémoire de trame. Si sur une ligne donnée il y a plus de triangles que de processeurs, le traitement de cette ligne s'effectue en plusieurs passes. Un post-traitement est alors nécessaire pour éliminer les parties cachées, il est effectué par le processeur "RVB Z-buffer". Tout espoir d'obtenir une animation en temps réel est perdu.



Chaque processeur " Δp " est composé des unités suivantes (voir Figure 2.17) :

- L'unité "X-Unit" détermine par interpolations pour chaque ligne écran les deux extrémités X_g et X_d du segment représentant le triangle sur cette ligne. Cette unité gère aussi le balayage de la ligne écran et effectue la comparaison entre le pixel courant et les valeurs X_g et X_d . Elle peut ainsi déterminer en tout pixel si le triangle dont elle s'occupe est présent.
- Les unités "Z-Unit", "Nx-Unit", "Ny-Unit" et "Nz-Unit" calculent les valeurs de profondeur et de normale en tout pixel où le triangle est présent par interpolations linéaires. "Z-Unit" compare ensuite sa valeur de profondeur avec celle que lui envoie le " Δp " précédent dans le pipe-line. Si son triangle est devant pour un pixel donné, le processeur " Δp " envoie ses valeurs de profondeur et de normale sinon il fait suivre les valeurs reçues de son prédécesseur.
- L'unité "M-Unit" sert à stocker les paramètres propres à l'objet: couleur, coefficients de réflexion,...
- L'unité "Control Unit" interprète les commandes qui séquent le chargement des triangles dans les processeurs et contrôle le traitement des lignes écran.
- L'unité "G-Unit" génère les signaux activant le balayage ligne.

Le circuit intégré a 30 broches d'entrée; les 60 signaux sont acquis grâce à un multiplexage temporel.

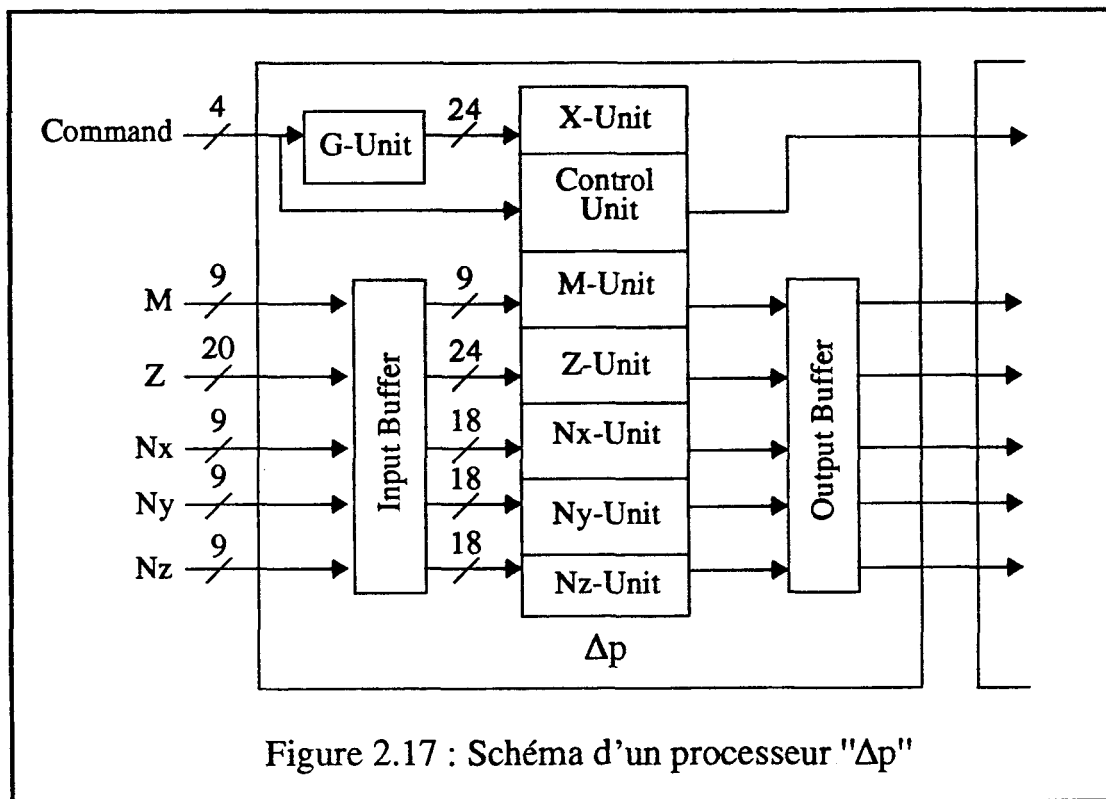


Figure 2.17 : Schéma d'un processeur "Δp"

Les parallélismes utilisés par cette machine sont les suivants:

i) Un pipe-line pixels pour l'élimination des parties cachées (les processeurs "Δp") et pour les calculs d'éclairage (les 16 "NVS")

ii) Un parallélisme objets; les processeurs "Δp" traitant en parallèle les différents triangles.

iii) Il y a donc une parallélisation importante de la génération des pixels. En tenant compte de l'effet pipe-line, s'il y a autant de processeurs que de triangles pour toutes les lignes, le temps de génération d'un pixel est de 1 cycle.

Cette machine comprend une grande partie matérielle, les processeurs "Δp" et "NVS" sont réalisés en CMOS VLSI. Un processeur "Δp" intègre 25000 transistors et possède 68 broches. Le circuit intégré "NVS" a lui aussi 68 broches avec la même répartition. Les concepteurs de ce prototype envisagent d'intégrer 100 processeurs "Δp" par dm^2 . L'ensemble des 16 circuits intégrés "NVS" a une capacité de calcul de 2 milliards de multiplications par seconde.

Les unités "XTC" et "Y buffer" doivent avoir des capacités de calcul considérables (1 Gigaflops) pour alimenter au maximum de leurs possibilités les deux pipe-lines effectuant le rendu. Cela implique d'avoir une liaison entre eux ayant un débit considérable.

Les principales originalités de cette proposition sont les suivantes:

- L'utilisation d'une très imposante quantité de circuits intégrés VLSI.
- La bonne qualité de l'éclairage, méthode de Phong avec cinq sources lumineuses.
- L'exploitation d'un parallélisme massif objets.

Les concepteurs de cette machine estiment qu'elle pourra afficher une scène comprenant 25.000 petits triangles dont 12.000 visibles en un vingtième de seconde sur un écran 1280*1024 en utilisant 1.000 processeurs "Δp". Ils espèrent, en augmentant le nombre de processeurs "Δp", arriver à des performances de 1.000.000 de triangles par seconde.

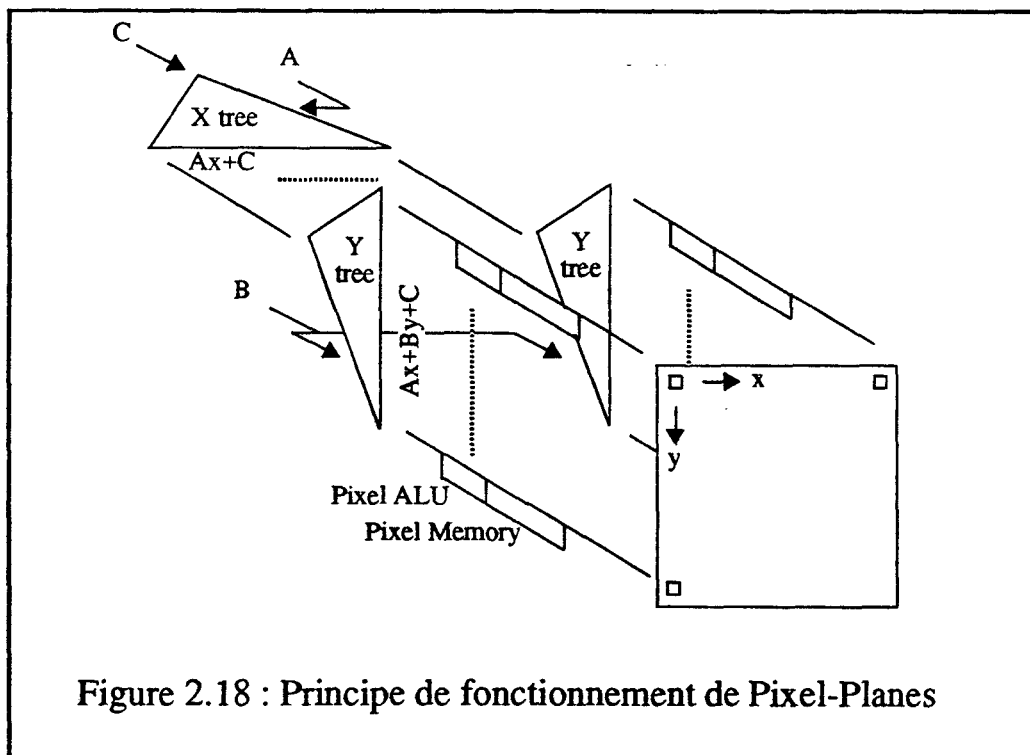
La machine PROOF étudiée à l'Université de Tubingen utilise la même approche en exploitant un suréchantillonnage pour faire de l'anti-aliassage [Schnei88][Schnei90] [Clauss89][Clauss90][SchnCl89].

II.1.12 Pixel-planes 4

[FuchPo81][FuchGH85][EyleAF88][FuchPE89a]

Pixel Planes 4 est une machine SIMD un processeur VLSI par pixel, où toute la visualisation se fait par évaluation d'expressions linéaires de la forme $V(x,y)=Ax+By+C$ (x,y) étant les coordonnées d'un pixel et A, B, et C des constantes pour une image donnée.

Le calcul et la diffusion aux processeurs-pixel des valeurs $V(x,y)$ se fait par des arbres d'additionneurs. Chaque processeur-pixel est composé d'une ALU un bit et de 72 bits de mémoire (voir Figure 2.18).



L'unité d'évaluation des expressions et de mémoire de trame est réalisée avec 2048 circuits intégrés pour afficher sur un écran 512*512. Chacun de ces circuits intégrés comprend 64 processeurs-pixel pour couvrir un pavé 8*8 pixels de l'écran et le morceau d'arbre de diffusion correspondant (voir Figure 2.19).

Les objets traités par cette unité sont des polygones convexes en 3D. Pour un polygone

donné, chaque processeur-pixel détermine s'il appartient à la projection écran de cet objet par évaluation des équations des droites frontières. Cela correspond à définir un polygone comme une intersection de demi-plans et donc impose aux polygones d'être convexes. Chaque cellule effectue l'élimination des parties cachées en son pixel en utilisant l'algorithme du Z-buffer. Pour cela une des expressions traitées est l'équation du plan dans l'espace contenant le polygone en cours de visualisation.

L'utilisation d'expressions linéaires permet, pour le codage des couleurs, d'appliquer la méthode d'interpolation de Gouraud.

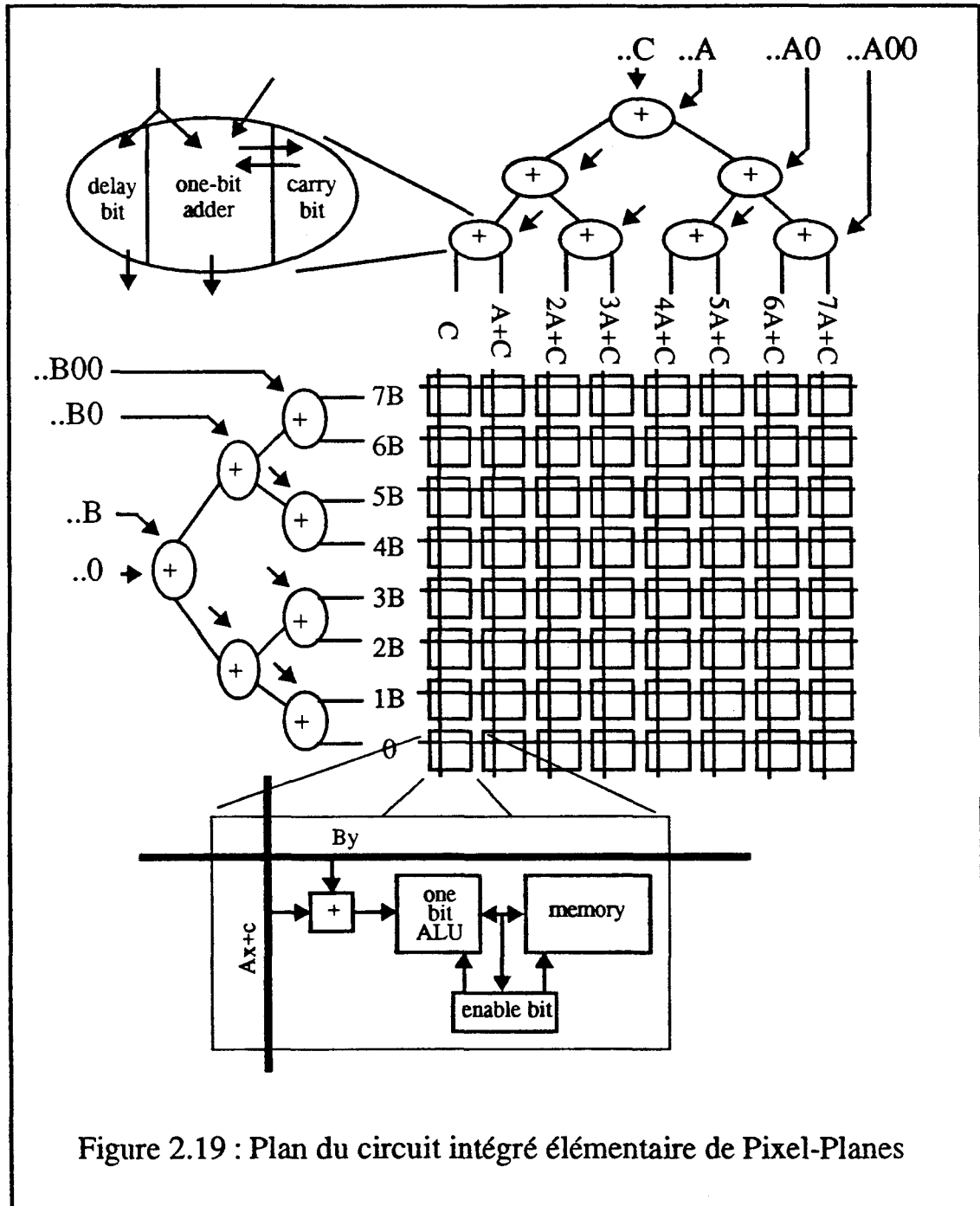


Figure 2.19 : Plan du circuit intégré élémentaire de Pixel-Planes

Décrivons le schéma global de la machine alimentant l'unité de visualisation et de mémoire de trame "Enhanced Memory Array" que nous venons de présenter.

Ses unités principales sont les suivantes (voir Figure 2.20):

- Une station de travail graphique MicroVax II faisant office de processeur hôte sur lequel sont définis et modifiés les objets. Cette unité établit aussi pour chaque scène la liste des objets à visualiser.
- Un processeur graphique "Graphics Processor" commandé par le Hôte. Il acquiert les polygones à visualiser et effectue toutes les transformations (changement de repère, découpage dans le polyèdre de vision et projection en 2D). Il fait aussi les calculs des éclaircissements aux sommets des polygones et termine par la transformation des objets en expressions linéaires.
- L'unité "Image Generation Controller" qui envoie à l'unité de rendu et de mémoire de trame les valeurs A, B, C, les instructions et les adresses. C'est elle qui contrôle l'activité de l'unité de rendu.
- Le "Video Controller" qui récupère dans les mémoires des différents processeurs-pixels les valeurs à afficher.

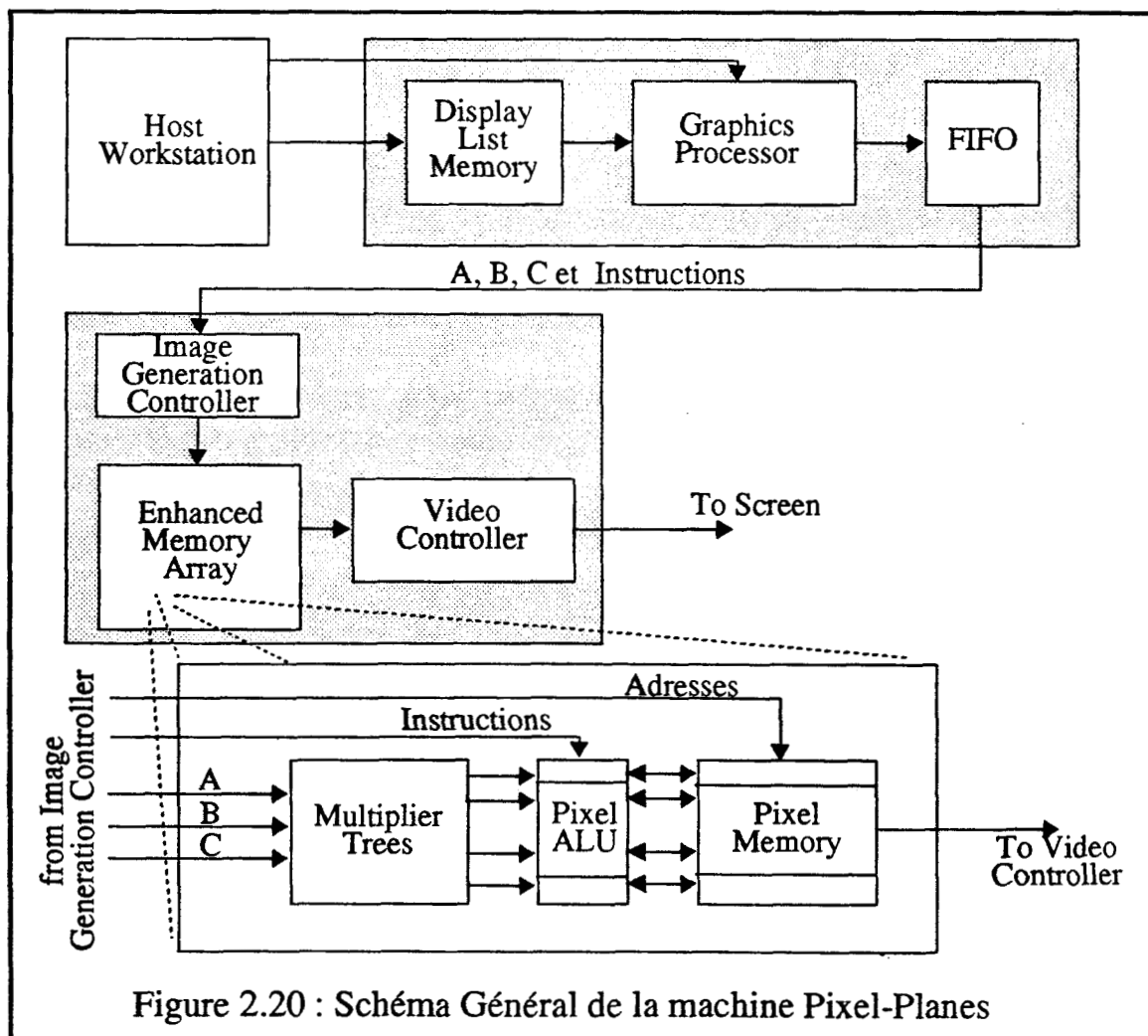


Figure 2.20 : Schéma Général de la machine Pixel-Planes

Les parallélismes exploités sont les suivants:

i) Un pipe-line objets entre les différentes expressions linéaires dans les arbres d'additionneurs qui alimentent les processeurs-pixels.

ii) Un parallélisme pixels massif, tous les processeurs-pixel traitant la même expression linéaire simultanément.

iii) Cette structure permet une parallélisation remarquable de toutes les opérations effectuées au niveau pixel. Malheureusement les opérations effectuables à ce niveau sont très limitées. Il n'y a que 72 bits de mémoire et une ALU un bit en chaque pixel.

La visualisation d'une scène se fait par l'envoi d'un flôt de commandes (polygone par polygone) dans les arbres d'additionneurs. Le temps pour afficher une scène est donc fonction du nombre d'expressions permettant de la visualiser.

Les ombres portées peuvent être obtenues par post-traitement en considérant les volumes d'ombre comme des objets à part entière en utilisant l'algorithme présenté par Crow [Crow77]. Le temps de traitement est alors multiplié par le nombre de sources lumineuses traitées.

Un algorithme permettant de visualiser directement des objets définis par un arbre CSG a été développé pour cette machine. Ce logiciel est d'un grand intérêt car le processeur hôte n'a plus à calculer les intersections entre toutes les objets primitifs des arbres CSG. Nous expliciterons cet algorithme dans le chapitre suivant.

Les performances de Pixel-Planes 4 sont les suivantes:

- 39.000 triangles de taille quelconque par seconde avec ombrage de Gouraud et Z-buffer pour un écran 512*512. (20% de moins pour des quadrilatères)
- 11.000 triangles de taille quelconque par seconde avec ombrage de Gouraud et Z-buffer avec application des ombres portées.
- 13.000 sphères s'interpénétrant par seconde avec ombrage.

D'après ses concepteurs, cette proposition a les limites suivantes:

- i) Le matériel développé est sous-utilisé vu la structure SIMD choisie.
- ii) la mémoire en chaque pixel est insuffisante
- iii) La mémoire de trame n'est pas accessible directement sans passer par les arbres d'additionneurs.
- iv) L'unique processeur graphique alimentant l'unité "Enhanced Memory Array" n'est pas assez performant

Pour combler ces lacunes, l'équipe de Fuchs réalise une nouvelle génération de cette machine Pixel-Planes 5, que nous allons présenter maintenant.

II.1.13 Pixel-planes 5

[GoldFu86][GoldHF86][GoldMT89][FuchPE89b]

La principale nouveauté sur Pixel-Planes 5 est de ne plus utiliser une structure SIMD avec des arbres d'additionneurs pour tout l'écran mais pour une zone écran de 128*128 pixels. Un processeur s'occupant d'une zone de 128*128 pixels est appelé "Renderer". Plusieurs de ces processeurs sont utilisés en parallèle MIMD pour valoriser l'intégralité de la mémoire de trame.

Un processeur "Renderer" (voir Figure 2.21) est réalisé à l'aide de 64 circuits intégrés VLSI comprenant chacun deux colonnes de 128 processeurs-pixels et les arbres d'additionneurs permettant de les alimenter. Les arbres de diffusion de Pixel-Planes 5 (Quadratic Expression Evaluator) fournissent aux 128*128 processeurs-pixels une expressions du second degré. La mémoire de chaque processeur-pixel est de 208 bits. Une mémoire tampon "Backing Store" de 4K bits par pixel est jointe à la matrice de 128*128 processeurs-pixels. Une unité "Image Generation Controller" contrôle les processeurs-pixels et la mémoire tampon.

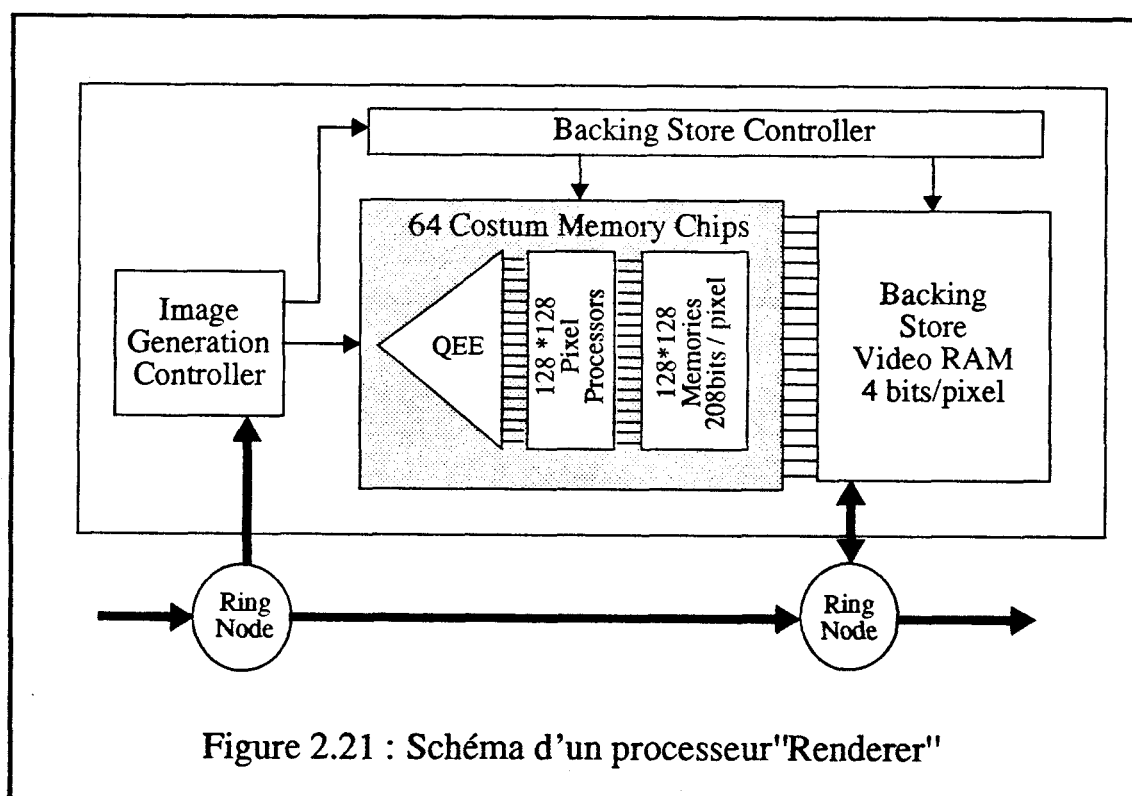


Figure 2.21 : Schéma d'un processeur "Renderer"

La machine Pixel-Planes 5 a pour cœur un anneau de communication "Ring" de 8 voies indépendantes de 32 bits parallèle fonctionnant à 20 Mhz. Les processeurs "Renderer" (8 ou 10) décrits précédemment sont connectés à cet anneau.

Les autres unités de Pixel-Planes 5 sont (voir Figure 2.22):

- La mémoire de trame et le module vidéo.
- L'ordinateur hôte (une station SUN 4).

- 32 processeurs "Graphics Processor" qui traitent les transformations géométriques et le codage des objets en expressions linéaires ou quadratiques. Les "Graphics Processors" sont des i860 du constructeur Intel [GrimKB89]. Ces processeurs 64 bits ont des performances très importantes, de l'ordre de 500.000 opérations flottantes par seconde.

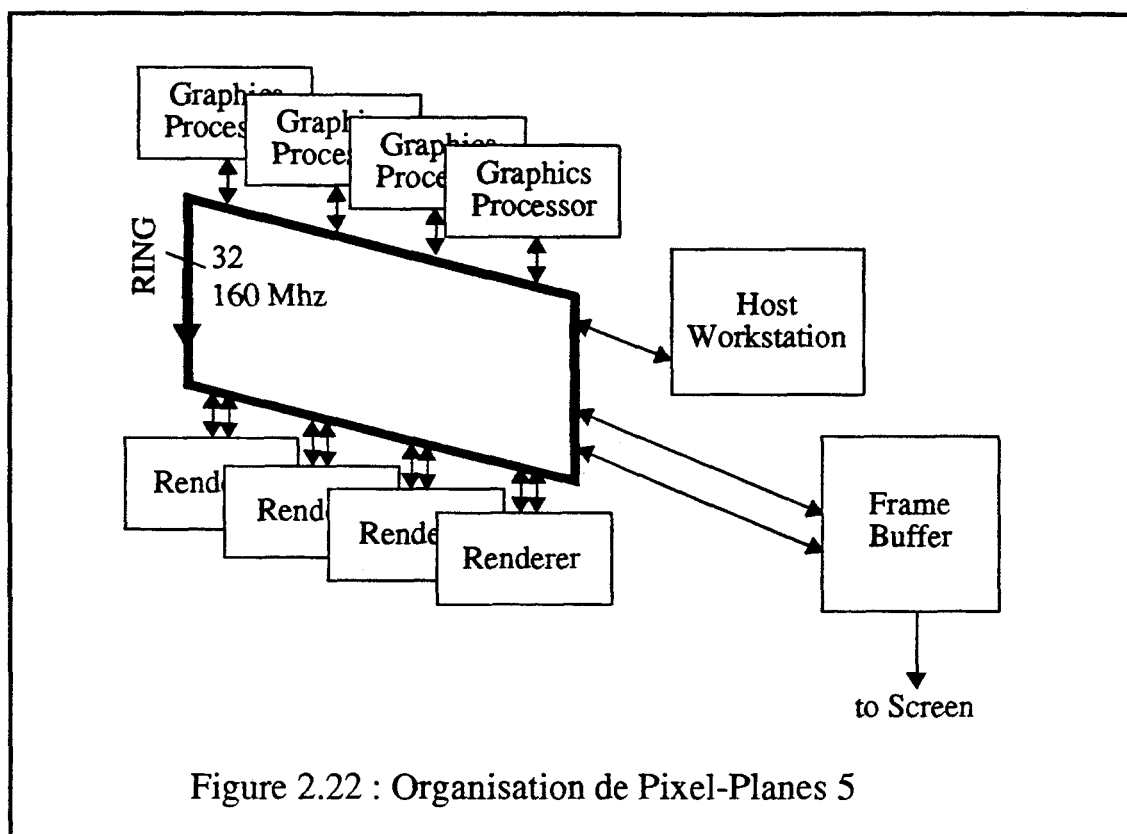
La listes des objets graphiques est répartie sur les différents "Graphics Processor". Quand des objets sont ajoutés ils sont répartis sur les processeurs. Un des processeurs appelé "Master Graphics Processor" contrôle les autres. Il envoie les commandes aux "Graphics Processors" et synchronise les échanges avec les processeurs "Renderer".

C'est aussi lui qui contrôle les processeurs "Renderer" et qui leur distribue les 80 zones de la mémoire de trame qu'ils doivent traiter. Cette gestion se fait dynamiquement en fonction de la répartition des objets graphiques sur l'écran.

Décrivons maintenant les différentes phases du processus d'affichage:

- 1) Le processeur hôte fournit les objets graphiques aux "Graphics Processors".
- 2) L'application demande au "Master Graphics Processor" de calculer une nouvelle image, signal qu'il répercute aux autres processeurs.
- 3) Les "Graphics Processors" effectuent les transformations et calculent les expressions linéaires ou quadratiques et les instructions correspondantes pour chaque objet. Ils placent celles-ci dans une boîte ou plusieurs si l'objet est à cheval sur plusieurs zones. Chaque boîte correspond à une zone de 128*128 pixels de la mémoire de trame (un écran 1280*1024 nécessite 80 boîtes).
- 4) Chaque "Graphics Processor" envoie en utilisant l'anneau le contenu d'une boîte à l'unité "Renderer" traitant la zone correspondante de l'écran. Chaque "Renderer" traite les informations reçues d'un "Graphics Processor" et stocke les résultats dans la mémoire tampon.
- 5) Quand un processeur "Renderer" reçoit les informations du dernier "Graphics Processor", il détermine la valeur finale en tout pixel de la zone qu'il traite.
- 6) Les unités "Renderer" envoient les valeurs finales pour une zone à la mémoire de trame.
- 7) Quand les 80 zones sont traitées, la mémoire de trame est affichée sur l'écran video. Deux mémoires en bascule sont utilisées.

La méthode d'ombrage de Phong peut être employée, ce sont les processeurs "Renderer" qui effectuent les calculs en tout pixel. Un processeur-pixel met 23000 cycles ou 0,57 ms pour calculer l'ombrage de Phong en un pixel pour une source lumineuse.



Cette machine exploite les parallélismes suivants:

- i) Un parallélisme objets pour les transformations géométriques entre les objets traités par des processeurs "Graphics Processors" distincts.
- ii) Un parallélisme objets pour la conversion en pixels entre objets appartenant à deux zones distinctes de la mémoire de trame traitées en parallèle par deux "Renderer".
- iii) Un parallélisme pixels entre les différents pixels d'une même zone 128*128 de la mémoire de trame et entre pixels appartenant à des zones distinctes mais traitées en parallèle par plusieurs "Renderer".
- iv) Un pipe line objets entre expressions linéaires quadratiques dans les arbres d'additionneurs alimentant les processeurs-pixels d'un "Renderer".

Cette machine actuellement en cours de réalisation aura des performances de l'ordre de 1.000.000 de triangles avec ombrage de Phong par seconde.

Il nous semble toutefois qu'elle sera limitée par le fait qu'elle ne permet pas d'utiliser beaucoup d'effet pipe-line. Un "Graphics Processor" ne peut commencer à alimenter les unités de rendu que lorsqu'il a fini l'intégralité des traitements de ses objets.

La seule perte de performance liée au changement de concepts de Pixel-Planes 4 à Pixel-Planes 5 concerne la génération des ombres portées. En effet l'utilisation MIMD des processeurs "Renderer" ne permet plus d'utiliser efficacement l'algorithme de Crow pour les ombres portées.

II.2 Les Tableaux et Enseignements

Nous allons maintenant, à l'aide de tableaux, construire une "analyse comparative" des machines présentées. Vis -à-vis de chaque tableau nous mettons quelques commentaires permettant de préciser les sujets et objectifs du tableau et nous faisons une rapide synthèse permettant de tirer quelques enseignements.

Ces tableaux portent sur les thèmes suivants:

1) L'organisation des machines pour prendre en compte les différentes phases du processus de visualisation. Faut-il choisir une approche matérielle ou définir du logiciel sur des processeurs d'usage général utilisés en parallèle?

2) Les parallélismes et les effets pipe-line utilisés, orientation objets ou pixels des machines. Nous dressons deux tableaux sur ce thème.

3) Les algorithmes utilisés (ou utilisables) par ces machines. Nous nous intéressons plus particulièrement aux algorithmes d'élimination des parties cachées et de calcul des éclaircissements.

4) Les objets graphiques qu'elles peuvent visualiser et la stratégie pour transformer ces objets en pixels.

5) Les performances annoncées par les concepteurs de ces machines.

Nous concluons ce chapitre en évoquant un problème rarement étudié: les liaisons physiques entre les différentes unités des machines de synthèse d'images.

	PREPARATION	TRANSFORMATION	CONVERSION
Geometry System		Traite des segments pipe-line de 12 Geometry Engines	
PRC			Traite des polygones 4 unités en pipe-line
SAGE			Traite 1/4 de ligne écran pipe-line de 256 étages
Silicon Graphics 4D/240GTX	Hôte: 4 processeurs RISC en parallèle	5 circuits intégrés Geometry Engine	Traite des polygones 4 Unités en pipe-line
Stellar GS1000	Hôte: Processeur Multi-flots	Processeur Vectoriel de calculs	3 Unités en pipe-line 16 circuits en parallèle
PIXEL MACHINE	Hôte	pipe line de 9 ou 18 Processeurs DSP32	Matrice de Processeurs de signal DSP32
CAP	Hôte	Hôte	Matrice de Processeurs i80186
EXPERTS	Hôte	Hôte	M circuits ayant chacun N boitiers esclaves
Machine de Torborg	Hôte et utilisation d'une liste d'objets	8 Processeurs en parallèle	un circuit intégré composé de 4 unités
Machine GSP-NVS	Hôte	3 Unités en pipe-line	de nombreux circuits utilisés en pipe-line
Pixel-Planes 4	Hôte: Micro vax II : Workstation	Un Processeur graphique	un Processeur par pixel 64 Processeurs par chip utilisation SIMD
Pixel-Planes 5	Hôte: SUN 4	32 Processeurs i860 en parallèle	un Processeur par pixel utilisation MIMD en 16 unités



 Logiciel sur processeurs généraux
 Réalisation matérielle

TABLEAU 2.1 : Organisation des machines présentées

II.2.1 Organisation des machines présentées

II.2.1.1 Commentaires

Nous n'avons pas fait figurer dans le Tableau 2.1 les phases de définition et d'affichage du processus de visualisation, car elles ne sont pas des critères significatifs pour comparer les machines.

La définition des objets graphiques se fait toujours par logiciel sur le processeur hôte.

L'affichage est effectué par un processeur spécialisé généralement appelé module vidéo qui balaye la mémoire de trame pour remplir l'écran TRC au rythme du balayage écran. Ce module comprend une seconde unité permettant de reséquencer les pixels dans le cas où les machines ont une mémoire de trame répartie sur plusieurs sites.

II.2.1.2 Enseignements: solutions matérielles ou logicielles

Les performances des machines utilisant d'importantes unités réalisées en VLSI sont bien meilleures. Comparons par exemple les machines CAP et Experts qui utilisent les mêmes algorithmes, à peu près la même structure et qui sont contemporaines. Les performances de Experts qui utilise des circuits dédiés sont quatre fois supérieures.

Malheureusement un inconvénient majeur des solutions matérielles est qu'elles sont très figées. Par exemple, le circuit intégré PRC est très performant mais il ne peut traiter que des polygones avec ombrage de Gouraud. Il affichera une sphère comme une juxtaposition de facettes, mettra beaucoup de temps pour le faire et le résultat visuel sera médiocre.

Remarquons que les trois machines annonçant des résultats de l'ordre du million de triangles par seconde utilisent toutes un nombre important de processeurs intégrés en VLSI:

- 1024 processeurs-pixels (environ 4000 transistors par processeur) pour la machine construite à partir du circuit intégré SAGE.
- au moins 1000 processeurs " Δp " pour GSP-NVS. Chaque processeur utilise 25 000 transistors.
- environ $10 \cdot 128 \cdot 128$ processeurs-pixels pour la machine Pixel-Planes 5. 256 processeurs-pixels sont intégrés par circuit.

	Parallélisme Objets	Parallélisme Pixels
Geometry System	Entre les 4 composantes d'un segment en coordonnées homogènes (très bas niveau)	
PRC		
SAGE		Entre les pixels traités par des circuits intégrés différents en configuration parallèle (max 4)
Silicon Graphics 4D/240GTX		Les pixels sont générés par les 20 "Image Engine" travaillant en parallèle MIMD
Stellar GS1000		Les pixels sont traités 16 par 16 par les circuits intégrés "Toes" pour un objet donné
PIXEL MACHINE		Pour toutes les opérations situées au niveau pixel par la matrice de 16 à 64 DSP 32 à fonctionnement MIMD
CAP		Pour toutes les opérations situées au niveau pixel par la matrice de 180186 à fonctionnement MIMD
EXPERTS	Entre objets traités par des Processeurs "SLP" distincts Entre segments traités par des Processeurs "PXP" distincts	Les pixels sont générés par N*M Processeurs "PXP" fonctionnant en MIMD (M et N sont bornés par 16)
Machine de Torborg	Pour les transformations entre objets complexes traités par des processeurs distincts (max 8) Entre segments d'un triangle (max 4)	Génération des pixels au rythme de 20 par cycle par le circuit intégré de Rendu
Machine GSP-NVS	Pour les opérations de rendu Entre les triangles, égal au nombre de processeurs " Δp " (au moins un millier)	
Pixel-Planes 4		Tous les processeurs pixels traitent une expression linéaire simultanément (512*512 processeurs)
Pixel-Planes 5	pour les transformations Entre les objets traités par des processeurs objets distincts	Tous les processeurs pixels d'une zone traitent simultanément une expression Fonctionnement MIMD entre les 16 "Renderer"

TABLEAU 2.2 : Parallélismes exploités

	Pipe-line Objets	Pipe-line Pixels
Geometry System	Entre les différentes phases des transformations géométriques (12 étages)	
PRC	Entre les différents segments horizontaux d'un objet pour la génération des pixels (4 étages)	
SAGE	Pour la génération des pixels et le Z-buffer entre segments horizontaux de différents objets (256 étages)	
Silicon Graphics 4D/240GTX	Pour les transformations géométriques et la conversion en pixels (quelques étages)	
Stellar GS1000	Pour les différentes phases des transformations (processeur vectoriel) et de conversion en pixels (3 étages)	
PIXEL MACHINE	Pour la partie transformation en utilisant les "Pipe-Nodes" (9 ou 18 étages)	
CAP		
EXPERTS		
Machine de Torborg		
Machine GSP-NVS		Pour la conversion en pixels et le Z-buffer des triangles (nombre de processeurs " Δp " étages) Pour l'éclairage par les 16 "NVS"
Pixel-Planes 4	Entre les différentes expressions linéaires dans les arbres d'additionneurs	
Pixel-Planes 5	Entre les expressions quadratiques dans les arbres d'additionneurs	

TABLEAU 2.3 : Effets Pipe-line exploités

II.2.2 Parallélismes et effets pipe-lines exploités

II.2.2.1 Commentaires

Les Tableaux 2.2 et 2.3 précisent les parallélismes exploités par les machines. La grille des 4 possibilités proposées recouvre toutes les solutions envisageables. Il faut toutefois très précisément déterminer la partie du processus de visualisation parallélisée et quel est l'algorithme utilisé.

Rappelons que le parallélisme pixels et le pipe-line pixels ne sont utilisables que pour la phase du processus de visualisation que nous avons nommée conversion des objets en pixels. Les parties amont du processus de visualisation ne travaillent que sur les objets graphiques.

II.2.2.2 Enseignements

Le parallélisme utilisé par presque toutes les machines est le suivant:

a) Un parallélisme pixels et un pipe-line objets pour la conversion des objets en pixels et toutes les opérations situées au niveau pixel. Seule la machine Experts propose un parallélisme objets à ce niveau, mais ceci lui ôte toute possibilité d'exploiter un effet pipe-line et ses résultats sont modestes.

b) Un pipe-line objets pour les transformations géométriques. Deux machines, celle de Torborg et Pixel-Planes 5, exploitent un parallélisme objets à ce niveau.

c) Sur les machines les plus performantes les premières phases du processus de visualisation (définition des objets et préparation des images) sont parallélisées en utilisant plusieurs processeurs en parallèle comme machine hôte.

Une seule machine se distingue franchement: GSP-NVS. En effet cette machine exploite un parallélisme objets et un pipe-line pixels pour la conversion des objets en pixels et le Z-buffer.

	Elimination des parties cachées	Algorithmes pour calculer l'ombrage et autres possibilités pour le rendu
Geometry System		
PRC		Méthode d'interpolation de Gouraud
SAGE	Z-buffer par ligne	Méthode d'interpolation de Gouraud
Silicon Graphics 4D/240GTX	Z-buffer par ligne	Méthode d'interpolation de Gouraud
Stellar GS1000	Z-buffer	Méthode d'interpolation de Gouraud ou de Phong Possibilité de disposer de 16 sources lumineuses
PIXEL MACHINE	Z-buffer ou lancer de rayons	Toute solution est envisageable La machine est programmable
CAP	Z-buffer et visulisation directe d'arbre CSG	Méthode d'interpolation de Gouraud
EXPERTS	Z-buffer par ligne	Méthode d'interpolation de Gouraud
Machine de Torborg	Z-buffer par ligne	Méthode d'interpolation de Gouraud
Machine GSP-NVS	Z-buffer	Méthode d'interpolation de Phong Possibilité de disposer de 5 sources lumineuses
Pixel-Planes 4	Z-buffer et visulisation directe d'arbre CSG	Méthode d'interpolation de Gouraud Ombres portées par post-traitements (Crow) Effets de transparence en profondeur
Pixel-Planes 5	Z-buffer et visulisation directe d'arbre CSG ou Radiosité	Méthode d'interpolation de Phong Effets de transparence en profondeur Ou méthode de radiosité

TABLEAU 2.4 : Algorithmes Utilisés

II.2.3 Algorithmes utilisés

II.2.3.1 Commentaires

Seuls les algorithmes pour la partie finale du processus de visualisation sont présentés dans le Tableau 2.4. En effet ce sont eux qui déterminent la qualité des images obtenues et qui permettent de relativiser les performances annoncées (voir le Tableau 2.6).

II.2.3.2 Enseignements

II.2.3.2.1 L'élimination des parties cachées

Toutes les machines présentées dans ce travail utilisent l'algorithme du Z-buffer (ou du Z-buffer par ligne) pour éliminer les parties cachées bien que cet algorithme ne soit pas le plus performant a priori. De très nombreux autres algorithmes d'élimination des parties cachées ont été proposés. Les principaux sont:

- La subdivision en sous-espaces tridimensionnels proposé initialement par Warnock.
- Le classement des objets en fonction de leur profondeur (ce qui peut impliquer des découpage d'objets si ceux-ci ne sont pas disjoints en profondeur) dont les principales versions ont été proposées par Newell, Newell et Sancha puis par Fuchs.
- Les algorithmes à lignes de balayage.

Mais toutes ces méthodes utilisent des tris d'objets ou des découpages qui sont des opérations peu parallélisables.

L'enseignement à tirer de cette observation est que, pour définir des machines performantes, il faut choisir des algorithmes facilement et massivement parallélisables même si cela impose de réaliser de nombreuses opérations inutiles. Illustrons cette affirmation par un exemple: calculer la profondeur d'un objet en tous les pixels où il est présent si celui-ci est totalement caché est un travail inutile, c'est cependant la solution choisie sur toutes les machines actuellement développées.

II.2.3.3 Les calculs d'éclairément

Le choix de l'algorithme d'ombrage est très important, c'est en grande partie lui qui détermine la qualité de l'image obtenue. Les machines présentées montrent qu'il est facile et rapide d'appliquer un ombrage avec l'algorithme d'interpolation de Gouraud. De plus cet algorithme est aisément implémentable sur un processeur spécialisé réalisé en VLSI. Les circuits intégrés PRC et SAGE en sont de bons exemples. Malheureusement la qualité du rendu est médiocre.

Les résultats annoncés par les concepteurs de la machine Stellar GS1000 montrent qu'utiliser la méthode d'interpolation de Phong divise les performances de l'unité de rendu par quatre. Cette machine est pourtant particulièrement bien adaptée au type de calculs nécessaires à l'algorithme de Phong. La machine GSP-NVS intègre 16 calculateurs VLSI dédiés aux calculs de l'ombrage de Phong.

La qualité de l'image n'est pas à négliger pour obtenir de bonnes performances. Nous reviendrons plus en détail sur ce sujet dans le chapitre suivant.

Chapitre II: Présentation des Machines

	Objets traités par l'unité de conversion	Organisation pour convertir les objets en pixels
Geometry System	Segments 3D Coordonnées homogènes	
PRC	Polygones 3D quelconques	Découpage en segments horizontaux
SAGE	Tous, Polygones avec les interpolateurs verticaux	Découpage en segments horizontaux
Silicon Graphics 4D/240GTX	Polygones 3D	Découpage en trapèzes et triangles eux mêmes découpés en segments verticaux
Stellar GS1000	Segments Triangles 3D Sphères	Codage avec des expressions linéaires Utilisation de ces expressions par blocs de 16 pixels
PIXEL MACHINE	Tous	Quelconque En fonction de l'algorithme implémenté
CAP	Polygones 3D	
EXPERTS	Polygones 3D	Découpage en segments horizontaux
Machine de Torborg	Segments et petits triangles rectangles	Découpage en segments horizontaux
Machine GSP-NVS	Triangles 3D	Traitement direct des triangles 3D
Pixel-Planes 4	Polygones 3D convexes Sphères	Codage en expressions linéaires
Pixel-Planes 5	Polygones 3D convexes Sphères Cylindres	Codage en expressions linéaires ou quadratiques

TABLEAU 2.5 : Le Processus de conversion en pixels

II.2.4 Le processus de conversion en pixels.

II.2.4.1 Commentaires

Le tableau 2.5 nous semble particulièrement important car il précise quels sont les objets graphiques que peut traiter l'unité de conversion des différentes machines. Nous explicitons quel est le processus de découpage ou/et de codage de ces objets pour engendrer les pixels.

II.2.4.2 Enseignements

Toutes les machines que nous avons présentées (sauf Geometry System) parallélisent la conversion des objets en pixels. Ce qui n'est en rien surprenant, c'est bien à ce niveau que se présente la plus importante masse de calculs. Utiliser l'algorithme du Z-buffer impose de valoriser tous les objets en tous les pixels où ils sont présents. Afficher 1.000 objets de surface moyenne 1.000 pixels nécessite donc 1.000.000 d'opérations.

Pour paralléliser cette conversion, deux stratégies très différentes ont été envisagées:

a) Utiliser des processeurs associés aux pixels de la mémoire de trame. C'est la méthode choisie par presque toutes les machines. Ces machines peuvent se classer en deux sous-groupes:

- Celles dont les processeurs sont affectés à une partie figée de la mémoire de trame. Parfois la répartition des parties peut être programmée en fonction de l'algorithme utilisé.

- Celles ayant un (ou des) processeur (s) pouvant travailler sur un bloc de la mémoire de trame et qui traitent successivement les différents blocs de la mémoire de trame. Seules deux machines utilisent cette solution: Pixel-Planes 5 dont un processeur traite un bloc de 128x128 pixels et la Stellar GS1000 avec un processeur traitant un bloc de 4x4 pixels.

b) Utiliser des processeurs associés aux objets. Seule la machine GSP-NVS adopte cette solution en utilisant en parallèle un certain nombre de processeurs " Δp " traitant chacun un triangle.

Remarquons par ailleurs que presque toutes les machines actuellement développées sont définies pour visualiser des polygones 3D voire des polygones particuliers: triangles, quadrilatères convexes (voir Tableau 2.5). Seules quelques machines proposent des solutions pour visualiser directement des sphères, et une seule des cylindres (Pixel-Planes 5).

	Performance en nombre d'objets affichés par seconde.
Geometry System	100 000 segments transformés pour un chip Geometry Engine 4 fois plus avec le pipe-line de 12 chips
PRC	1000 polygones de 1000 pixels de surface moyenne avec ombrage de Gouraud
SAGE	1 000 000 polygones de 1000 pixels de surface moyenne avec ombrage de Gouraud et Z-buffer avec 4 chips en parallèle interpolateurs verticaux
Silicon Graphics 4D/240GTX	100 000 polygones de 100 pixels de surface avec ombrage de Gouraud et Z-buffer
Stellar GS1000	150 000 triangles de 100 pixels de surface avec ombrage de Gouraud et Z-buffer pour une source lumineuse 4 fois moins avec un ombrage de Phong dans les même conditions
PIXEL MACHINE	non précisé
CAP	700 polygones avec ombrage de Gouraud et Z-buffer
EXPERTS	3000 polygones avec ombrage de Gouraud et Z-buffer
Machine de Torborg	100 000 triangles de surface moyenne 19 pixels avec ombrage de Gouraud et Z-buffer
Machine GSP-NVS	500 000 triangles avec ombrage de Phong et Z-buffer
Pixel-Planes 4	40 000 triangles de taille quelconque avec ombrage de Gouraud et Z-buffer 13 000 sphères avec ombrage et Z-buffer
Pixel-Planes 5	1 000 000 triangles avec ombrage de Phong et Z-buffer

TABLEAU 2.6 : Performances annoncées

II.2.5 Performances annoncées

II.2.5.1 Commentaires

Les chiffres figurant dans le Tableau 2.6 sont ceux annoncés dans les articles à partir desquels nous avons construit cette présentation. Ils sont à prendre avec réserve pour les raisons suivantes:

i) parfois les performances données sont celles du module matériel de conversion en pixels, mais le reste de la machine ne permet pas d'utiliser ce module au maximum de ses possibilités.

ii) souvent ces chiffres ne sont atteints que pour des cas particuliers favorables, par exemple: a) dans le cas où les objets sont bien répartis sur l'écran, b) uniquement pour de petits objets (surface inférieure à 100 pixels).

iii) ils sont à mettre en relation avec les moyens mis en oeuvre.

II.2.5.2 Enseignement

Ce sont les machines utilisant les modules matériels les plus importants qui obtiennent les meilleurs résultats en nombre de polygones par seconde. Ceci n'est en rien surprenant; ces machines ont été réalisées spécifiquement pour cela.

II.2.6 Conclusion

Nous allons terminer cette étude des machines en mettant en évidence un problème délicat à résoudre et pourtant rarement étudié, à savoir:

Les communications entre unités

Les machines commerciales de haut de gamme nécessitent déjà des liaisons entre unités ayant des débits considérables:

- Deux bus de 64 bits parallèles permettant des transferts de données entre unités de 64 Moctets par seconde pour la machine Silicon Graphics 4D/240GTX
- Des liaisons 512 bits parallèle à 20 Mhz pour le chemin de données de la Stellar GS1000.
- Les concepteurs de Pixels-Planes 5 pensent utiliser un chemin de données de 8 voies de 32bits parallèles chacune (à 20 Mhz) pour alimenter en expressions linéaires et quadratiques les processeurs "Renderer".

A notre avis les communications entre unités deviendront problématiques sur les futures machines. Des problèmes se poseraient sans doute pour alimenter un processeur de rendu utilisant une dizaine de circuits intégrés SAGE. Pour définir des machines capables de visualiser des scènes complexes en temps réel le paramètre des communications est crucial et doit être pris en compte dès la conception.

Résumé

Actuellement un des objectifs des équipes concevant des machines de visualisation est d'atteindre une animation en temps réel.

Cet objectif impose de remettre en cause les principes fondamentaux sur lesquels reposent toutes les machines commercialisées. Nous avons tenté, dans ce chapitre, de dégager les nouvelles bases, sous-jacentes dans les définitions des machines que nous avons présentées, qui permettront de faire de l'animation interactive.

Il semble admis que pour construire des processeurs d'affichage très rapides:

- il faut réaliser un module matériel spécifique réalisant la conversion des objets en pixels et toutes les opérations situées en aval.
- seule l'utilisation d'un parallélisme massif permet d'augmenter les performances d'un ordre de grandeur. Les machines utilisant les concepts classiques ne pourront jamais afficher en temps réel des scènes complexes.
- il faudra utiliser un module hôte multiprocesseur pour alimenter les modules graphiques performants actuellement en cours de définition. Se poseront alors de gros problèmes de communications entre unités éventuellement insolubles avec certaines solutions architecturales.

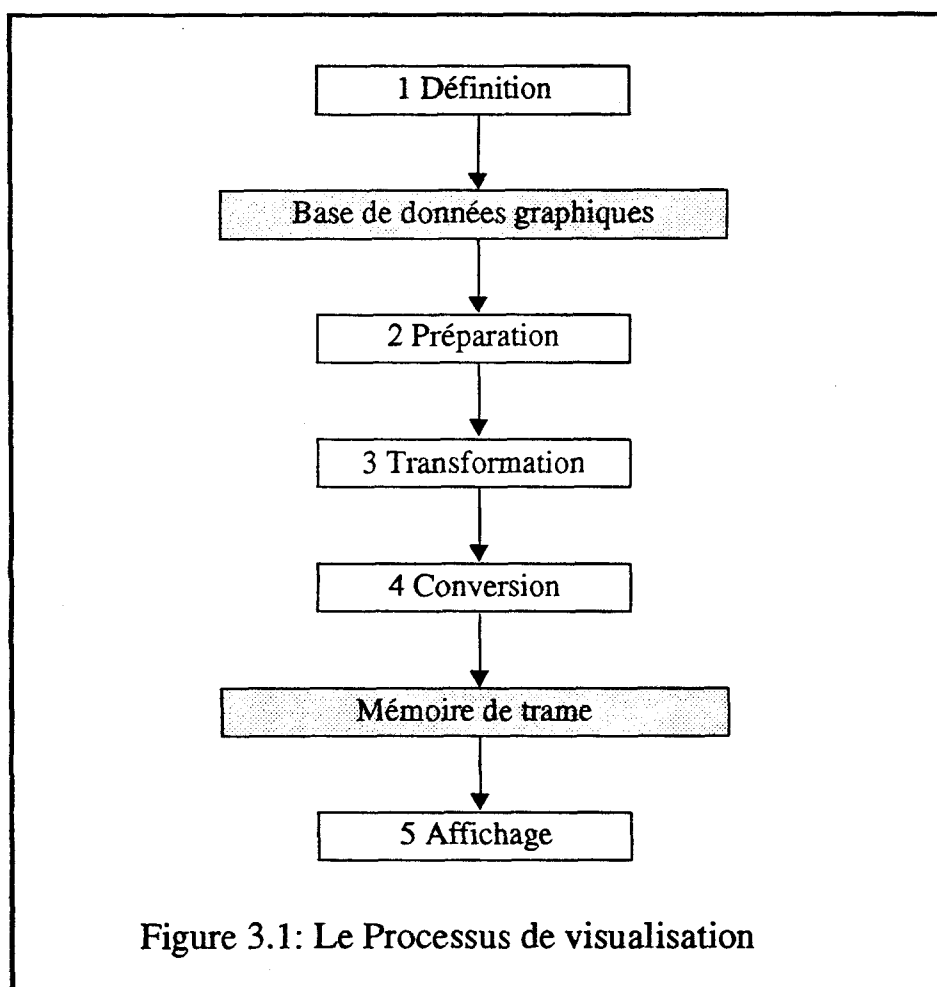
Terminons ce chapitre en remarquant qu'une limite importante, commune à toutes les machines, est la pauvreté des primitives graphiques qu'elles peuvent traiter, le plus souvent des triangles 3D avec interpolations.

III VERS UN RENDU TEMPS REEL, ALGORITHMES ET ARCHITECTURE

Ce chapitre comporte deux parties. Nous commençons par une analyse algorithmique de l'organisation du processus de visualisation pour tenir compte des contraintes imposées par le temps réel. Ensuite nous présentons des architectures massivement parallèles envisageables pour implanter ce processus de visualisation. Nous concluons en choisissant la structure de la machine I.M.O.G.E.N.E.

III.1 Organiser le processus de visualisation: Algorithmes

Nous allons dans cette partie étudier comment organiser le processus de visualisation [Martin82] pour afficher des images en temps réel. Nous prenons comme point de départ le découpage en cinq parties que nous avons proposé au premier chapitre (voir Figure 3.1). En fonction des conclusions du chapitre II et de nos objectifs, nous allons choisir à quels stades du processus de visualisation il est préférable de réaliser les tâches suivantes: l'élimination des parties cachées, les autres opérations inter-objets et les calculs d'éclairément. Nous mènerons parallèlement une présentation des algorithmes existants pour effectuer ces tâches, en présentant s'ils sont utilisables dans une machine d'affichage temps réel.



Rappelons, avant d'entrer dans les détails, que pour afficher en temps réel une scène il est impératif que les parties 2, 3, 4 et 5 du processus de visualisation soient réalisables dans le temps imparti à l'affichage d'une trame.

III.1.1 Les Opérations inter-objets

Un des obstacles majeurs à la parallélisation du processus de visualisation est que les traitements des différents objets présents dans une scène ne sont pas indépendants. Si les différents objets étaient indépendants, la solution évidente pour paralléliser serait de répartir les objets sur plusieurs processeurs fonctionnant en parallèle. Le premier des traitements inter-objets est l'élimination des parties cachées. Nous détaillons dans le paragraphe suivant qu'un algorithme (le Z-buffer) et une solution d'implantation (après la conversion des objets en pixels) sont actuellement utilisés par toutes les machines. Nous extrapolons ce résultat dans les paragraphes suivants pour déterminer comment réaliser les autres opérations inter-objets. Outre l'élimination des parties cachées, nous classons dans les opérations inter-objets la mise en place des ombres portées, le traitement des objets transparents (pour lesquels on ne peut utiliser les algorithmes d'élimination des parties cachées) et la visualisation directe des objets construits à partir d'arbres CSG. Nous ne traitons pas l'anti-aliasage comme un problème inter-objets, ce problème est beaucoup plus général et sera traité séparément au paragraphe III.1.5.

III.1.1.1 L'élimination des parties cachées

Nous allons présenter brièvement les quatre principales familles de procédés définies pour éliminer les parties cachées [SuthSS74][Rogers85].

III.1.1.1.1 Les Algorithmes par Subdivision

Le principe de base est de réaliser une subdivision récursive du volume englobant la scène à visualiser en fonction de l'écran jusqu'à aboutir à une situation décidable.

La première solution exploitant cette idée a été proposée par Warnock[Warnoc68]. Catmull [Catmul75] et Griffiths[Griffi75] ont proposé d'autres algorithmes utilisant un procédé analogue. Le point de départ est un découpage la scène en quatre parties en fonction de l'écran. Si, après un découpage, on ne peut décider quoi afficher dans une partie de l'écran on renouvelle le découpage. On décide d'afficher dans une zone donnée si un objet remplit totalement la zone ou si la zone est totalement vide. Dans chaque zone on doit effectuer un classement des objets et des tests pour déterminer si un des objet recouvre intégralement cette zone. Dans le cas le plus défavorable, le découpage s'arrête quand une zone a la taille d'un pixel.

III.1.1.1.2 Les Algorithmes à Listes de Priorité

Les algorithmes de cette famille ont pour principale caractéristique de pré-calculer un ordre de visibilité avant l'affichage.

Le plus connu de ces algorithmes est celui de Newell, Newell et Sancha [NeweNS72] aussi appelé algorithme du peintre. L'algorithme se déroule en deux phases:

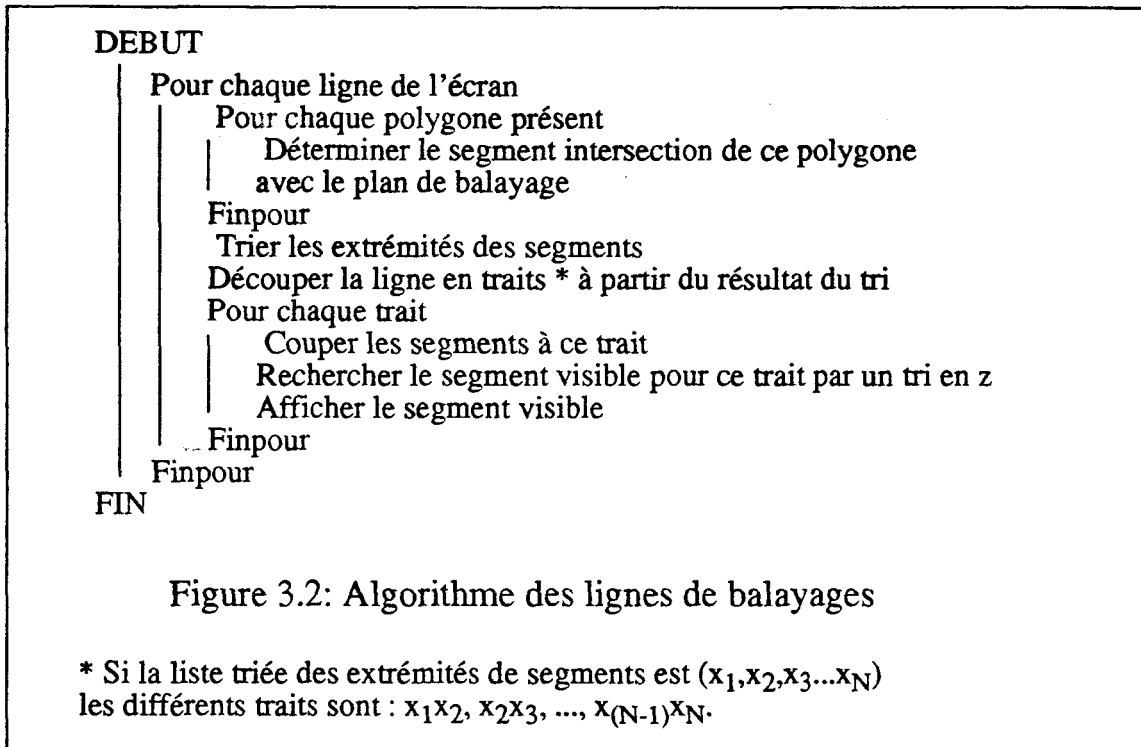
- 1) On trie les objets en fonction de leur distance à l'observateur. Si deux objets ne sont pas disjoints sur ce critère, on découpe l'un des objets en plusieurs sous-objets.
- 2) On affiche l'ensemble des objets en partant du plus éloigné de l'observateur.

L'équipe de Fuchs [FuchAG83] a proposé une variante de cet algorithme, effectuant un pré-traitement inter-objets durant lequel les objets sont découpés les uns par rapport aux

autres. Cet algorithme est très efficace tant qu'on ne déplace pas les objets, tout déplacement impliquant de refaire le coûteux pré-traitement.

III.1.1.1.3 Les algorithmes de lignes de balayage

Cette famille d'algorithmes propose de traiter l'élimination des parties cachées ligne par ligne dans l'espace image [Boukni70][Watkin70]. La figure 3.2 donne une version de cet algorithme pour traiter des facettes polygonales.



Cet algorithme nécessite une très grande quantité de tris et de découpages d'objets.

III.1.1.1.4 Le Z-buffer

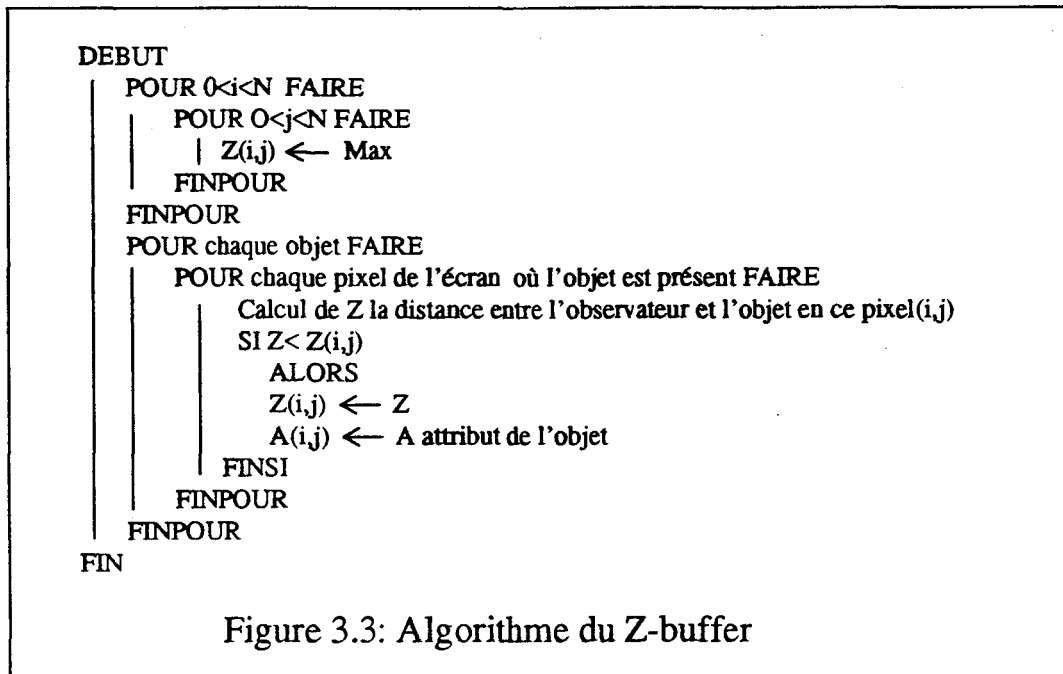
Dans sa version courante cet algorithme nécessite une matrice de pixels ayant la taille de l'écran graphique avec une cinquantaine de bits par pixel au minimum. Cette mémoire sert à stocker des valeurs de profondeur Z et des attributs A. On appelle attribut les valeurs qui permettront de déterminer quoi afficher en chaque pixel. Cet attribut peut être des intensités si les calculs d'éclairéments ont déjà été effectués, ou les composantes d'un vecteur normal et les caractéristiques propres de l'objet si l'éclairément est calculé après l'élimination des parties cachées.

Voici le principe de fonctionnement de cet algorithme (voir Figure 3.3):

1) Les profondeurs en tous les pixels de la matrice sont initialisées à la plus grande valeur possible.

2) pour chaque objet de la scène on détermine sa profondeur en tous les pixels où il est présent. Si en un pixel donné la profondeur de l'objet est inférieure à celle stockée dans la mémoire on stocke dans la mémoire pour ce pixel le couple (Z,A) de cet objet. Les objets

sont traités dans un ordre quelconque.



Cet algorithme est d'un usage très général:

Les objets traités peuvent être d'un type quelconque à la seule condition de pouvoir calculer en tout pixel la distance de l'observateur à l'objet.

Aucun tri ni aucun autre pré-traitement inter-objets ne sont utilisés. Les objets sont traités dans un ordre quelconque.

La seule limite de cet algorithme est le nombre considérable d'accès à la mémoire qu'il faut réaliser.

Il existe une version particulière du Z-buffer où le même procédé est utilisé ligne par ligne. Une ligne écran donnée correspond à la projection d'un plan perpendiculaire à l'écran. On détermine les segments intersections des objets avec ce plan. On applique l'algorithme précédent sur ces segments pour obtenir le résultat pour une ligne.

III.1.1.1.5 Justification du choix du Z-buffer, Conséquences

Les éléments permettant d'expliquer pourquoi le Z-buffer est la méthode unanimement retenue sont :

- 1) La simplicité de l'algorithme qui permet une implantation matérielle.
- 2) Les facettes polygonales qui sont utilisées par presque toutes les unités de rendu sont facilement prises en compte par cet algorithme. En effet une interpolation bilinéaire à partir des profondeurs aux sommets permet de déterminer la profondeur en tout pixel.
- 3) L'absence de tout traitement inter-objets avant la conversion des objets en pixels.

Ce dernier argument est, à notre avis, le principal; en effet tous les autres algorithmes nécessitent soit un tri des objets, soit des découpages d'objets au cours de la visualisation d'une image. Ces opérations sont difficilement parallélisables et on ne sait pas a priori le

temps que prendra un tel traitement, celui-ci étant fonction des positions des objets les uns par rapport aux autres.

Le Z-buffer est un algorithme qui élimine les parties cachées après la conversion des objets en pixels. Les traitements des différents objets sont indépendants et, pour un objet donné, les pixels le composant peuvent être traités en parallèle. On peut donc paralléliser massivement cet algorithme en utilisant de nombreux processeurs en parallèle ou en pipeline.

En contrepartie le Z-buffer impose de convertir en pixels tous les objets indépendamment de leur visibilité par l'observateur. Les objets sont traités dans un ordre quelconque.

III.1.1.1.6 Conclusion

Nous pouvons extrapoler, à partir de ce travail sur l'élimination des parties cachées, les affirmations suivantes:

Un algorithme est bien adapté aux machines de rendu temps réel si les traitements des différents objets (ou des différents pixels) sont indépendants. Ceci pour utiliser un parallélisme très important sur les objets (ou sur les pixels). Cette affirmation justifie l'abandon de la notion de "parallélisation du processus" pour l'étude des machines rapides de visualisation.

Dans la mesure du possible il faut reporter les opérations inter-objets après la conversion des objets en pixels. Ceci permet d'utiliser davantage le parallélisme pour les premières phases (jusqu'à la conversion en pixels) du processus de visualisation, les traitements de différents objets étant distincts. Paralléliser les traitements effectués après la conversion est aussi envisageable; en effet ils sont indépendants d'un pixel à l'autre. Faire ce choix implique que certaines opérations inter-objets seront difficilement ou partiellement réalisables.

La seconde affirmation implique que les traitements en chaque pixel deviendront complexes, car il faudra effectuer à ce niveau les opérations inter-objets. Mais les traitements des différents pixels d'une image étant indépendants, on peut utiliser des solutions parallèles.

III.1.1.2 Les ombres portées

L'affichage des ombres portées améliore considérablement le réalisme d'une image. La complexité de cette opération est voisine de celle de l'élimination des parties cachées. En effet tout objet peut faire de l'ombre à tout autre relativement à une source lumineuse donnée. On peut regrouper en quatre classes les différentes méthodes proposées

- 1) le découpage géométrique
- 2) la méthode de Crow
- 3) la méthode de Williams
- 4) le lancer de rayons

Nous ne décrivons dans ce travail que les trois premières solutions, le lancer de rayon qui prend naturellement en compte les ombres portées ayant été exclu a priori de cette étude vu son coût.

III.1.1.2.1 Le découpage géométrique

Deux familles d'algorithmes se basent sur un découpage géométrique. Une première proposée par Appel [Appel 68] qui détermine les frontières des ombres par un algorithme de balayage de ligne au cours de l'élimination des parties cachées. La seconde solution consiste à effectuer deux passes. La première divise les polygones en deux classes, ceux à l'ombre et les autres. Les polygones partiellement éclairés sont divisés. La seconde passe est l'élimination des parties cachées par un algorithme à listes de priorité. Le plus connu des algorithmes de cette seconde famille a été proposé par Atherton [AtheWG78]

Ces algorithmes nécessitent des tris ou des découpages d'objets, sont peu parallélisables et ne permettent pas une implémentation matérielle. De plus ils travaillent sur les objets avant leur conversion en pixels.

III.1.1.2.2 La méthode de Crow

Cette méthode [Crow77][Berger86] a été initialement proposée par Crow pour les scènes construites à partir de polygones 3D convexes. Bergeron a généralisé cette méthode à d'autres objets comme les polygones avec "trou" ou non plans.

A chaque objet éclairé est associé un volume d'ombre lui-même considéré comme un objet. Pour une facette triangulaire, le volume d'ombre est délimité par la facette triangulaire et les trois demi-plans définis par la source lumineuse et un coté du triangle (voir Figure 3.4). Les volumes d'ombre sont des objets invisibles mais ils peuvent obscurcir les objets visibles. Ils sont pris en compte après l'élimination des parties cachées.

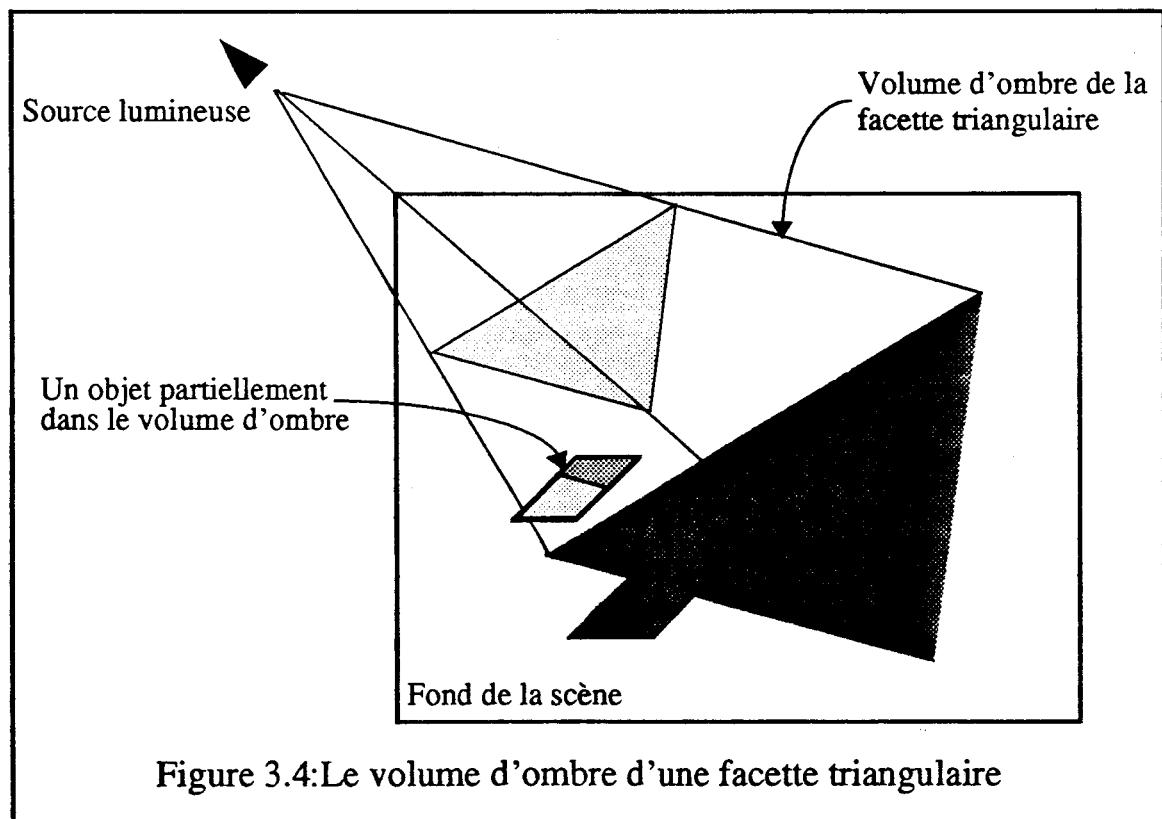


Figure 3.4: Le volume d'ombre d'une facette triangulaire

Cette méthode est utilisée par la machine Pixel-Planes 4. Après l'élimination des parties cachées sur l'ensemble des objets visibles, chaque processeur pixel conserve l'objet visible en son pixel avec sa profondeur. Les volumes d'ombre sont traités ensuite; si l'objet visible est dans le volume d'ombre, on modifie l'intensité de l'éclairage.

Cette méthode est très coûteuse car le nombre de volumes d'ombre à traiter augmente avec le nombre de sources lumineuses. Elle se prête cependant bien à une implémentation matérielle: en effet les différentes ombres peuvent être appliquées dans un ordre quelconque. Le traitement des ombres est effectué après la conversion des objets en pixels, paralléliser de façon importante est donc envisageable.

III.1.1.2.3 La méthode de Williams

Cette méthode [Willia78] est basée sur la technique du Z-buffer.

A partir de chaque source lumineuse, considérée comme un oeil, on détermine la profondeur de la surface vue par l'algorithme du Z-buffer dans une mémoire de trame auxiliaire. Les objets ou parties d'objets éclairés sont ainsi déterminés. Parallèlement, on effectue dans la mémoire de trame principale l'élimination des parties cachées. On mémorise en tout pixel la profondeur p de l'objet vu.

A tout pixel (i,j) correspond le point M de coordonnées (i,j,p) . On effectue un changement de repère vers le repère de la source lumineuse, M a pour coordonnées (x,y,z) dans ce repère. Si la valeur z est égale (à epsilon près) à celle stockée dans la mémoire de trame à l'adresse (x,y) , c'est que le point M est éclairé par la source lumineuse.

Le changement de repère peut, dans certain cas, entraîner d'importants problèmes d'aliassage sur l'image finale. En effet plusieurs pixels de l'image peuvent correspondre à un même point de la mémoire auxiliaire.

Tout comme le Z-buffer, cette méthode de calcul des ombres portées peut être implantée matériellement. Son coût est élevé car elle nécessite au minimum d'utiliser deux mémoires de la taille de l'écran d'affichage. Cette méthode traite les ombres portées après la conversion des objets en pixels.

III.1.1.2.4 Conclusion sur les ombres portées

Les deux dernières méthodes présentées calculent les ombres portées après la conversion des objets en pixels, donc elles sont parallélisables. De plus elles peuvent être implantées matériellement.

Malheureusement, ces deux méthodes sont très coûteuses. Celle de Crow multiplie le nombre d'objets de la scène à visualiser par le nombre de sources lumineuses et nécessite d'utiliser un algorithme plus complexe que le Z-buffer. Celle de Williams nécessite d'employer autant de mémoires de trame que de sources lumineuses pour atteindre le temps réel. Cette méthode peut entraîner des conflits d'accès à la mémoire auxiliaire si les différents pixels de la mémoire de trame sont traités en parallèle.

III.1.1.3 Traitement des objets transparents [KayGre79][Mammen89]

Un traitement réaliste des objets transparents ne peut se faire qu'en utilisant la méthode du lancer de rayons avec prise en compte des réfractions. Toutefois il est possible, en utilisant la méthode de rendu géométrique, de prendre en compte les transparences en profondeur.

Quand un objet transparent est devant un objet opaque pour un pixel donné, on peut calculer l'intensité lumineuse au pixel en question avec la formule

$$I = t * I1 + (1-t) * I2 \text{ avec } 0 \leq t \leq 1$$

où I1 est l'intensité lumineuse pour l'objet transparent, I2 celle de l'objet juste derrière et t est le coefficient de transparence de l'objet transparent.

Les objets transparents ne peuvent pas être traités parmi les objets opaques par l'algorithme du Z-buffer. Il faut dans un premier temps effectuer l'élimination des parties cachées pour l'ensemble des objets opaques. Les objets transparents sont traités ensuite, un par un, en commençant par le plus proche devant l'objet visible. De cette façon, les objets transparents sont traités après la conversion des objets en pixels.

Ce traitement peut facilement être câblé. Il l'est par exemple sur les machines Silicon Graphics et Stellar.

Lorsqu'il y a plusieurs objets transparents dans une scène il faut effectuer un tri en profondeur de ces objets pour les traiter du plus éloigné au plus près de l'observateur. Cette opération est peu parallélisable et ne peut être réalisée que par logiciel. Afficher, en temps réel, une scène comprenant beaucoup d'objets transparents est difficilement envisageable.

III.1.1.4 La visualisation directe d'arbres CSG

La modélisation la plus utilisée en C.A.O. est l'arbre de construction C.S.G. (Constructive Solid Geometry).

La structure d'un arbre C.S.G. est un arbre binaire dont les noeuds sont des opérateurs booléens (Intersection, Union et Différence) et les feuilles les objets primitifs [Tilove80]. Les objets primitifs généralement utilisés sont: les polyèdres, les sphères, les cylindres et les cônes. Il est envisageable d'utiliser des objets primitifs plus complexes.

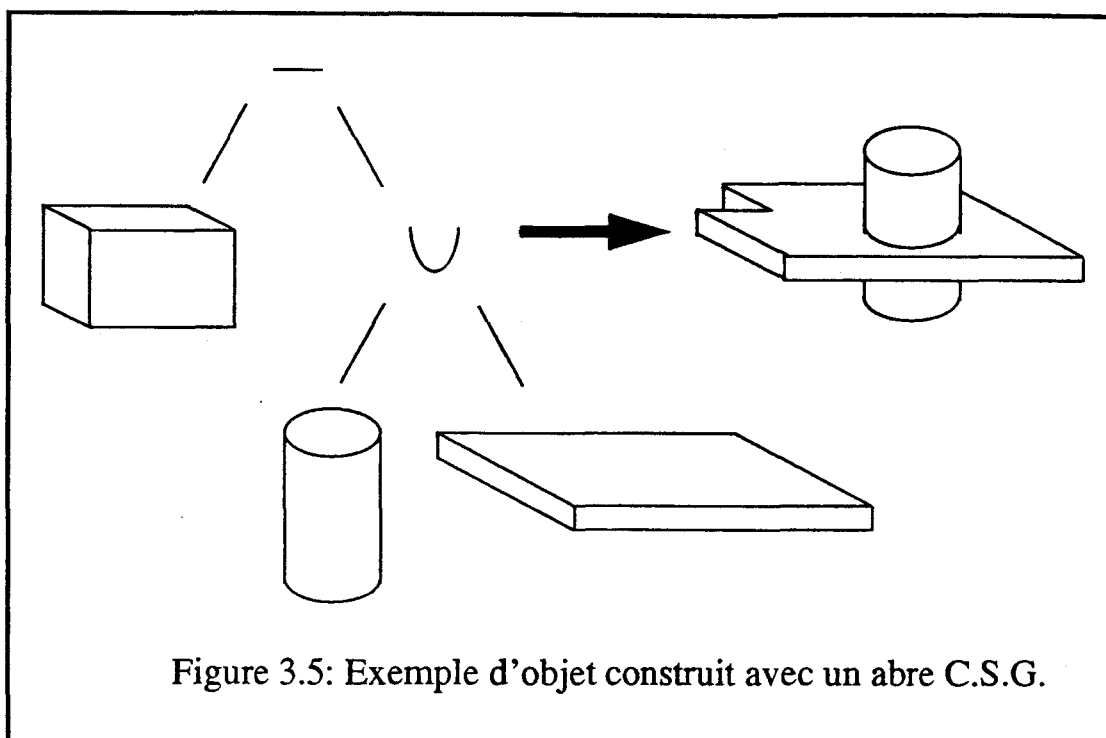


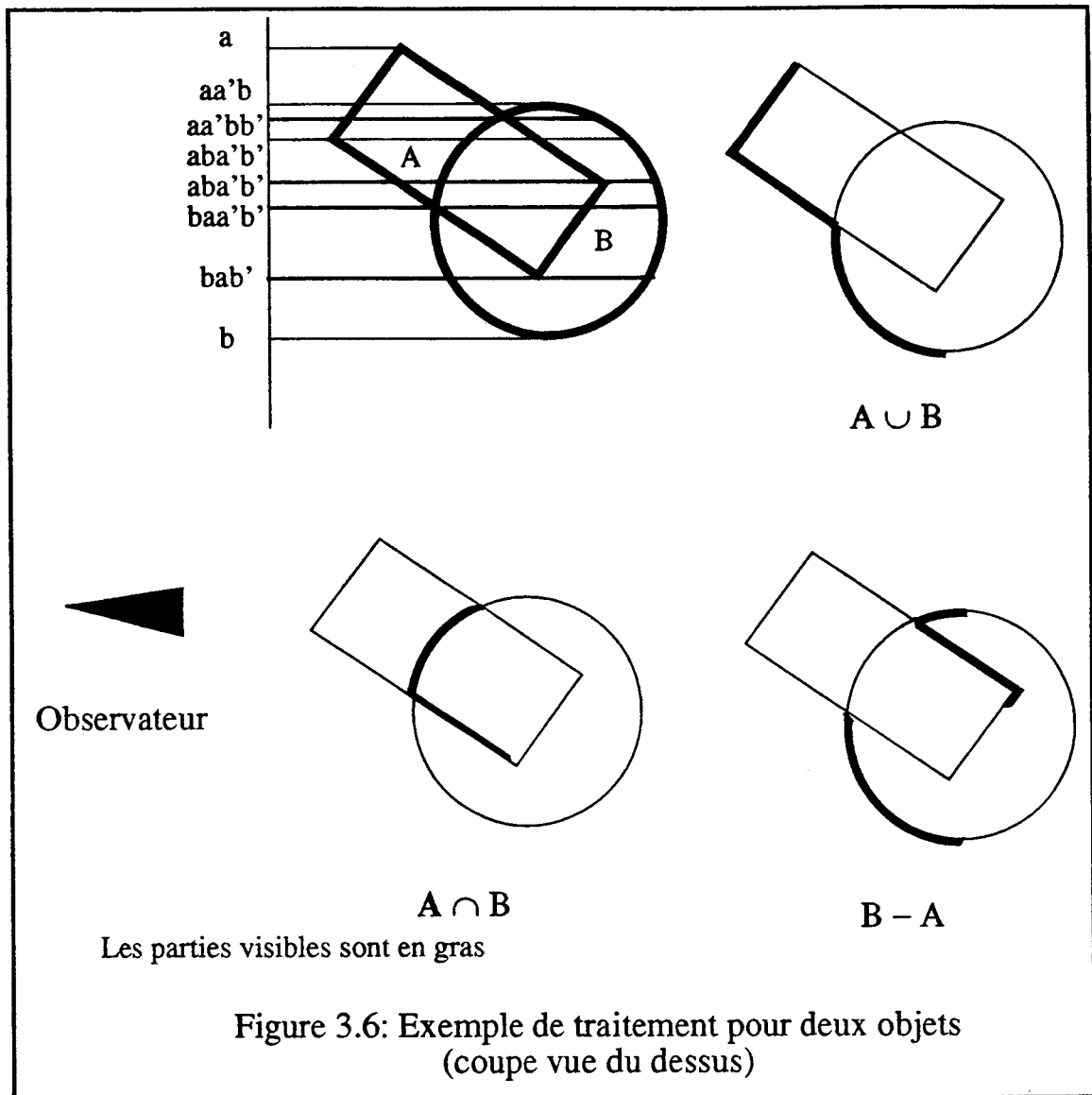
Figure 3.5: Exemple d'objet construit avec un arbre C.S.G.

L'opération consistant à déterminer toutes les faces de l'objet résultat à partir de la structure d'arbre est une opération très coûteuse [Jansen85]. En effet elle nécessite de calculer toutes les intersections entre les objets primitifs. Ces calculs sont à refaire à chaque modification des positions relatives des objets primitifs les uns par rapport au autres. Cette opération ne peut être effectuée en temps réel.

C'est pourquoi sont proposés des algorithmes où les différents objets primitifs sont convertis en pixels séparément, l'objet final étant construit ensuite en chaque pixel. Avec ces solutions, il est envisageable d'afficher des arbres CSG en temps réel en utilisant du matériel spécialisé. Nous allons détailler plusieurs propositions faites dans cette direction.

III.1.1.4.1 L'algorithme d'Atherton [Athert83]

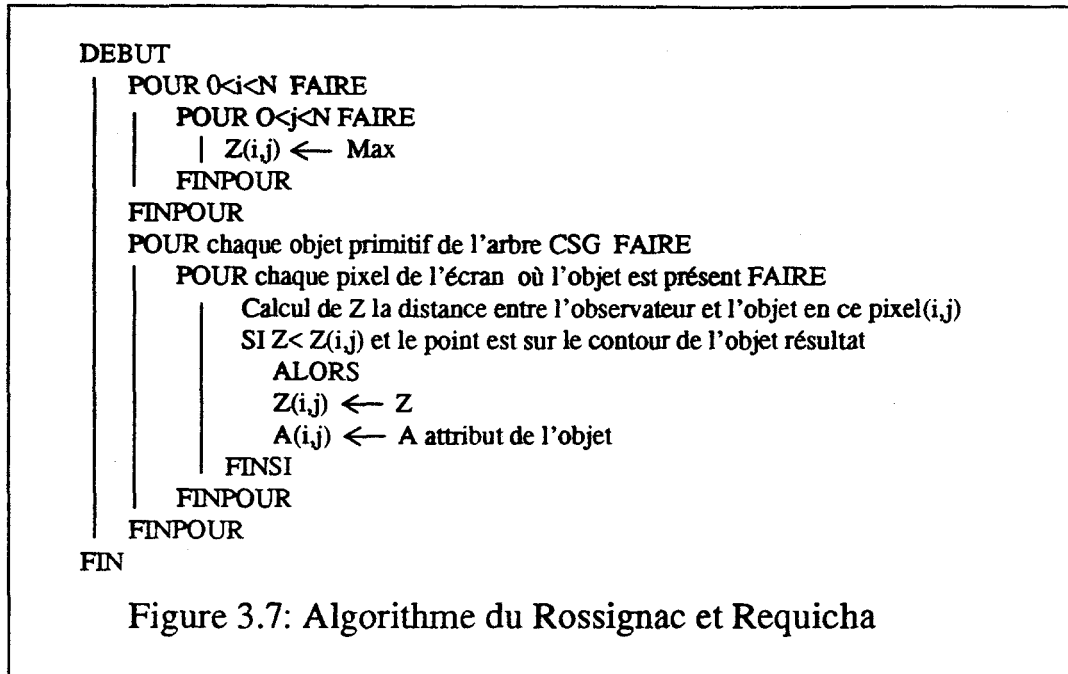
Atherton a proposé un algorithme permettant de visualiser directement des objets CSG à partir de l'algorithme, d'élimination des parties cachées, des lignes de balayage. Pour chaque trait (mot défini Figure 3.3) on effectue une recherche du segment visible en fonction de l'arbre CSG et des objets primitifs présents en ce trait. La Figure 3.6 est un exemple simple du traitement à effectuer pour deux objets.



Pour déterminer l'objet visible en chaque trait, il n'est pas toujours nécessaire d'effectuer un traitement complet de l'arbre CSG. En utilisant la cohérence de la structure CSG, on peut largement simplifier le travail. Par exemple, si un seul objet est présent pour un trait donné, le traitement est élémentaire.

III.1.1.4.2 L'algorithme de Rossignac et Requicha

Cet algorithme [RossRe86] est une version de l'algorithme du Z-buffer qui permet de visualiser les objets construits à partir d'arbre CSG (voir Figure 3.7).



Les objets primitifs sont définis à l'aide d'expressions algébriques $f(x,y,z)$. Pour évaluer si un point P est sur le contour ou intérieur à un objet on évalue $f(P)$ (P est sur le contour si $f(P)=0$ à epsilon près).

Pour déterminer si un point est sur le contour d'un objet défini par un arbre CSG on utilise le résultat suivant: un point est sur le contour de l'objet composé s'il appartient à la frontière d'une primitive. On peut donc déterminer si un point est sur le contour de l'objet résultat en utilisant un algorithme récursif de parcours de l'arbre et en évaluant des expressions algébriques.

Une originalité de cet algorithme est la méthode avec laquelle il détermine les pixels où les objets sont présents. Les surfaces des primitives sont définies par des expressions paramétrées en (u,v) , un choix de $(\Delta u, \Delta v)$ permet de définir un maillage de points qui sont ensuite projetés. La difficulté réside dans le choix du maillage: trop fin il engendre des calculs inutiles (plusieurs point du maillage sont sur le même pixel), trop gros il y aura des "trous" dans les objets.

III.1.1.4.3 L'algorithme de Jansen

Cet algorithme [Jansen 86] est défini pour la machine Pixel-Planes 4, mais peut être utilisé sur toute machine utilisant un processeur par pixel et un mécanisme permettant de fournir efficacement les objets à tous les processeurs pixels.

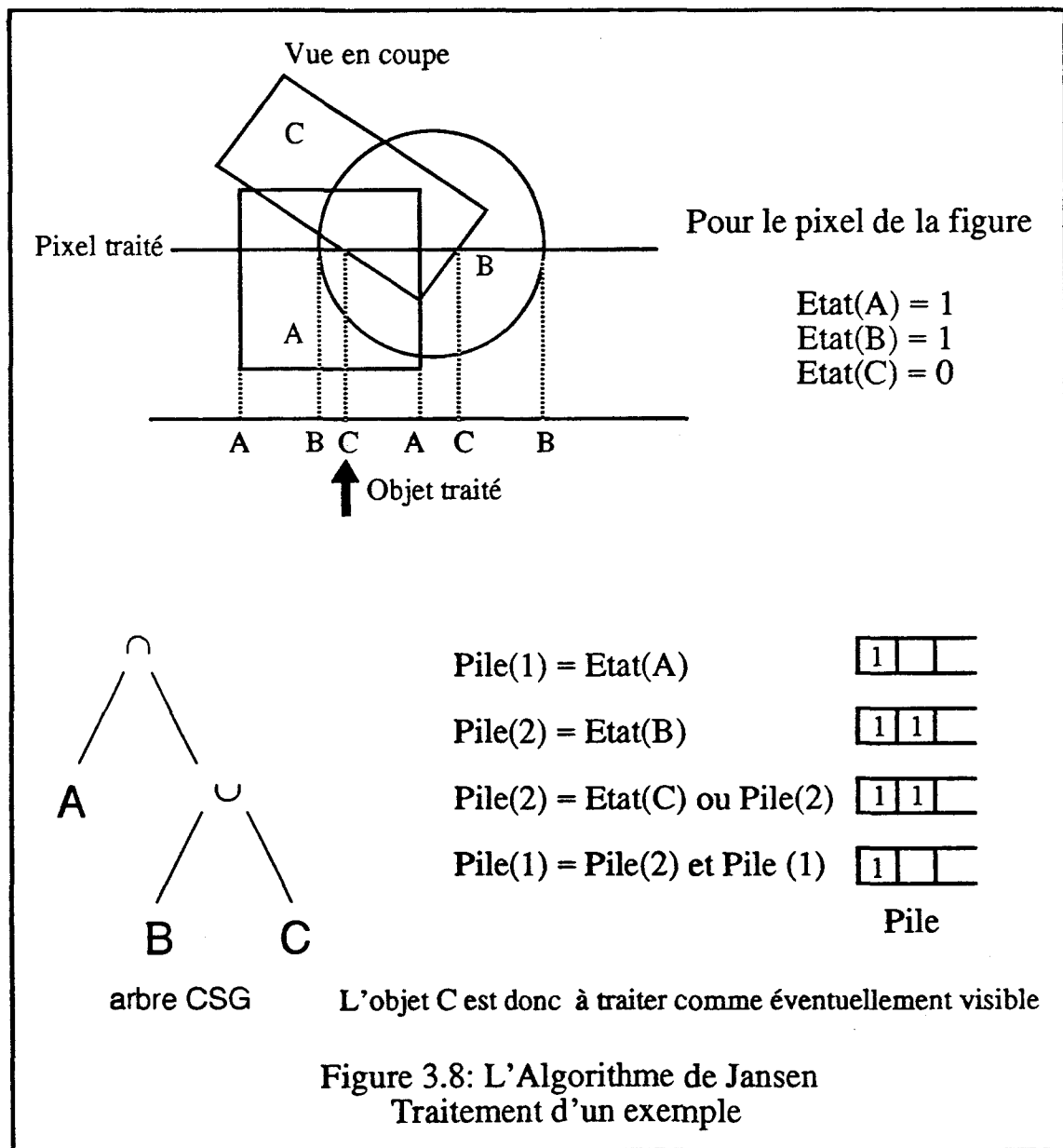
Cet algorithme utilise deux positions mémoires par pixel pour stocker des valeurs de profondeur. La première pour évaluer l'arbre CSG, la seconde pour effectuer l'élimination des parties cachées en utilisant l'algorithme du Z-buffer.

Le traitement en chaque pixel (ils sont tous traités en parallèle) est le suivant:

Pour chaque objet primitif de l'arbre CSG,

- placer la profondeur de la face avant de l'objet (ou arrière si l'objet est soustrait) dans la première position mémoire. Comparer cette valeur avec les valeurs de profondeur avant et arrière de chaque objet primitif de l'arbre (l'arbre est parcouru en ordre post-fixé). Le résultat (état) est codé 1 si la face est intérieure à l'objet et 0 sinon. Les états sont stockés dans une pile mémoire de (N-1) bits (N étant la profondeur de l'arbre CSG). Le traitement d'un noeud non terminal consiste à combiner les résultats des deux bits du haut de la pile en fonction de l'opérateur (voir Figure 3.8). Les opérateurs correspondent aux traitements suivants: "ou inclusif" pour l'union, "et" pour l'intersection et "et avec le complémentaire" pour la différence.

- quand le parcours est terminé, si le dernier bit en haut de la pile est 1 pour un pixel donné, l'objet est classé visible et sa profondeur est placée dans la seconde position mémoire si elle est inférieure à celle stockée (opération du Z-buffer).



III.1.1.4.4 L'algorithme de Goldfeather

Cet algorithme [GoldFH86][GoldMT89] est lui aussi défini pour la machine Pixel-Planes 4. Il se différencie de celui de Jansen en n'utilisant pas de pile mémoire.

Cet algorithme nécessite un pré-traitement de l'arbre CSG pour le normaliser [Jansen87], c'est-à-dire le transformer en une union de groupes d'intersections. Un algorithme récursif de parcours de l'arbre et un ensemble de règles (voir Figure 3.9) réalisent cette normalisation sur l'ordinateur hôte. La figure 3.10 donne un exemple d'arbre normalisé.

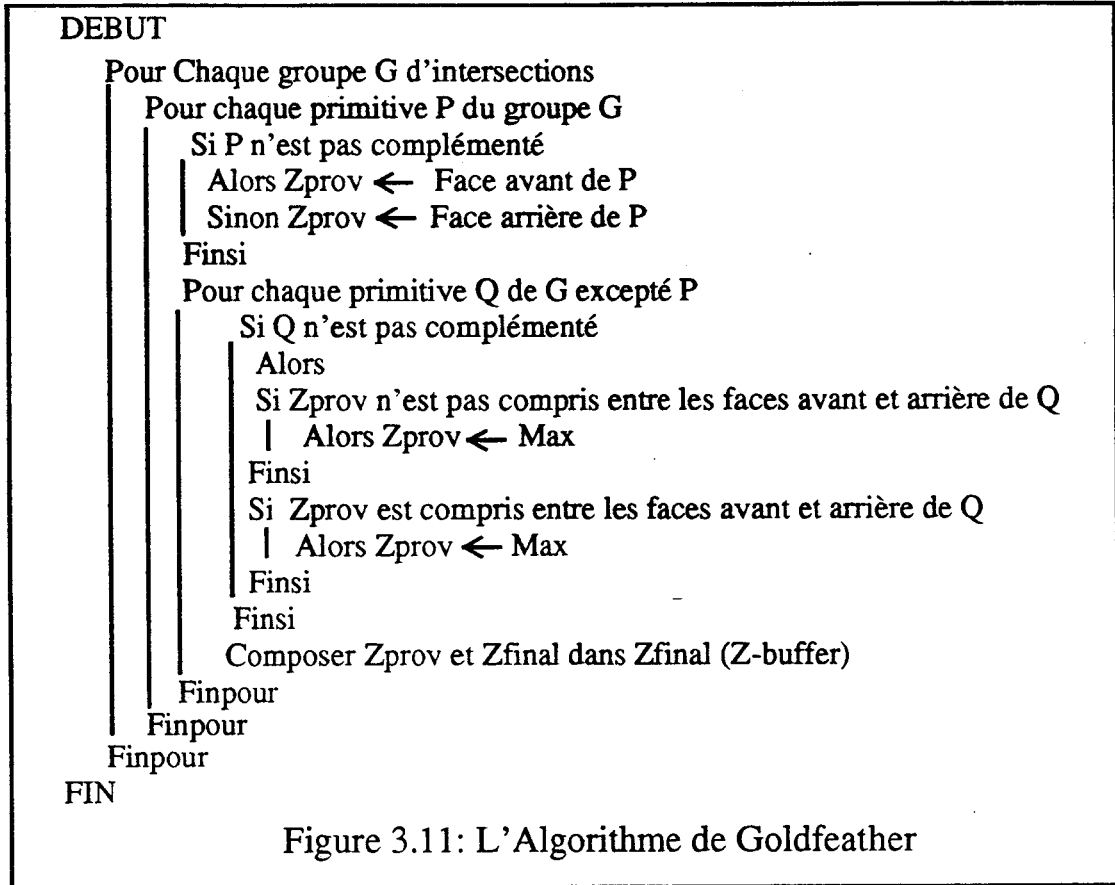
$$\begin{aligned}
 X - (Y \cup Z) &= (X - Y) - Z \\
 X \cap (Y \cup Z) &= (X \cap Y) \cup (X \cap Z) \\
 X - (Y \cap Z) &= (X - Y) \cup (X - Z) \\
 X \cap (Y \cap Z) &= (X \cap Y) \cap Z \\
 X - (Y - Z) &= (X - Y) \cup (X \cap Z) \\
 X \cap (Y - Z) &= (X \cap Y) - Z \\
 (X \cup Y) - Z &= (X - Z) \cup (Y - Z) \\
 (X \cup Y) \cap Z &= (X \cap Z) \cup (Y \cap Z) \\
 X - Y &= X \cap \bar{Y} \text{ où } \bar{Y} \text{ est le complémentaire de } Y
 \end{aligned}$$

Figure 3.9: Règles de normalisation

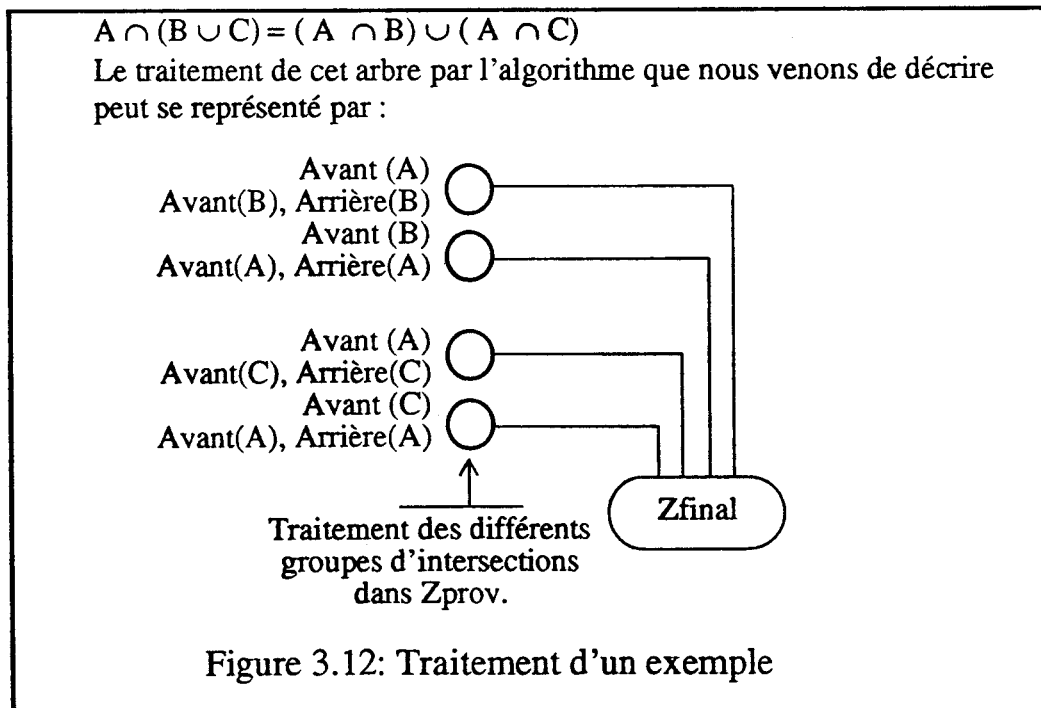
$$\begin{aligned}
 ((A \cup B) \cap C) - (D \cap E) &= (A \cap C - D - E) \cup (B \cap C - D - E) \\
 \text{Ce qui peut aussi se réécrire en utilisant la dernière règle:} \\
 ((A \cup B) \cap C) - (D \cap E) &= (A \cap C \cap \bar{D} \cap \bar{E}) \cup (B \cap C \cap \bar{D} \cap \bar{E})
 \end{aligned}$$

Figure 3.10: Un exemple de normalisation

Ensuite l'objet représenté par l'arbre normalisé est affiché en utilisant deux mémoires pour les profondeurs Z_{prov} et Z_{final} par pixel. L'algorithme d'affichage est représenté Figure 3.11. Cet algorithme est effectué par tous les processeurs-pixels simultanément.



Nous proposons, figure 3.12 , le traitement par cet algorithme de l'arbre que nous avons pris comme exemple pour l'algorithme de Jansen.



Cet algorithme nécessite moins de mémoire en chaque pixel que celui décrit précédemment, malheureusement la normalisation de l'arbre peut entraîner une augmentation considérable du nombre de primitives de l'arbre. En effet il y a de nombreuses duplications de primitives. Pour limiter ce phénomène on élague l'arbre en utilisant des boîtes englobantes des primitives. Un pré-traitement sur les boîtes englobantes permet d'ôter de l'arbre de nombreuses branches (par exemple si l'intersection de deux boîtes englobantes est vide l'intersection des deux objets l'est a fortiori). Elaguer augmente encore le temps de pré-traitement logiciel de l'arbre CSG.

III.1.1.5 Conclusion sur les arbres CSG

Les différents algorithmes que nous venons de présenter montrent qu'il est possible de traiter les arbres CSG après la conversion des objets en pixels. Malis cette opération a un coût élevé, elle nécessite des traitements multiples des primitives, beaucoup de place mémoire ou des pré-traitements de l'arbre CSG [RossVo89].

Les deux derniers algorithmes décrits sont développées spécifiquement pour une machine ayant un processeur par pixel et un système de diffusion des objets à tous ces processeurs. Ils montrent que pour une machine donnée, il est envisageable de trouver des solutions particulières pour afficher efficacement des arbres CSG.

III.1.2 les Eclairéments

Il est important de bien différencier les deux notions suivantes: les modèles d'éclairément et les techniques d'ombrage par interpolations. Elles sont souvent amalgamées car une même personne, Phong, a proposé une méthode d'interpolation et un modèle d'éclairément. Le choix d'un modèle d'éclairément consiste à déterminer les lois d'optique et les caractéristiques de la lumière prises en compte. Nous exposons ces notions dans le paragraphe suivant, nous choisissons le modèle qui nous semble bien adapté à une machine affichant en temps réel. Nous présentons ensuite les méthodes d'interpolation qui permettent à partir de certaines valeurs aux sommets d'une facette de calculer la valeur à afficher en tout pixel de l'écran où la facette est visible.

III.1.2.1 Les Modèles d'éclairément

Nous allons décrire le modèle d'éclairément le plus fréquemment utilisé.

Lorsqu'un objet reçoit de l'énergie lumineuse, il peut absorber ou réfléchir (transmettre) cette énergie. C'est cette lumière réfléchie qui rend l'objet visible. Les caractéristiques de la lumière réfléchie pour un objet dépendent de la direction et de la géométrie de la source lumineuse ainsi que des propriétés et de l'orientation de la surface de l'objet.

La lumière réfléchie peut se diviser en deux composantes: la Réflexion Diffuse et la Réflexion Spéculaire. La lumière réfléchie diffuse est émise régulièrement dans toutes les directions alors que la lumière spéculaire est émise autour de la direction symétrique de celle de la source lumineuse par rapport à la normale à la surface appelée direction spéculaire.

Afin que les objets non directement éclairés ne soient pas totalement noirs, il faut tenir

compte de l'éclairement diffusé par l'environnement. Pour cela on définit une composante de lumière ambiante.

$I(x)$ l'éclairement en un point x peut être exprimé par la formule: $I(x) = I_a(x) + I_d(x) + I_s(x)$.

Nous allons détailler l'expression des trois composantes $I_a(x)$, la lumière ambiante, $I_d(x)$, la lumière diffuse, et $I_s(x)$ la lumière spéculaire.

III.1.2.1.1 La lumière ambiante

$$I_a(x) = I_a * K_a$$

I_a est l'intensité de la lumière ambiante.

K_a est la constante de diffusion l'objet.

Cette composante ne dépend pas de la position du point x .

III.1.2.1.2 La composante diffuse

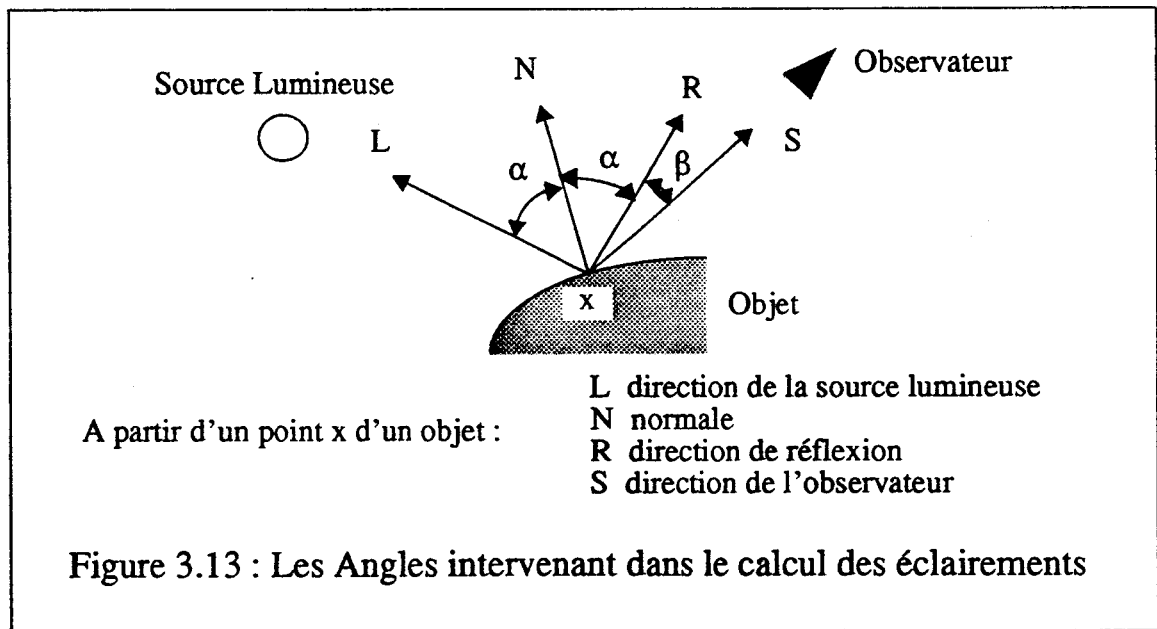
La loi de Lambert donne une bonne description de l'éclairement diffus.

$$I_d(x) = I_l * K_d * \cos(\alpha) ; 0 \leq \alpha \leq \pi/2$$

I_l est l'intensité incidente émise par la source ponctuelle.

K_d est une constante dépendant du matériau de l'objet.

α est l'angle entre N , la normale à la surface, et L , le vecteur reliant le point x à la source lumineuse (voir Figure 3.13).



III.1.2.1.3 La Composante spéculaire

La lumière spéculaire dépend de l'angle d'incidence, de la longueur d'onde et des caractéristiques du matériau.

Le modèle empirique le plus utilisé a été proposé par Bui Tuong Phong [Phong75]:

$$I_s(x) = \Pi * w(i, \lambda) * \cos^n(\beta) = \Pi * K_s * \cos^n(\beta)$$

Π est l'intensité incidente émise par la source ponctuelle.

$w(i, \lambda)$ est la fonction de réflectance qui dépend de l'angle d'incidence i et de la longueur d'onde λ . Vu les difficultés pour déterminer cette fonction on la remplace très souvent par une constante K_s fonction uniquement de l'objet.

n est un exposant qui détermine la distribution spatiale de la lumière spéculaire. n est grand pour les surface métalliques et plus petit pour les surfaces mates ($1 \leq n \leq 200$).

β est l'angle entre R le vecteur réfléchi et S le vecteur reliant le point x à l'observateur (voir Figure 3.13).

III.1.2.1.4 Résumé sur les éclairements

Les lois que nous venons de décrire ne tiennent pas compte des distances des objets à la source lumineuse et à l'observateur. Il est cependant évident qu'éclairer plus intensément les objets plus proches de l'observateur augmente le réalisme.

Pour prendre en compte ce phénomène, on multiplie les termes de réflexion diffuse et de réflexion spéculaire par une fonction $F(d)$, où d est la distance du point x à l'observateur. Expérimentalement on a déterminé que choisir $F(d) = 1/(d+k)$ avec k constant donne de bons résultats.

Le modèle d'éclairément que nous venons de décrire peut s'exprimer :

$$I(x) = I_a * K_a + \Pi * F(d) * \{ K_d * (N' \cdot L') + K_s * (R' \cdot S')^n \}$$

où N' , L' , R' et S' sont des vecteurs normalisés colinéaires respectivement à N , L , R , et S .

Si plusieurs sources lumineuses éclairent la scène, l'éclairage complet en un point est la somme de la composante ambiante avec une composante diffuse et une composante spéculaire par source lumineuse.

III.1.2.1.5 Autres Modèles

De nombreux autres modèles existent pour calculer l'éclairément:

Warn [Warn83] modifie le modèle précédent pour prendre en compte des sources lumineuses directionnelles, telles des spots.

Torrance et Sparrow [TorrSp67] proposent un modèle théorique pour la lumière réfléchi très proche de la réalité physique mais demandant de très gros volumes de calculs.

Whitted [Whitte80] propose un modèle plus complet prenant en compte les réfractions pour les objets transparents et les éclairages indirects transmis entre objets, en particulier les ombres portées.

Notre objectif est de définir un processeur temps réel, nous ne pouvons donc pas utiliser des modèles d'éclairément trop complexes. D'ailleurs nous avons choisi au début de ce travail de ne considérer que la méthode de rendu géométrique des objets; utiliser un modèle d'éclairément prenant en compte des transparences autres qu'en profondeur et les interactions lumineuses entre objets n'est alors plus envisageable.

III.1.2.2 Les Méthodes d'interpolation

Le plus souvent les objets graphiques sont modélisés par un ensemble de facettes planes juxtaposées. Si on utilise une normale constante pour chaque facette on obtient un résultat peu réaliste. Pour obtenir des résultats de meilleure qualité deux méthodes de lissage par interpolation ont été proposées par Gouraud [Gourau71] puis par Phong [Phong75].

III.1.2.2.1 La Méthode de Gouraud

Pour une facette donnée, le principe est le suivant:

- On détermine la normale à la surface en chaque sommet de la facette en faisant la moyenne des normales des faces dont ce point est un sommet.
- On calcule les intensités en tous les sommets.
- Ensuite les intensités en tout point de la face sont calculées par interpolation linéaire sur les valeurs aux sommets.

Les principales caractéristiques de cette méthode sont:

- 1) son faible coût, les calculs d'éclairement n'étant faits qu'aux sommets des facettes.
- 2) l'apparition d'effets visuels désagréables comme des bandes de Mach.
- 3) le très mauvais rendu des réflexions spéculaires. A tel point que cette méthode est le plus souvent utilisée sans réflexion spéculaire.

III.1.2.2.2 La méthode de Phong

La méthode de Phong se présente comme suit:

- On calcule les normales en tous les sommets de la face.
- On détermine la normale en tous les points de la face par interpolation bilinéaire à partir des normales aux sommets.
- On calcule l'éclairement en tous les points de la face.

Cette méthode résout les principaux problèmes de la méthode de Gouraud. En contrepartie les volumes de calcul sont bien supérieurs, le calcul d'éclairement étant fait en tout pixel.

Notons que les méthodes de Gouraud et de Phong ne s'utilisent que pour des facettes triangulaires, éventuellement des quadrilatères convexes si les valeurs au quatrième sommet dépendent linéairement de celles aux trois autres.

Lorsque ces méthodes sont utilisées en animation, il peut apparaître des discontinuités d'une image à la suivante.

De très nombreux travaux ont été réalisés pour améliorer ou accélérer cette méthode [BishWe86] [KuijB189] [Clauss89].

III.1.2.3 Conclusion sur l'éclairement

Pour obtenir des images convenablement réalistes il est indispensable, même pour une machine temps réel, de tenir compte de la composante spéculaire de la lumière. Ce choix nous impose d'utiliser la méthode d'interpolation de Phong pour visualiser les objets approchés à partir de facettes planes.

Les calculs d'éclairage doivent être effectués en tout pixel, c'est-à-dire après la conversion des objets en pixels. Le calcul d'éclairage en un point est très coûteux; des normalisations de vecteur, des produits scalaires, des produits, des sommes et des calculs d'exposant sont nécessaires. Il est donc très intéressant de reporter les calculs d'éclairage après les opérations inter-objets afin de ne réaliser ces calculs que pour les points visibles.

Afin de faire les calculs d'éclairage il est nécessaire de connaître en tous les pixels de tous les objets les composantes de la normale. Ces valeurs peuvent être calculées par la méthode d'interpolation de Phong pour les objets construits avec des facettes planes, mais il est tout à fait envisageable d'afficher d'autres objets sous réserve de pouvoir déterminer les composantes de la normale en tous les points.

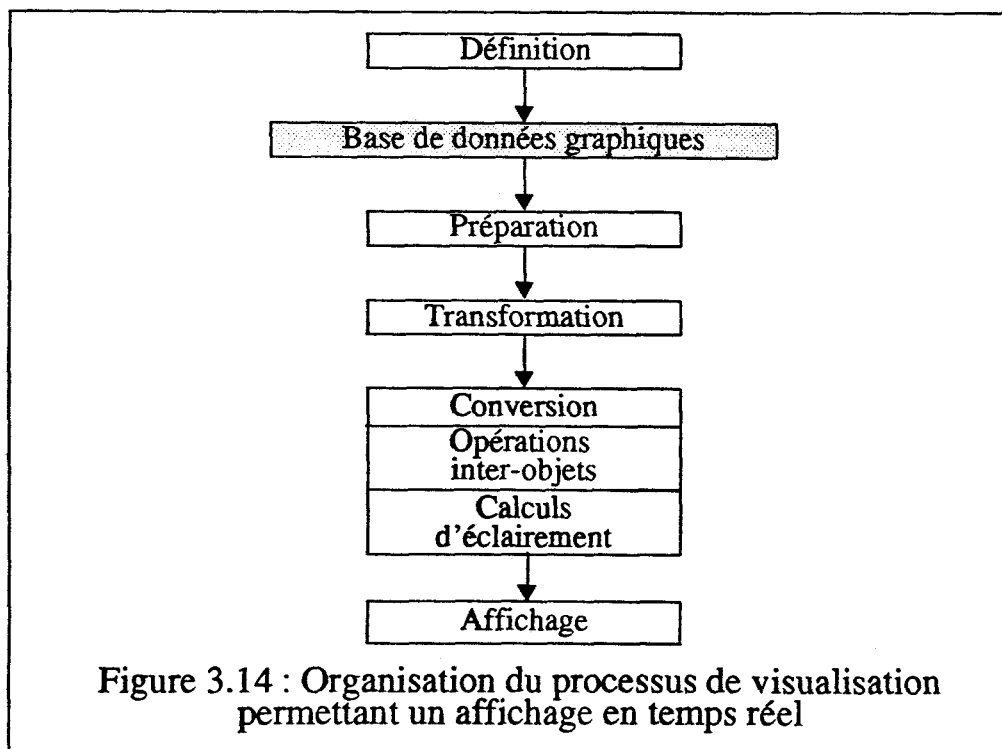
III.1.3 Organisation globale d'une machine d'affichage en temps réel

L'analyse que nous venons de faire sur les opérations inter-objets et les calculs d'éclairage peut se résumer comme suit :

Dans la mesure du possible il faut reporter après la conversion des objets en pixels les opérations inter-objets. Ceci dans le but de rendre les parties Préparation, Transformation et Conversion du processus de visualisation indépendantes d'un objet à l'autre. Il sera alors possible d'utiliser un grand nombre de processeurs en parallèle ou en pipe-line pour exécuter ces tâches.

Il est intéressant d'effectuer les calculs d'éclairage après les opérations inter-objets afin de n'effectuer qu'un calcul par pixel de l'écran. De plus cette solution permet d'utiliser d'autres méthodes de calcul que les interpolations sur des facettes planes.

La figure 3.14 représente l'organisation qui semble souhaitable pour le processus de visualisation d'une machine d'affichage temps réel.



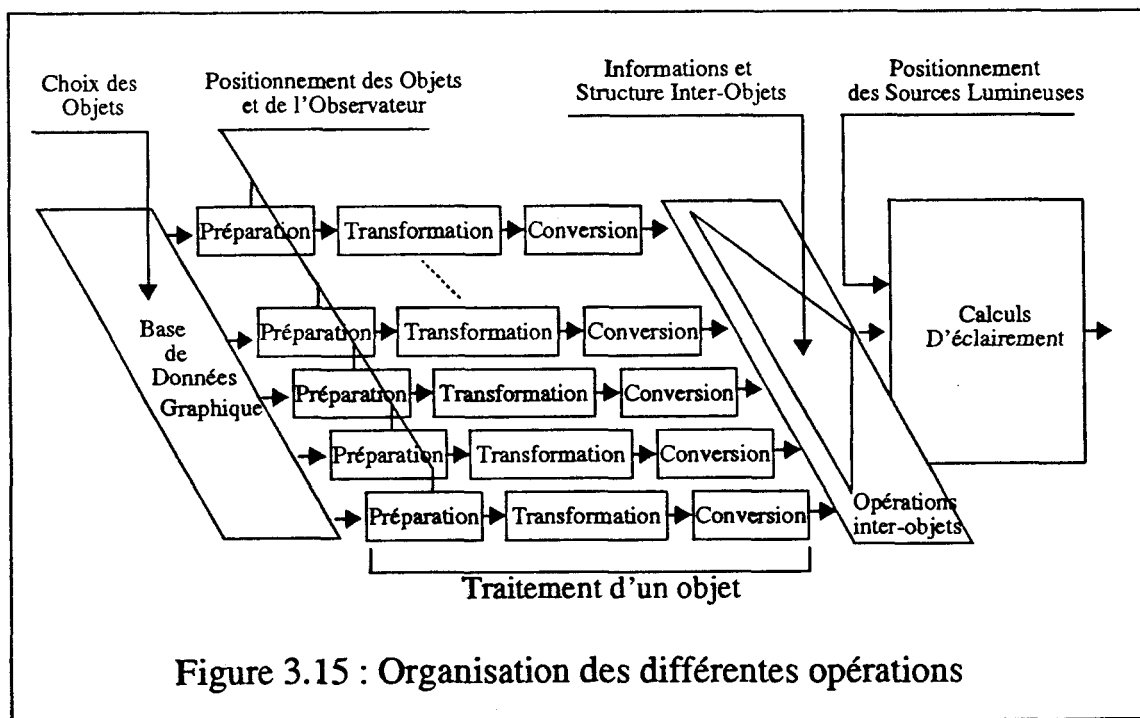
La figure 3.15 reprend l'organisation proposée, en mettant en évidence les traitements qui sont indépendants d'un objet à l'autre. Sont aussi exposés sur cette figure les stades du processus auxquels interviennent les différents paramètres qui définissent une image.

C'est la modification de ces paramètres qui matérialise l'animation. Nous pouvons par exemple constater que la modification des sources lumineuses n'a pas d'influence sur les parties Préparation, Transformation et Conversion du processus. Nous devons toutefois apporter la restriction suivante: si des ombres portées sont définies comme des objets, ces objets sont fonction des sources lumineuses dès la partie Préparation.

La coupure entre les parties logicielle et matérielle des machines d'affichage se situe entre les Transformations et la Conversion des objets en pixels.

Les premières parties du processus de visualisation jusqu'aux Transformations sont réalisées par logiciel sur un (ou plusieurs) processeur d'usage général appelé fréquemment le module pilote ou le module hôte. Un seul processeur spécifique, le Geometry Engine, propose une réalisation matérielle pour effectuer les Transformations.

Les dernières parties du processus de visualisation sont effectuées par une machine spécifique dédiée à cette seule tâche. Cette machine est souvent nommée Module Graphique.



III.1.4 La mémoire de trame

La mémoire de trame [GharGS89][CordMe90] ne figure pas dans les figures 3.14 et 3.15, la positionner dans le processus de visualisation est un problème délicat. En effet cette mémoire est un goulet d'étranglement dans le processus de visualisation.

La solution habituelle est de placer le résultat de la conversion d'un objet en pixels directement dans la mémoire de trame. Ceci nécessite un nombre d'accès à la mémoire considérable. En effet pour traiter un objet en un pixel, il faut acquérir la valeur stockée pour ce pixel dans la mémoire, faire un test sur la profondeur, et éventuellement effectuer un second accès à la mémoire pour placer la nouvelle valeur si l'objet est visible. De plus la mémoire de trame est consultée en continu, pixel par pixel, pour rafraîchir l'écran par le module Vidéo.

Pour une machine ayant une qualité de rendu convenable, le nombre de bits par pixel est de l'ordre d'une soixantaine (une vingtaine pour la profondeur, une dizaine par composante de la normale ou par couleur, plus celles nécessaires pour traiter d'autres phénomènes tels les transparences en Z).

Aujourd'hui, les mémoires les plus utilisées sont des composants vidéo RAM de 256 Kbits organisés en 64x4 avec accès parallèle 4 bits. Pour un cycle -lire, modifier, écrire- le temps moyen est de 375 ns en utilisant un accès rapide par bloc de 256 bits.

Les performances de ces mémoires de trame classiques sont largement insuffisantes pour une machine temps réel. Les solutions classiques pour augmenter la vitesse de travail sur la mémoire de trame se classent en deux familles :

- 1) utiliser plusieurs accès parallèles à la mémoire de trame.
- 2) Utiliser une mémoire de trame virtuelle très rapide (RAM statique) mais de plus petite taille. Les différentes parties de la mémoire de trame sont traitées successivement en utilisant cette mémoire virtuelle. Cette solution diminue de façon considérable le nombre d'accès à la mémoire de trame mais impose des pré-traitements sur les objets afin de travailler successivement sur différentes parties de la mémoire de trame.

La plupart des machines présentées dans le chapitre II exploite une de ces deux solutions pour supprimer le goulet d'étranglement que représente la mémoire de trame. La solution que nous préconisons est plus radicale, nous proposerons une architecture n'utilisant pas de mémoire de trame.

III.1.5 Les Phénomènes d'aliassage [Crow81]

L'aliassage est un phénomène parasite inévitable dû aux imprécisions de calcul et à l'affichage discret d'objets continus. Ces défauts d'aliassage se manifestent principalement de deux façons:

- Le clignotement des objets à la limite de la résolution de l'écran.
- Les marches d'escaliers (jaggies) sur les contours des objets. C'est à ce titre que l'on peut considérer l'aliassage comme un phénomène inter-objets.

On appelle anti-aliassage le procédé par lequel on remédie à ces phénomènes. Une solution est d'atténuer la couleur des pixels frontière entre les objets. Cette opération peut se réaliser avec deux techniques fort différentes:

Le Suréchantillonnage

On définit l'image avec une résolution supérieure au dispositif d'affichage. Pour un écran 1000x1000 il faut traiter au minimum une image 2000x2000 pour obtenir un suréchantillonnage en x et en y. Ensuite la valeur à afficher en chaque pixel est déterminée

en faisant une moyenne sur les valeurs en tous les points correspondant à ce pixel dans l'image suréchantillonnée.

Le Filtrage

Le filtrage est un traitement effectué juste avant la visualisation qui consiste à pondérer la valeur à afficher en un pixel en fonction des valeurs aux pixels voisins. Les outils pour effectuer les filtrages sont bien connus et couramment utilisés pour le traitement d'images.

Le suréchantillonnage est une méthode exacte alors que le filtrage n'est qu'approché, filtrer consiste à rendre "flous" les contours des objets. Malheureusement le coût du suréchantillonnage est très important.

III.1.6 La modélisation, les primitives visualisables

Sur les machines actuellement conçues, les primitives que peut visualiser le module graphique sont très limitées. La primitive la plus fréquemment utilisée est le triangle défini par les valeurs de profondeur et par soit les composantes de la normale soit les couleurs aux trois sommets. Les valeurs en tous les points intérieurs au triangle sont calculées par interpolation linéaire. Rares sont les machines proposant la visualisation directe de sphères sans effectuer de triangulation.

A notre connaissance aucune proposition pour afficher en utilisant des primitives de plus haut niveau n'est actuellement envisagée. Pourtant utiliser des facettes triangulaires présente de nombreux défauts:

1) les contours et intersections des objets courbes sont affichés comme une succession de segments donnant une image de faible qualité.

2) Cela impose de traiter une quantité de primitives graphiques considérable même pour une image construite à partir d'un faible nombre d'objets graphiques. Une sphère nécessite plusieurs dizaines (voire centaines) de facettes triangulaires.

Il nous semble donc intéressant de définir un module d'affichage capable de traiter des primitives plus complexes que la facette triangulaire. Bien évidemment la complexité matérielle du module graphique augmentera avec celle des primitives affichées. Utiliser des primitives graphiques de plus haut niveau permettrait d'obtenir des images de meilleure qualité tout en diminuant le nombre de primitives nécessaire à la définition d'une scène.

III.2 Les solutions matérielles pour réaliser un processeur d'affichage temps réel

L'analyse du chapitre II a mis en évidence que la conversion des objets en pixels est la partie la plus coûteuse du processus de visualisation. C'est cette tâche qu'il faut paralléliser prioritairement. Nous allons dans ce paragraphe discuter des différentes approches envisageables pour réaliser la conversion des objets en pixels. Nous en déduirons les structures possibles pour implanter le processus de visualisation de façon à afficher en temps réel.

III.2.1 Parallélisme Massif et Réalisation Matérielle

Nous pouvons affirmer que seule une solution matérielle est envisageable pour convertir les objets en pixels, vu les volumes de calcul à effectuer. Rappelons que convertir 1000 objets de 1000 pixels de surface moyenne nécessite un million d'opérations. Réaliser matériellement l'unité de conversion des objets en pixels est d'ailleurs l'option choisie sur presque toutes les machines existantes (voir Tableau 2.1). Actuellement sont développées des machines n'utilisant pas de solution matérielle [La Pixel Machine, TITAN] pour convertir les objets en pixels mais plusieurs processeurs d'usage général en parallèle. Ces machines n'ont pas l'ambition d'afficher en temps réel mais de proposer une vaste gamme d'algorithmes de visualisation avec des vitesses d'affichage supérieures à celles des machines mono-processeur.

Les solutions utilisant un nombre faible ou moyen de processeurs en parallèle ont montré leurs limites, voir les résultats des machines commerciales. Seules des solutions exploitant un parallélisme massif permettront d'afficher des scènes assez complexes en temps réel. C'est d'ailleurs la solution choisie pour toutes les machines récentes (SAGE, GPS-NVS et Pixel-Planes 5) qui annoncent des résultats de l'ordre du million de polygones par seconde.

Dans les paragraphes suivants, nous allons mettre en évidence les contraintes nouvelles induites par un parallélisme massif.

III.2.1.1 Régularité et Modularité

Le principe du parallélisme massif est de disposer d'un très grand nombre de processeurs de complexité assez faible. Pour offrir des coûts de réalisation raisonnables, il est souhaitable que les processeurs soient identiques. L'architecture de la machine doit être régulière pour utiliser de nombreux processeurs identiques.

Par ailleurs il est intéressant de choisir une architecture modulaire permettant d'augmenter les performances en ajoutant des processeurs.

III.2.1.2 Rendement et Ratio-coopération

Il faut admettre a priori que le Rendement des machines utilisant un nombre très important de processeurs sera faible. En effet utiliser en parallèle un très grand nombre de processeurs implique d'effectuer des opérations inutiles.

Quand on utilise un très grand nombre de processeurs il n'est plus envisageable de réaliser des liaisons directes entre tous les processeurs. Echanger des informations entre processeurs devient un opération d'un coût exorbitant sauf si chaque processeur n'est en relation qu'avec ses proches voisins.

Dans l'optique d'une machine temps réel utiliser une structure SIMD paraît raisonnable. En effet si les différents processus travaillent en mode MIMD, le Ratio-coopération deviendra très faible vu les mécanismes d'échanges qu'il faudra utiliser.

Deux approches massivement parallèles sont envisageables pour l'architecture du module effectuant la conversion des objets en pixels [Meriau84] :

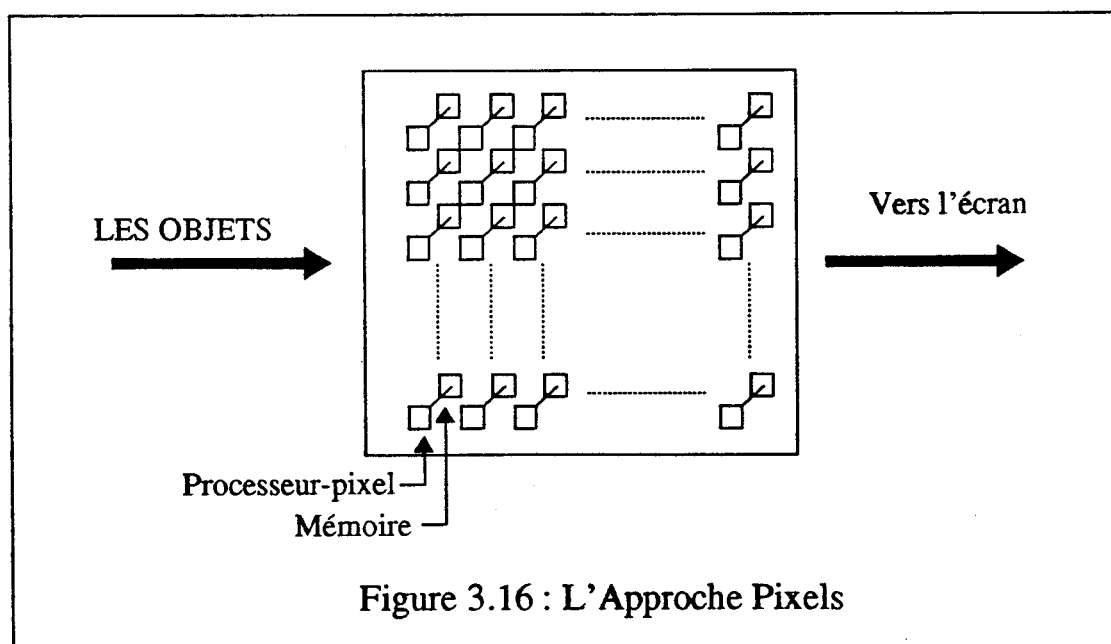
Effectuer un découpage de l'image, c'est-à-dire associer les processeurs aux pixels. On utilise un processeur par pixel, exploitant un parallélisme pixel massif.

Effectuer un découpage de la scène à visualiser, c'est-à-dire associer un processeur à chaque objet graphique. On exploite alors un parallélisme objet massif.

III.2.2 L'Approche Pixels

Utiliser un parallélisme massif signifie avec une approche pixels affecter un processeur à chaque pixel (voir Figure 3.16). La mémoire de trame est alors totalement répartie sur le réseau de processeurs, chaque processeur-pixel a sa propre mémoire. Il faut dans ce cas de figure définir un processeur Vidéo particulier ayant accès aux différentes mémoires.

La difficulté principale est alors de fournir les objets aux différents processeur-pixels.



Trois solutions, correspondant chacune à une architecture de machine, sont possibles pour fournir les objets aux processeurs : la Diffusion, le Réseau multi-pipeline et la

Propagation. Leprêtre a fait [Lepret89] une analyse de ces différentes solutions afin de déterminer laquelle est la mieux adaptée pour implanter des algorithmes de tracé de segments, de tracé de cercles et de remplissage. L'exposé que nous allons faire sur l'approche pixels utilise certains de ses résultats.

III.2.2.1 La Diffusion

Les différents objets sont envoyés à tous les pixels, les processeurs fonctionnent en parallèle SIMD et sont tous indépendants (voir Figure 3.17).

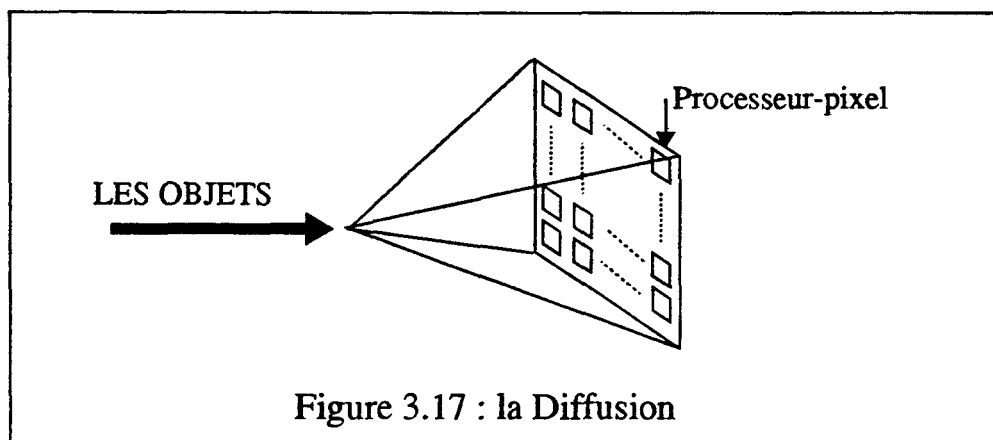


Figure 3.17 : la Diffusion

Tous les processeurs-pixels traitent simultanément le même objet. Le Rendement de l'opération de conversion est donc extrêmement faible: pour un objet donné il est égal au quotient du nombre de pixels où l'objet est présent sur le nombre total de pixels de l'écran.

Par contre le Ratio-coopération est proche de 1, il n'y a aucune communication entre les différents processeurs-pixels ni aucun prétraitement sur les objets.

Cette solution exploite un parallélisme pixels important mais ne permet aucun parallélisme objet. Les objets sont traités séquentiellement.

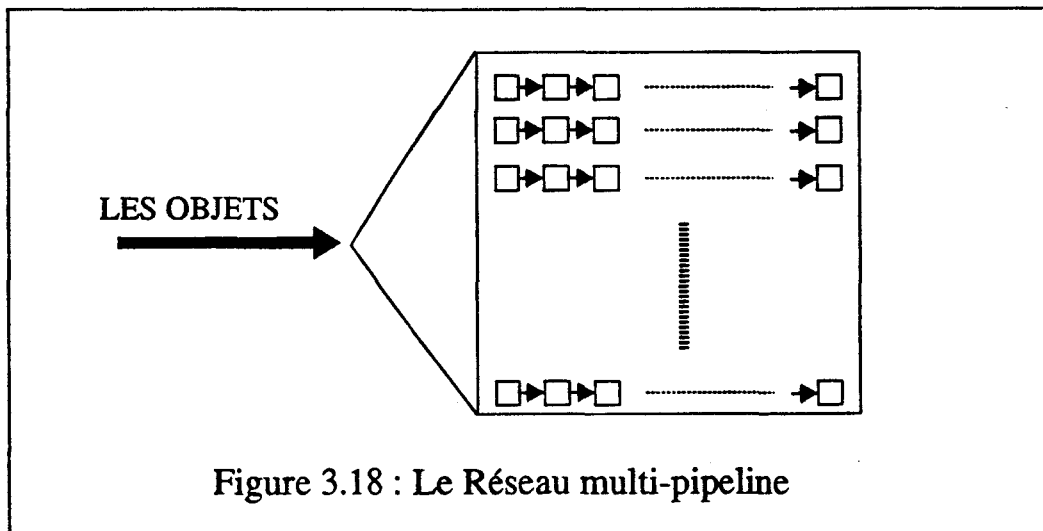
La machine Pixel-Planes 4 entre dans cette catégorie en considérant que tous les objets sont codés avec des expressions linéaires et que les arbres d'additionneurs sont un mécanisme de diffusion. Un pipeline objets est alors utilisé dans les arbres d'additionneurs.

III.2.2.2 Le Réseau multi-pipeline

Tous les processeurs pixels ont deux liens de communication, un lui permettant de recevoir du processeur de gauche et un lui permettant d'envoyer au processeur de droite.

Les objets circulent de la gauche vers la droite horizontalement. Toutes les entrées se font par le bord gauche. Chaque processeur reçoit les objets de son voisin de gauche et les envoie à celui de droite.

Les objets sont prédécoupés en segments horizontaux. Cette solution permet une implantation efficace des algorithmes où les valeurs de profondeur et celles à afficher sont calculées par interpolations.



Avec cette architecture le mécanisme de fonctionnement le moins coûteux est de faire travailler de façon synchrone tous les processeurs-pixels d'une même colonne.

Cette solution (voir Figure 3.18) permet d'obtenir un parallélisme pixels égal au nombre de lignes de l'écran de visualisation et un pipeline objets sur la chaîne de processeurs traitant une ligne-écran. Il est envisageable de traiter en parallèle des objets appartenant à des bandes horizontales distinctes de l'écran. Ceci nécessite un prétraitement des objets, et ce parallélisme n'est pas exploitable dans le cas défavorable où tous les objets de la scène sont présents sur une ligne donnée.

Comme la diffusion, le réseau multi-pipeline a un rendement très faible, toutefois un peu meilleur si plusieurs objets sont traités en parallèle. Le ratio-coopération reste important mais sera diminué du temps passé à trier les objets pour les traiter ensuite en parallèle.

Il est probablement envisageable de construire une mémoire intelligente de ce type en utilisant des composants SAGE.

III.2.2.3 La Propagation Cellulaire

On peut accéder directement à certains processeurs pixels à partir du hôte. La conversion d'un objet en pixels se fait alors en envoyant l'objet à un des processeurs pixels. Ce processeur propage l'objet à ses voisins qui déterminent si l'objet est visible en leur pixel et propage l'objet à ses voisins. Cette solution nécessite que chaque processeur pixel ait des liens de communication avec tous ses voisins (4 ou 8 suivant la connexité).

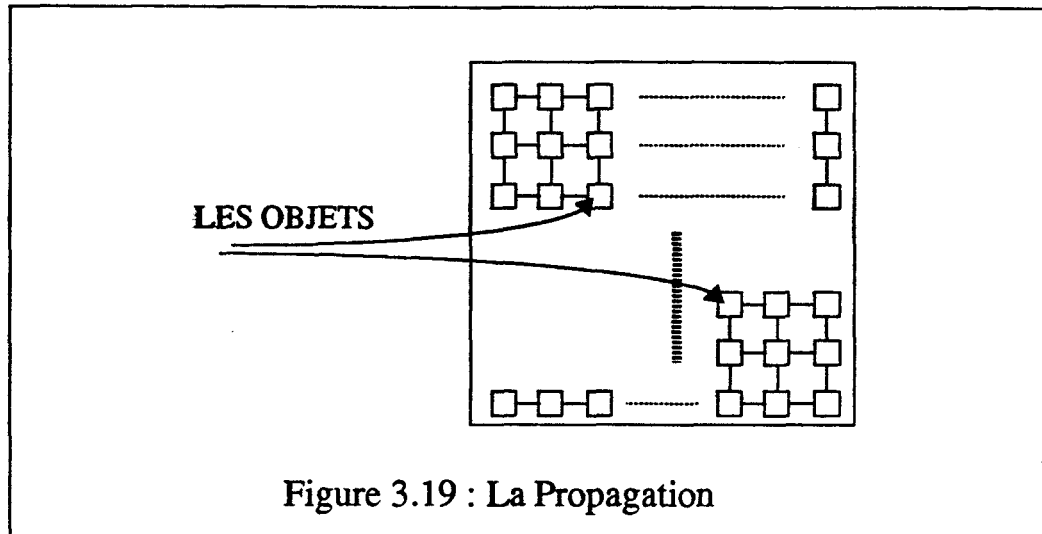


Figure 3.19 : La Propagation

Cette solution peut se décomposer en deux sous-solutions très différentes:

1) Il n'existe qu'un seul point d'accès au réseau, toute la propagation se fait à partir de ce point. En considérant le point germe au centre du réseau il faut $N / 2$ cycles avant que l'information ait atteint le bord du réseau, N étant le nombre de pixels sur une ligne ou une colonne de l'écran.

Cette solution permet d'exploiter un parallélisme pixels important et un pipeline objets. Mais elle n'autorise aucun parallélisme objets.

2) Il existe plusieurs points d'accès au réseau (voir Figure 3.19), ceci permet d'utiliser un parallélisme objets pour des objets étant sur des zones distinctes de l'écran. Cette solution impose de disposer d'un mécanisme pour fournir les objets aux différents points germe et de gérer dans le réseau des conflits entre les traitements des différents objets. Il n'est pas évident que la propagation avec plusieurs point d'accès soit plus efficace que celle avec un seul point d'accès, en effet chaque processeur-pixel effectue des tâches plus complexes.

La propagation impose au différents processeurs-pixels de travailler en MIMD, Le Ratio-coopération est alors très faible car les mécanismes contrôlant les communications sont coûteux. Le Rendement est du même ordre de grandeur qu'en utilisant la diffusion ou le réseau multi-pipeline.

A notre connaissance aucune machine ayant pour objectif l'affichage d'image de synthèse n'utilise cette solution, même partiellement.

III.2.2.4 Résumé

Si les machines utilisant un processeur par pixel exploitent naturellement un parallélisme pixel très important, elles n'autorisent qu'un parallélisme objets très faible qui de plus devient nul pour les cas défavorables. Par contre elles permettent d'utiliser d'importants effets pipeline sur les objets.

Les machines utilisant un processeur par pixel ont pour caractéristiques principales:

1) L'uniformité: quelle que soit l'architecture choisie, l'ensemble des processeurs-pixels a une structure très régulière, ce qui permet une implémentation efficace en VLSI.

2) La rigidité : le réseau de processeurs est conçu pour traiter des types précis de données. Tout changement implique de reconcevoir l'intégralité des processeurs.

3) Pour construire une machine travaillant en temps réel, cette solution présente un inconvénient majeur: le temps d'exécution des tâches sur chaque processeur est fonction du nombre d'objets intervenant en son pixel. Il est donc difficile de garantir a priori l'obtention du temps réel. En contrepartie la complexité matérielle est constante quel que soit le nombre d'objets.

III.2.3 L'Approche Objets

III.2.3.1 Présentation

Utiliser un parallélisme massif avec une approche objets impose d'affecter un processeur par objet graphique (voir Figure 3.20).

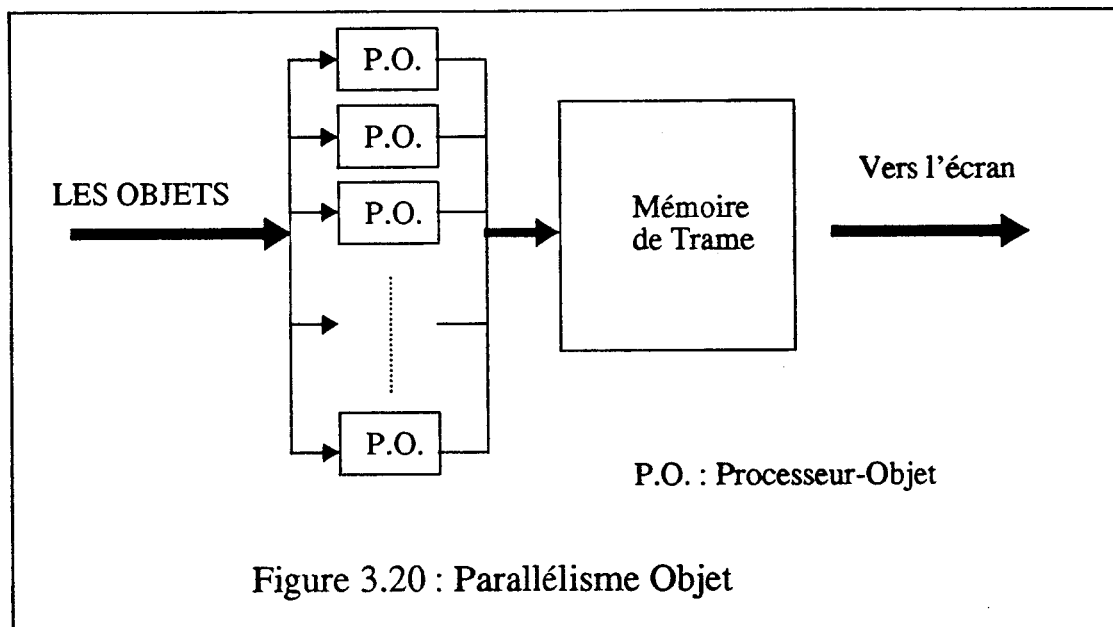


Figure 3.20 : Parallélisme Objet

Réaliser une machine sur ce schéma n'est pas envisageable. En effet il est impossible de définir une machine comprenant un très grand nombre de processeurs (plusieurs milliers) travaillant tous sur une même mémoire de trame. Il faut proposer d'autres solutions pour regrouper les résultats fournis par les différents processeurs objets. Nous appelons décideur le mécanisme regroupant les résultats (voir Figure 3.21).

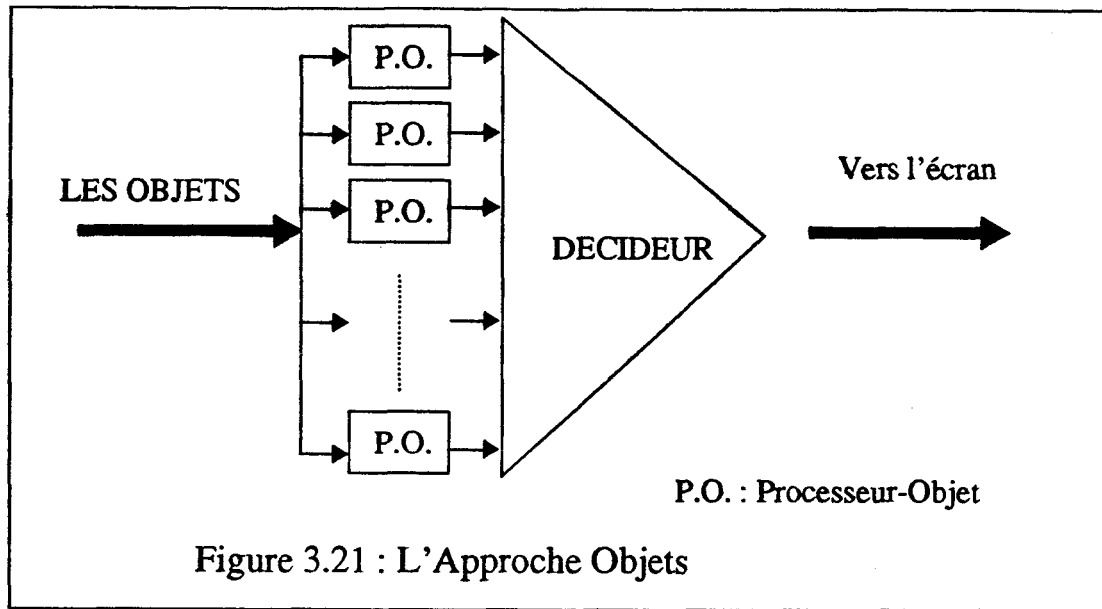


Figure 3.21 : L'Approche Objets

La solution pour se passer de la mémoire de trame est de faire travailler les différents processeurs objets au rythme du balayage de trame. Les opérations inter-objets sont alors réalisées dans le décideur.

Les pixels sont alors traités en pipeline. Ainsi la mémoire de trame devient inutile si les traitements inter-objets sont réalisés au rythme du balayage écran.

Avec cette configuration, les différents processeurs objets fournissent en tout pixel, au rythme du balayage écran, les caractéristiques de l'objet qu'il traite. Le décideur réalise les opérations inter-objets c'est-à-dire au minimum l'élimination des parties cachées.

III.2.3.2 Une solution

L'approche objets est nettement moins souvent exploitée que l'approche pixel. De nombreux choix restent à faire pour définir une machine utilisant cette approche. En particulier il faudra: déterminer comment réaliser le décideur, choisir les objets primitifs que peut traiter un processeur objet. A notre connaissance une seule machine récente a été construite avec une telle structure: GSP-NVS que nous avons décrit dans le chapitre II. Résumons ces principales caractéristiques:

- Elle est composée de processeurs réalisés en VLSI traitant chacun un triangle en 3 dimensions et fonctionnant en pipe-line.
- L'élimination des parties cachées est réalisée par l'algorithme du Z-buffer en pipeline. Chaque processeur reçoit de son prédécesseur dans le pipeline la valeur à afficher pour chaque pixel. Si pour un pixel le triangle qu'il traite a une profondeur inférieure, le processeur remplace la valeur par celle de son triangle sinon il fait suivre. Le décideur est ainsi réparti dans les processeurs objets.
- Le chargement des triangles dans les processeurs libres se fait au cours de la visualisation au début de chaque ligne, ceci afin de pouvoir utiliser un même processeur pour traiter plusieurs triangles verticalement disjoints.

Cette machine présente, dans l'optique d'obtenir une visualisation temps réel, plusieurs défauts:

i) il se peut que sur une ligne donnée il y ait plus de triangles que de processeurs objets. Les triangles ne sont alors pas tous traités en une passe et le temps réel ne peut être atteint.

ii) elle nécessite un tri des triangles en fonction de leur point d'ordonnée maximale pour chaque image, ce qui est un travail coûteux. A partir d'un certain nombre de triangles dans la scène le processeur hôte ne pourra effectuer ce travail en temps réel.

Ce dernier défaut disparaît si on utilise un processeur par triangle pour toute l'image. Cette solution nécessite de disposer de beaucoup plus de processeurs, c'est toutefois la seule solution qui permet d'obtenir un affichage temps réel.

Notons que dans le cas défavorable où tous les triangles sont présents sur une ligne donnée, un processeur objet traite un et un seul triangle pour une image donnée.

III.2.3.3 Résumé

Les machines utilisant une approche objets exploitent un parallélisme objets très important (égal au nombre de processeurs objets) mais ne permettent d'utiliser aucun parallélisme pixels. Par contre il est possible d'exploiter des effets pipeline sur les pixels.

Les principales caractéristiques de l'approche objets sont:

1) d'offrir un temps de traitement constant indépendamment du nombre d'objets à visualiser. Bien évidemment le nombre d'objets visualisables dans une scène donnée est fonction du nombre de processeurs objets disponibles. En utilisant des composants VLSI, il est aujourd'hui envisageable de construire une machine comprenant un très grand nombre de processeurs objets.

2) de se passer complètement de la mémoire de trame qui est le principal goulet d'étranglement dans le processus de visualisation.

3) de pouvoir visualiser des primitives graphiques de plusieurs types en utilisant des processeurs objets différents.

III.2.4 Comparaisons des deux approches

III.2.4.1 Faisabilité

A priori il est plus facile de réaliser une machine pixels. En effet l'ensemble des processeurs pixels sont tous strictement identiques (sauf éventuellement ceux des extrémités du réseau) et la structure générale est très régulière.

L'approche objet a pour qualité d'être modulaire; la puissance de la machine augmente quand le nombre de processeurs croît. Cette approche est plus souple que l'approche pixels: il est possible de définir des processeurs objets de plusieurs types pour traiter différentes primitives graphiques. En contrepartie définir des processeurs objets de plusieurs types diminue la régularité de la machine.

III.2.4.2 Performances

Avec l'approche pixels le temps pour traiter une scène est linéairement fonction du nombre d'objets graphiques à traiter. Cette approche n'impose pas de limite au nombre de primitives pour une scène donnée.

Avec une approche objets, le temps réel est obtenu naturellement si les processeurs objets et le décideur travaillent au rythme du balayage de trame. Par contre les performances en nombre d'objets sont fonction du nombre de processeurs objets inclus dans la machine.

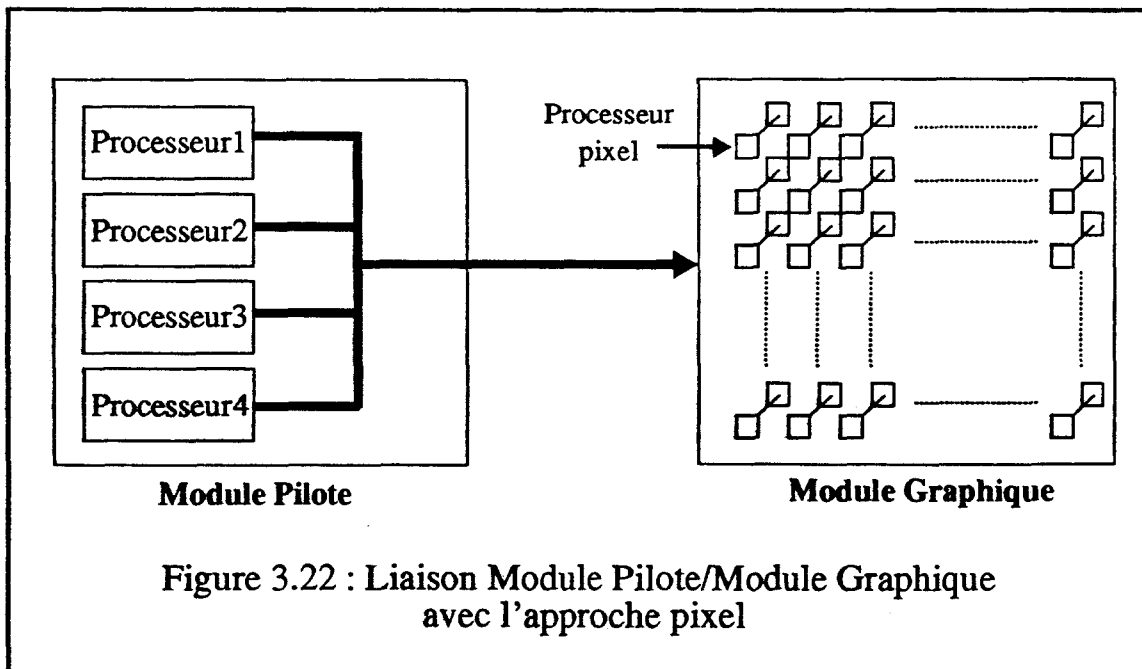
Il est fort difficile de comparer les performances des deux approches. Pour faire une comparaison significative, il faudrait simuler totalement un réseau de processeurs pixels pour déterminer la quantité de portes logiques qu'il comprend et estimer le nombre d'objets qu'il peut visualiser en temps réel. Ensuite il faudrait calculer combien de processeurs objets, avec le décideur correspondant, sont réalisables avec le même nombre de portes.

III.2.4.3 Liaison module Pilote / module Graphique

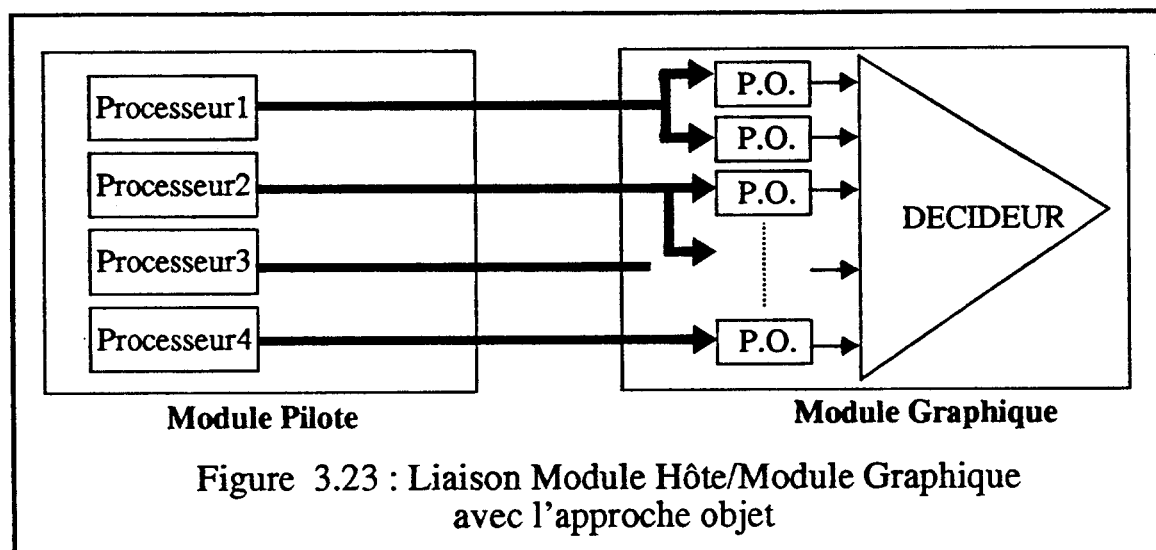
Un module graphique exploitant un parallélisme massif aura des performances importantes. Le module pilote et les liaisons devront avoir des performances suffisantes pour l'alimenter.

Dans les conclusions du chapitre II nous avons mis en évidence qu'il faudra utiliser un module hôte multiprocesseur pour exploiter correctement un module graphique performant. Chaque processeur traite une partie des objets de la scène. Se pose alors le problème des liaisons entre les processeurs du hôte et le module graphique.

Un objet donné peut être visible en n'importe quel pixel de l'image, donc avec une approche pixel une seule liaison entre le module hôte et le module graphique est envisageable (voir Figure 3.22). Cette liaison représente un goulet d'étranglement entre les deux parties de la machine d'affichage.



Avec l'approche objet on peut établir des liaisons entre chaque processeur du module hôte et une partie des processeurs objets du module graphique (voir Figure 3.23). Avec cette approche il n'y a donc pas de problème d'alimentation du module graphique.



III.2.4.4 Résumé

Nous résumons dans le Tableau 3.1 les éléments de comparaison entre les deux approches.

	Approche Pixels	Approche Objets
Performances temporelles	Fonction du nombre d'objets présent dans la scène à afficher	Constante, tous les processeurs travaillent au rythme du balayage de trame
Performances en nombre d'Objets	Pas de limite sinon le temps Traite un type d'objets donné	Directement fonction du nombre de processeurs objets disponibles
Les Objets traités	Considérer un nouveau type d'objet impose de redéfinir tous les processeurs pixels	Différents processeurs peuvent traiter différents types d'objets
Alimentation du processeur en objets	Une liaison seulement est possible entre le module pilote et le module graphique	Plusieurs liens sont possible entre le module pilote et le module graphique

TABLEAU 3.1 : Comparaison des deux Approches

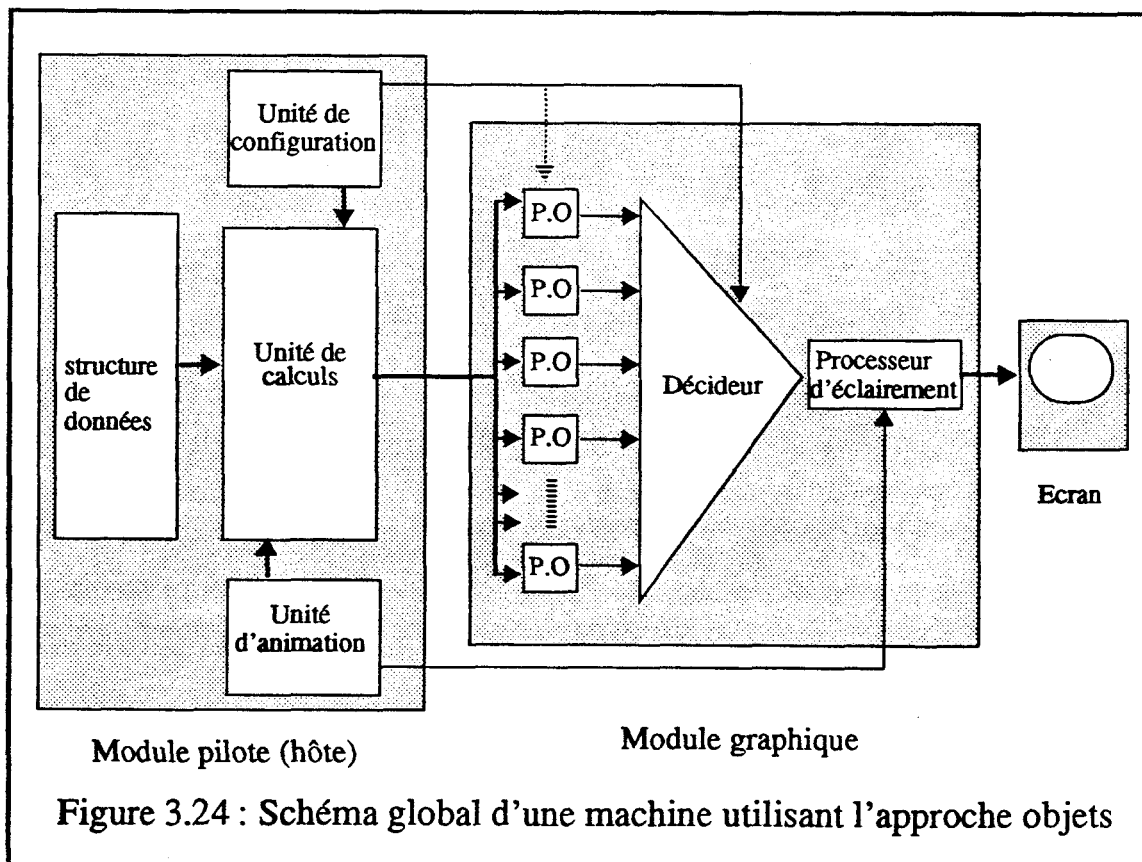
III.3 Notre choix: L'Approche Objets

Le raisonnement que nous venons de conduire nous incite à choisir une approche objets pour construire une machine d'affichage temps réel.

Les principaux critères justifiant ce choix sont:

- On est certain avec cette approche de réaliser une machine temps réel, même si ce n'est que pour un nombre limité d'objets graphiques.
- Cette solution est modulaire: on peut réaliser une première version avec un nombre restreint de processeurs objets et ensuite construire une machine plus performante en multipliant le nombre de processeurs.
- Cette approche permet de contourner le goulet d'étranglement existant entre le Module Pilote et le Module Graphique avec une approche pixels.
- Il est aujourd'hui envisageable de définir, avec les logiciels de CAO, des composants intégrés spécialisés travaillant à une vitesse suffisante pour faire des calculs en chaque point au rythme du balayage écran. La mémoire de trame, principal goulet d'étranglement des modules graphiques actuels peut alors être supprimée.

A partir de l'organisation du processus de visualisation proposée au paragraphe III.1.3 nous pouvons définir l'architecture la mieux adaptée pour réaliser une machine à découpage objet travaillant en temps réel (voir Figure 3.24).



Explicitons les tâches effectuées par les différentes unités de la machine présentée sur la figure 3.24:

III.3.1 Le module pilote

On peut fonctionnellement décomposer le travail du module pilote en trois unités:

- Une unité de configuration qui définit la structure du décideur et éventuellement des processeurs objets si ceux-ci sont programmables pour traiter différentes primitives graphiques. Elle effectue aussi l'affectation des primitives graphiques aux processeurs objets.

- Une unité d'animation qui fournit au processeur d'éclairage les positions et caractéristiques des sources lumineuses. Elle fournit à l'unité de calculs la position de l'observateur et les déplacements des objets. Cette unité envoie un flot de valeurs par image à visualiser, c'est-à-dire 25 fois par seconde pour afficher en temps réel.

Une unité de calculs qui détermine et envoie les données nécessaires à chaque processeur objet pour afficher la primitive graphique qu'il traite. Cette unité doit effectuer tous les calculs dans le temps imparti à l'affichage d'une image pour atteindre le temps réel.

III.3.2 Le module graphique

Le module graphique est composé d'un très grand nombre de processeurs objets qui traite chacun une primitive graphique. Les processeurs objets travaillent au rythme du balayage écran. Les différents processeurs objets alimentent le décideur qui réalise les opérations inter-objets. Un processeur d'éclairage effectue les calculs d'éclairage.

Toutes les unités du module graphique fonctionnant au rythme du balayage écran, le module ne comprend pas de mémoire de trame.

III.3.3 Conclusion du chapitre

Dans la première partie de ce chapitre nous avons présenté différents algorithmes et précisé lesquels étaient utilisables dans une machine temps réel, c'est-à-dire parallélisables de façon massive. Nous en avons déduit une organisation du processus de visualisation. Nous avons étudié ensuite les différentes solutions architecturales permettant de construire un module d'affichage temps réel. Une solution intéressante est de définir un module graphique orienté objet exploitant un parallélisme massif. Dans le chapitre IV nous décrirons I.M.O.G.E.N.E qui est une machine respectant les contraintes mises en évidence dans ce chapitre.

IV I.M.O.G.E.N.E

Dans ce dernier chapitre nous allons décrire la machine I.M.O.G.E.N.E. Nous commençons par décrire les Processeurs Élémentaires et les Processeurs Objets, ensuite nous présentons le Décideur. Nous terminons en étudiant l'ensemble du Module Graphique et le Processeur Hôte.

IV.1 Les Processeurs Élémentaires

IV.1.1 Introduction

Le choix des Processeurs Objets est très délicat car il détermine d'une part le type de scènes visualisables, et d'autre part la faisabilité d'une intégration VLSI. En effet, si nous choisissons un objet de base trop simple, le nombre de processeurs nécessaire à une modélisation correcte sera très élevé; si par contre il est trop complexe, l'intégration VLSI peut être compromise. Nous avons montré dans le chapitre précédent qu'il serait toutefois intéressant de définir des primitives de plus haut niveau que la facette triangulaire.

Il est souhaitable de visualiser les objets sous une forme la plus proche possible de celle sous laquelle ils sont dans la structure de données afin de limiter le travail du module hôte. Nous pensons en particulier aux objets définis par révolution autour d'un axe d'une courbe comprise dans un plan ou construits à l'aide de surfaces définies par des expressions polynomiales comme les fonctions Splines.

Les constatations que nous venons de faire nous conduisent à choisir un objet de base relativement complexe, et donc des Processeurs Objets complexes mais composés de plusieurs Processeurs Élémentaires facilement intégrables. Les différents Processeurs Élémentaires fonctionnent en parallèle de manière synchrone.

Quelque que soit le Processeur Objet choisi, celui-ci doit signaler au décideur si l'objet qu'il traite est présent ou non au pixel courant. Si l'objet est présent, il fournit au décideur la profondeur avant de l'objet, éventuellement la profondeur arrière et les composantes de la normale à l'objet en ce point.

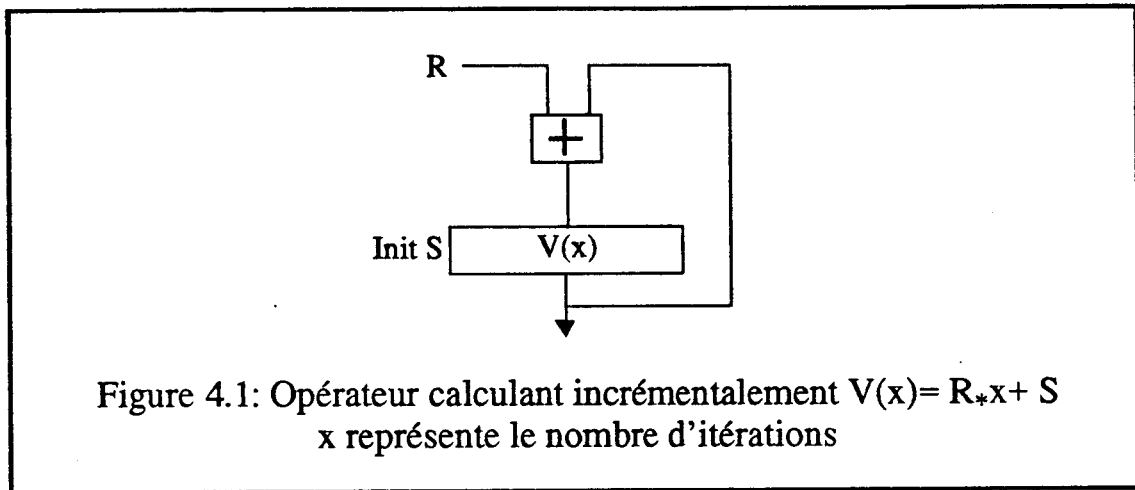
IV.1.2 Définition du Processeur Élémentaire

Nous avons précédemment montré qu'avec un découpage objet, les Processeurs Objets doivent fournir des valeurs en tout pixel de l'écran. Les Processeurs Élémentaires qui les composent doivent avoir les mêmes caractéristiques. De plus ces valeurs doivent être fournies au rythme du balayage écran (après la valeur au point (x,y) , il faut fournir celle au point $(x+1,y)$ puis $(x+2,y)$...). Une solution simple est de définir le Processeur Élémentaire comme une entité calculant une valeur $V(x,y)=Ax^2+By^2+Cxy+Dx+Ey+F$ ou $F(x,y)=Ax+By+C$, $A,B...F$ étant des constantes entières et (x,y) les coordonnées d'un pixel, cette valeur pouvant être calculée incrémentalement au rythme du balayage écran, pour chaque pixel de coordonnées (x,y) .

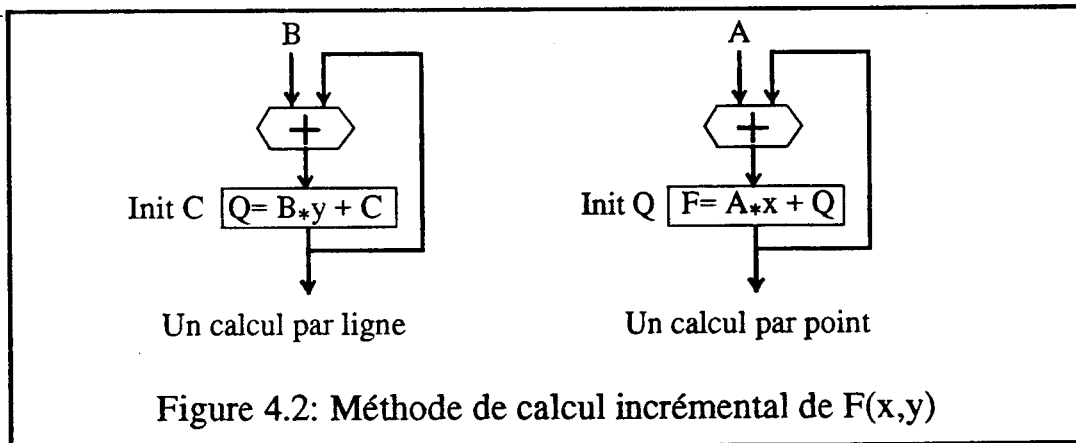
Nous allons détailler la méthode incrémentale de calcul de $F(x,y)$ et de $V(x,y)$. Nous appellerons dans la suite PE1 le Processeur Élémentaire calculant F , et PE2 celui calculant V .

IV.1.3 Le Processeur Élémentaire de premier ordre

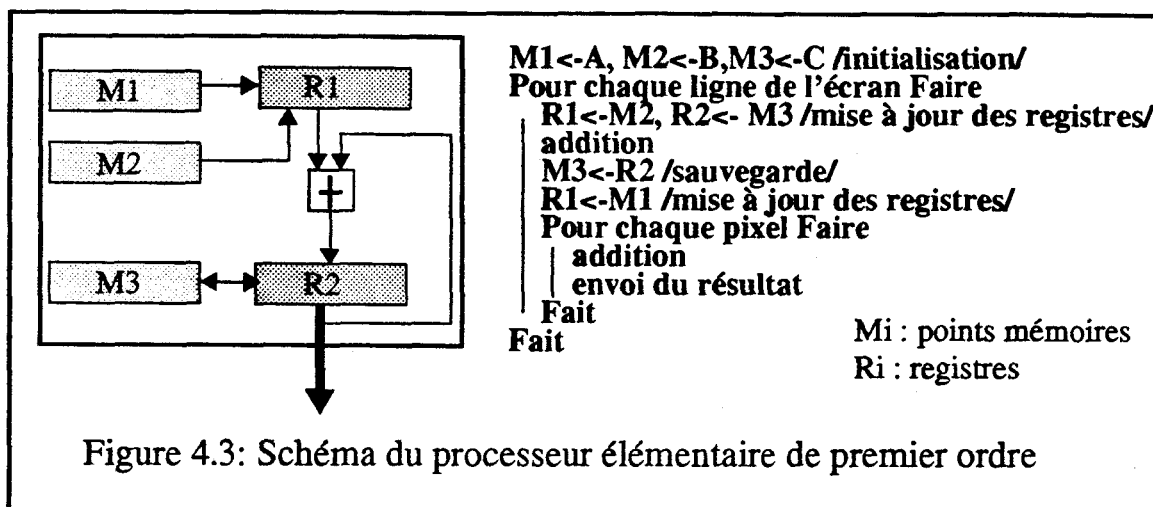
L'opérateur défini figure 4.1 permet de calculer une valeur $V(x) = Rx + S$ où x est le nombre d'itérations. Ceci correspond physiquement au nombre de tops de calcul.



En posant $Q(y) = By + C$ on obtient $F(x,y) = F'(x) = Ax + Q$. Le calcul de $Q(y)$ s'effectue au début de chaque ligne, celui de $F'(x)$ en chaque pixel en utilisant deux opérateurs du type de celui de la figure 4.1 (voir Figure 4.2).



Une structure globale unique utilisant deux registres et trois points mémoires permet de calculer $F(x,y)$ à la condition de mémoriser la valeur $Q(y)$ (voir Figure 4.3).

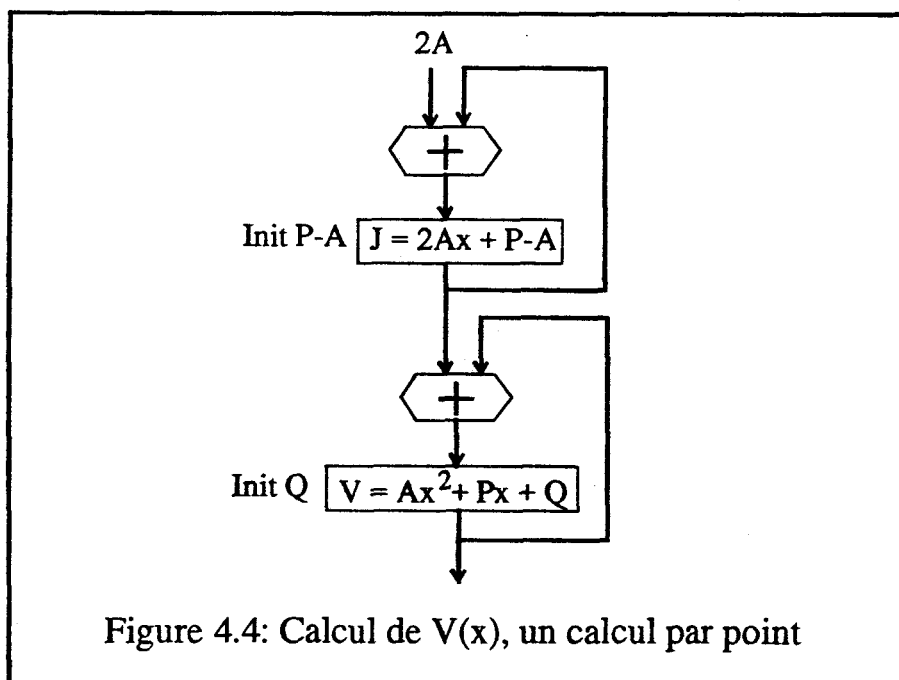


IV.1.4 Le Processeur Élémentaire de second ordre

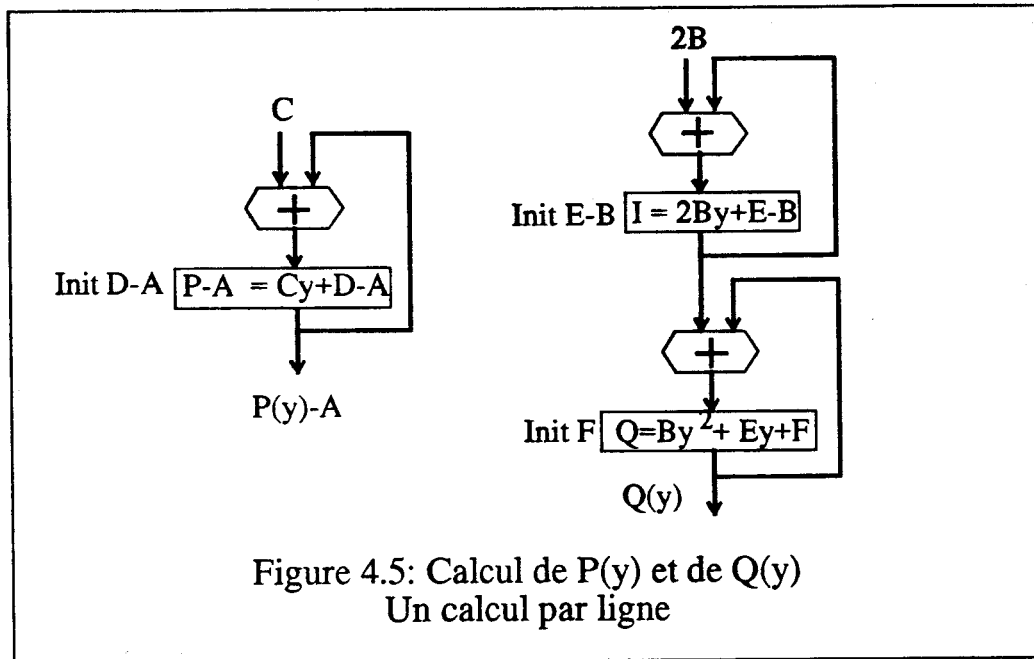
En posant $P(y) = Cy + D$ et $Q(y) = By^2 + Ey + F$, on obtient $V(x, y) = V_1(x) = Ax^2 + Px + Q$. La méthode de calcul retenue consiste à calculer $P(y)$ et $Q(y)$ à chaque début de ligne, puis pour chaque pixel de la ligne à calculer $V_1(x)$.

La structure de la figure 4.1 permet de calculer incrémentalement l'expression du 1er degré $Rx + S$. Or $V_1(x+1) = V_1(x) + (V_1(x+1) - V_1(x))$, $V_1(x+1) - V_1(x)$ étant une expression du 1er degré, calculable avec la structure indiquée. On peut donc calculer incrémentalement $V_1(x)$ en connectant deux structures en série.

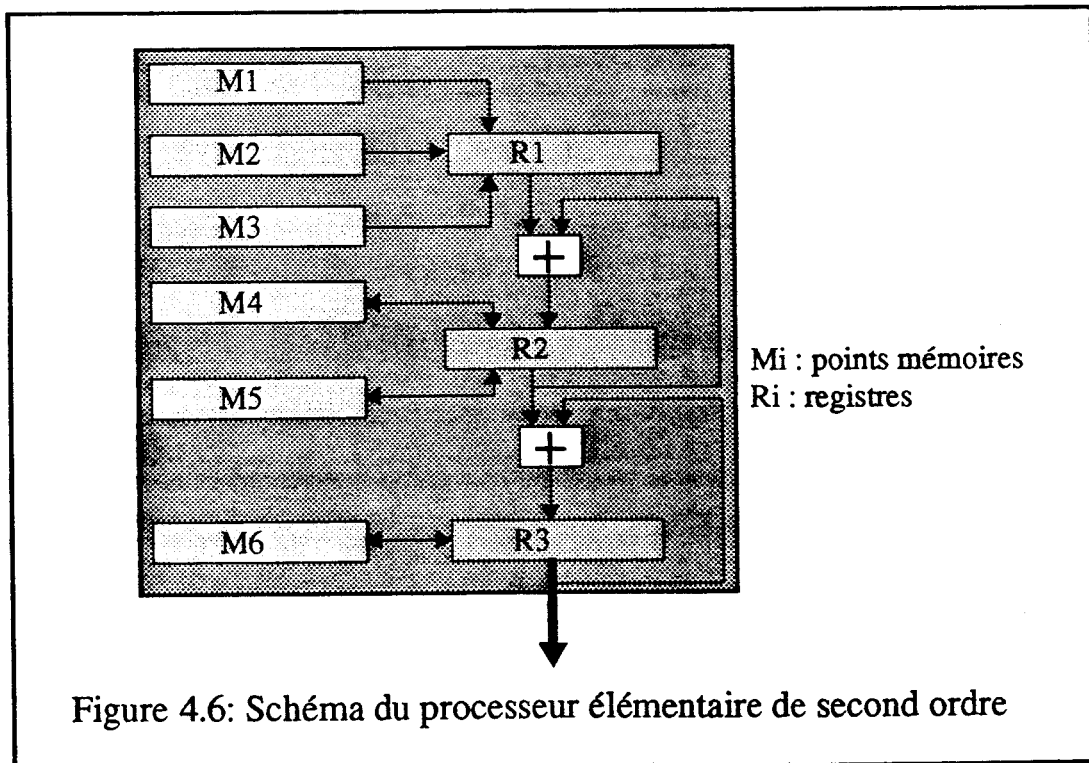
On peut visualiser le procédé de calcul à l'aide du schéma suivant (Figure 4.4):



Les calculs des valeurs $P(y)$ et $Q(y)$ peuvent se faire avec les structures présentées figure 4.5.



On voit ainsi apparaître une structure globale unique, composée de trois registres et de deux additionneurs, sous réserve de sauvegarder quand nécessaire les valeurs de P , I , Q , J , et V dans des registres spécialisés. La figure 4.6 représente le schéma du processeur élémentaire de second ordre, la figure 4.7 l'algorithme de fonctionnement de cette structure.



M1<-2A, M2<-2B, M3<-C, M4<-E-B, M5<-D-A, M6<-F /initialisation/
Pour chaque ligne de l'écran Faire
 | **R1<-M2, R2<-M4, R3<-M6 /mise à jour des registres/**
 | **addition 1er additionneur**
 | **addition 2nd additionneur**
 | **M4<-R2, M6<-R3**
 | **R1<-M3, R2<-M5 /mise à jour des registres/**
 | **addition 1er additionneur**
 | **M5<-R2, R1<-M1**
Pour chaque pixel Faire
 | **addition 1er additionneur**
 | **addition 2nd additionneur**
 | **envoi du résultat**
Fait
Fait

Figure 4.7: Algorithme de fonctionnement de la structure précédente

IV.1.5 Solution matérielle

Toutefois, une telle solution multiplie les connexions entre registres, ne supporte pas le mode entrelacé et complique la liaison avec le module pilote. Nous avons donc choisi une solution moins coûteuse en connexions et en points de contrôle, en remplaçant les registres de sauvegarde par une mémoire (voir Figure 4.8).

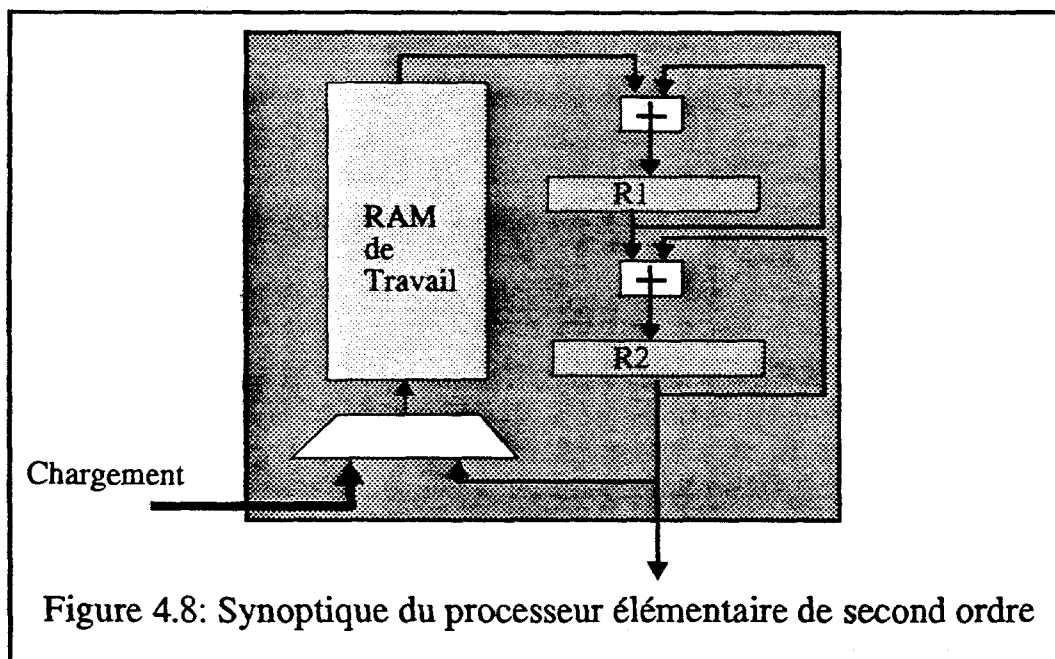


Figure 4.8: Synoptique du processeur élémentaire de second ordre

Une telle solution amène plusieurs remarques :

- La mémoire doit posséder 6 mots pour un balayage non entrelacé, 12 pour un balayage entrelacé (car il faut récupérer les coefficients originaux au début de la 2ème demie trame).
- Le fait d'utiliser une mémoire peut sembler pénalisant, la sauvegarde du 1er registre nécessitant par exemple la remise à zéro du deuxième et au moins 3 tops d'horloge. Toutefois, ces sauvegardes n'étant nécessaires qu'en retour ligne, le facteur temps est alors moins critique.
- L'architecture du processeur laisse apparaître un pipeline évident, les deux additions pouvant être exécutées simultanément. C'est alors le plus lent des éléments qui détermine le temps minimum de génération d'un pixel.

Le processeur est contrôlé par un Contrôleur microprogrammé, implémentant un algorithme de fonctionnement déduit de celui décrit figure 4.7.

IV.1.6 Choix du nombre de bits

Pour réaliser les processeurs élémentaires, il est essentiel de déterminer sur combien de bits doivent être effectuées les additions. Nous supposons que l'écran contient $2^{10} * 2^{10}$ pixels.

Pour l'expression de premier degré, la droite la plus proche de la verticale a une pente de 2^{10} . Le rapport des coefficients A et B est alors de 2^{10} . En supposant que le plus petit des coefficients est 1 le second vaut 2^{10} . Les valeurs x et y varient sur l'écran entre 0 et 2^{10} , donc la valeur maximale que peut atteindre F(x,y) est de l'ordre de 2^{20} . Une vingtaine de bits sont donc nécessaires pour un processeur élémentaire du premier ordre.

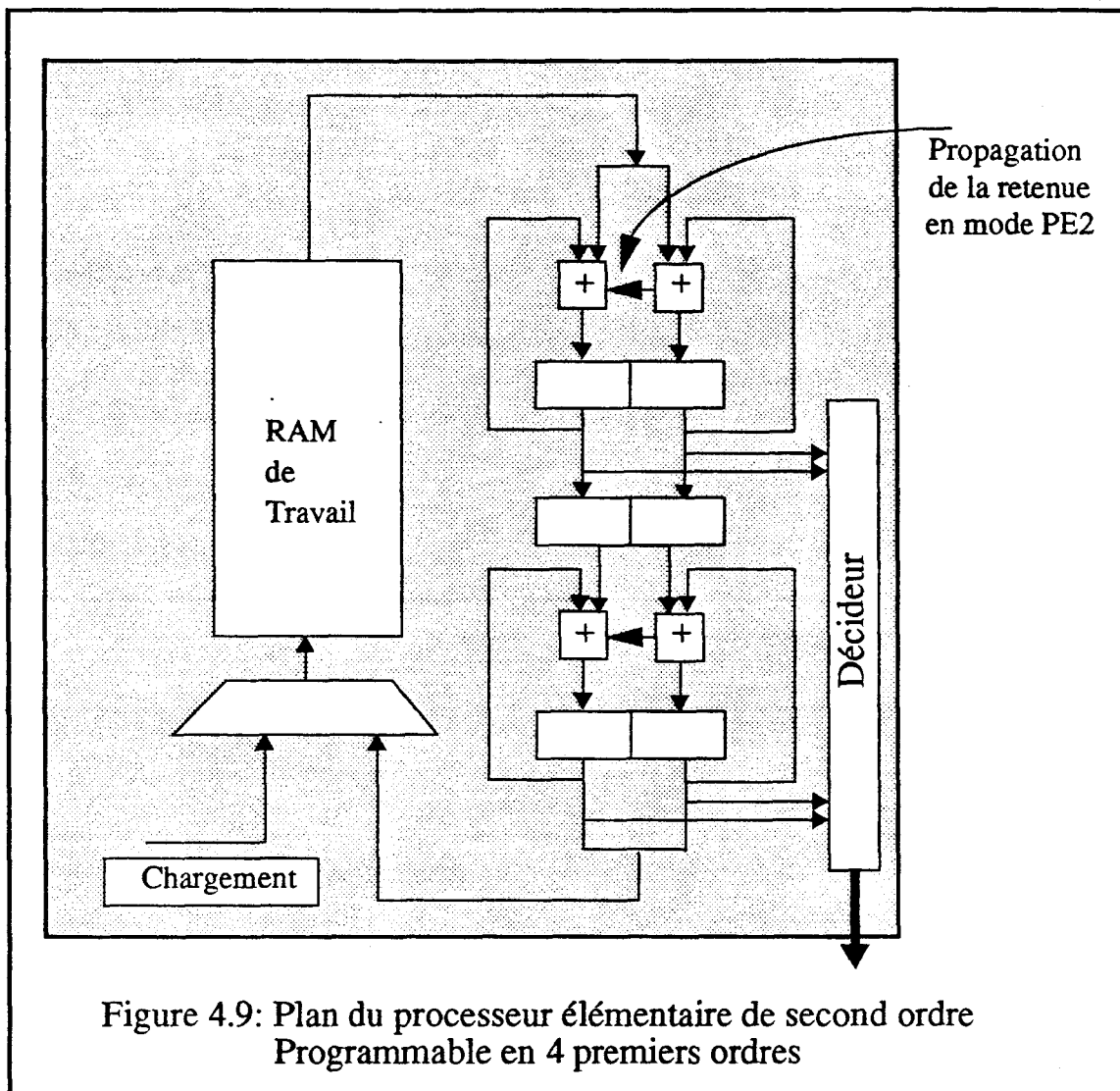
Un raisonnement du même type montre que le rapport entre les deux coefficients A et B de V(x,y) peut atteindre 2^{20} . Ces coefficients étant multipliés par des termes de second degré, V(x,y) peut atteindre 2^{40} . Il faut donc utiliser des additionneurs d'une quarantaine de bits pour le processeur élémentaire de second degré.

Malheureusement, même avec ces tailles, il n'est pas exclu d'avoir des débordements en cours de calcul car il ne sera pas toujours possible de ramener le plus petit des coefficients de l'expression à la valeur 1. En effet il est indispensable dans certains cas de normaliser les expressions en les multipliant par des constantes afin de pouvoir les comparer. Toutefois les différents travaux de simulation, que nous avons mené, laissent espérer qu'utiliser 23 ou 24 bits pour un premier degré et le double en second degré sera suffisant.

IV.1.7 Le Processeur Élémentaire programmable

Pour augmenter la souplesse d'utilisation des processeurs élémentaires, nous envisageons de définir un processeur du second ordre pouvant se diviser en 4 processeurs du premier ordre et un décideur faisant l'unification des 4 valeurs (voir Figure 4.9).

Ce processeur élémentaire dans sa version quatre premier degré permet de visualiser de façon efficace le contour des facettes (voir le paragraphe suivant). Le décideur fait, dans ce cas, l'union des quatre valeurs.



IV.1.8 Un autre Processeur Élémentaire

Afin d'afficher des objets en utilisant le moins possible de processeurs élémentaires, il serait intéressant de définir un processeur élémentaire plus performant. Nous pensons qu'un processeur résolvant une équation du second degré en x , y et z (voir Equation 1) serait particulièrement performant, puisqu'il permettrait de visualiser aisément toutes les quadriques en trois dimensions. Appelons PEQ Processeur Élémentaire Quadratique ce nouveau processeur. Kedem et Ellis ont défini une machine de lancer de rayon dont les primitives graphiques sont des quadriques qu'ils construisent incrémentalement [KedeE189].

$$Az^2 + Bx^2 + Cy^2 + Dzx + Ezy + Fxy + Gz + Hx + Iy + J = 0 \quad (\text{Equation 1})$$

En posant $S = Dx + Ey + G$ et $T = Bx^2 + Cy^2 + Fxy + Hx + Iy + J$, l'Equation 1 devient l'Equation 2.

$$Az^2 + Sz + T = 0 \text{ (Equation 2)}$$

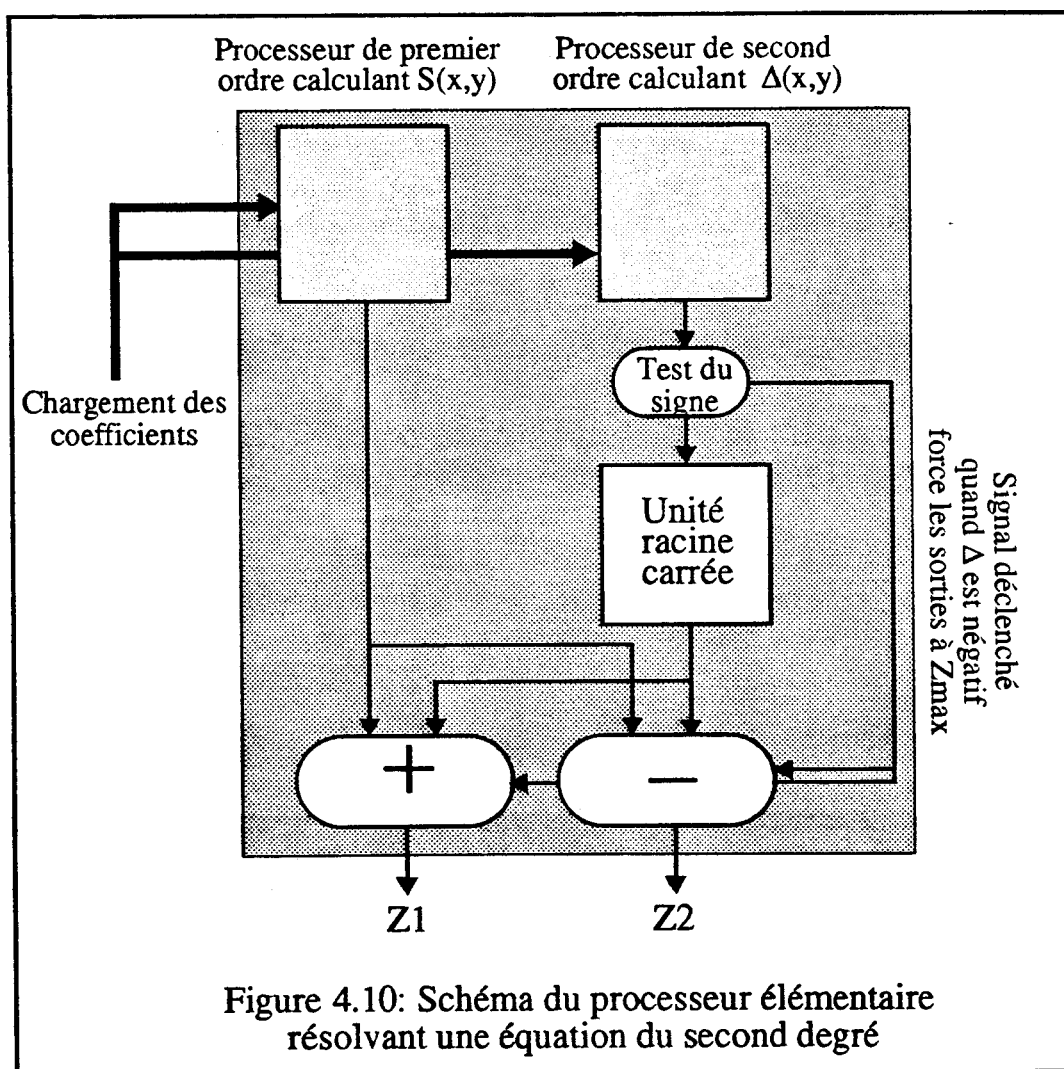
Son déterminant est donc $\Delta = S^2 - 4AT = (Dx + Ey + G)^2 - 4A(Bx^2 + Cy^2 + Fxy + Hx + Iy + J)$

Remarquons que Δ est une expression du second degré en x et y .

Si Δ est positif ou nul les solutions de l'équation sont :

$$Z1 = (-S + \Delta^{.5}) / 2A \text{ et } Z2 = (-S - \Delta^{.5}) / 2A$$

Un processeur élémentaire résolvant cette équation est donc réalisable en utilisant un processeur élémentaire de premier ordre calculant S , un de second ordre calculant Δ et un module permettant de prendre la racine carrée d'un nombre dans le temps imparti à une addition (voir Figure 4.10). La division par $2A$ n'est pas effectuée, les valeurs fournies par le processeurs ne seront pas les profondeurs mais des multiples de celles-ci. Si on veut comparer les valeurs fournies par différents processeurs de ce type il sera nécessaire de faire une normalisation des coefficients des différentes expressions.



L'étude d'un processeur de ce type est actuellement en cours; il existe des solutions matérielles permettant de calculer très rapidement une racine carrée avec une erreur acceptable [Hashem90]. Par contre nous n'avons pas encore étudié le comportement de ce processeur dans les cas où l'équation est dégénérée.

IV.1.9 Réalisation

Dans un premier temps le centre de nos travaux a été la réalisation d'un Processeur Élémentaire de second ordre [Karpf89]. Nous avons réalisé un premier prototype câblé avec des composants standards Dual In Line, puis nous avons défini un circuit imprimé utilisant des composants CMS [BourNo90]. La carte pour un Processeur du second degré non programmable est une double Europe quatre couches.

Ces tentatives valident le principe de base mais montrent que ces solutions sont mal adaptées, les composants commercialisés ne correspondant pas à nos besoins. Par exemple 13 boîtiers sont nécessaires pour réaliser un additionneur 40 bits.

Nous avons donc réalisé un prototype du processeur élémentaire de second ordre avec le logiciel de CAO VLSI SOLO 1400. Le circuit intégré développé [Degran90] travaille sur 32 bits. Il est composé de deux RAM de stockage (pour travailler en mode entrelacé), leur logique de contrôle et la chaîne de calculs (voir Figure 4.8).

Il est réalisé en technologie semi-custom CMOS 2μ . Il contient 2990 portes, environ 25000 transistors et est monté dans un boîtier 68 broches dont 48 sont effectivement utilisées. D'après les simulations il supporte une fréquence de 10 Mhz; il permettra donc d'afficher sur un écran 512x512 en mode entrelacé.

Ce premier circuit montre qu'il est possible de réaliser des processeurs élémentaires avec quelques milliers de portes logiques. On peut donc envisager de définir en full-custom des circuits intégrés comprenant plus d'une dizaine de processeurs élémentaires et ainsi de définir des processeurs capables de visualiser chacun un objet graphique assez complexe.

Nous pensons maintenant définir maintenant des Processeurs Élémentaires 48 bits, travaillant à une fréquence de 16 Mhz pour afficher en mode non entrelacé sur un écran 512x512. Cette option simplifie de façon importante les Processeurs Élémentaires.

IV.2 Les Processeurs Objets

IV.2.1 Définition

Les processeurs objets que nous décrivons dans cette partie sont construits en utilisant les processeurs du premier et du second ordre décrits précédemment.

Les sorties des Processeurs Elémentaires peuvent être interprétées de trois façons différentes :

- une valeur binaire (négative = absent, positive = présent) indiquant si l'objet est présent au pixel courant. La combinaison de plusieurs de ces valeurs permet de définir le contour de l'objet.
- la profondeur de l'objet dans le repère de l'observateur (éventuellement la profondeur arrière qui est nécessaire pour certaines opérations inter-objets).
- une des composantes de la normale à l'objet au pixel courant.

Ces différentes valeurs sont combinées dans un arbre pipeliné appelé Constructeur (voir Figure 4.11).

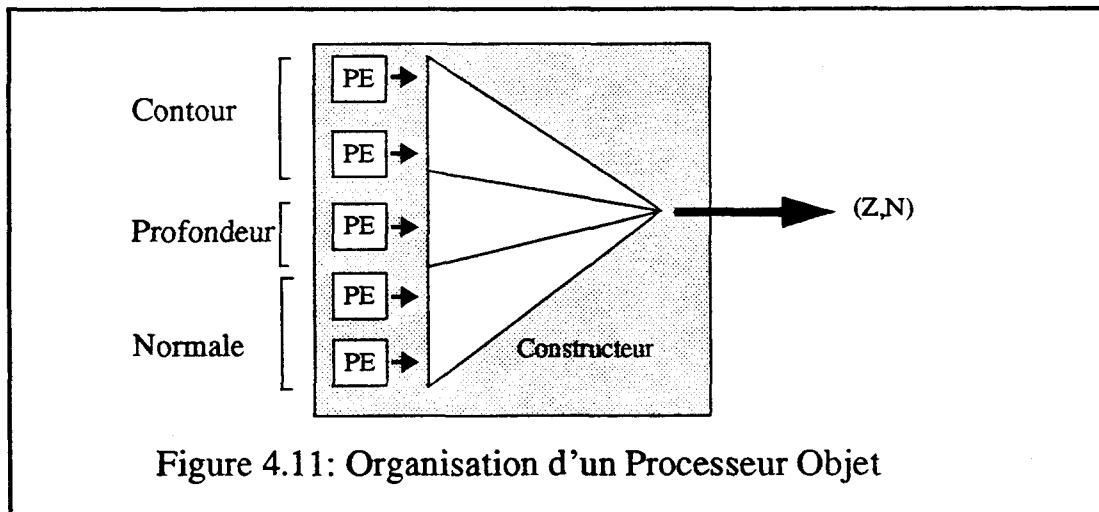


Figure 4.11: Organisation d'un Processeur Objet

Les objets que l'on peut visualiser doivent être convexes afin que leur contour écran soit calculable par une combinaison d'expressions linéaires ou quadratiques ayant une structure fixe, quel que soit l'angle sous lequel l'objet est vu.

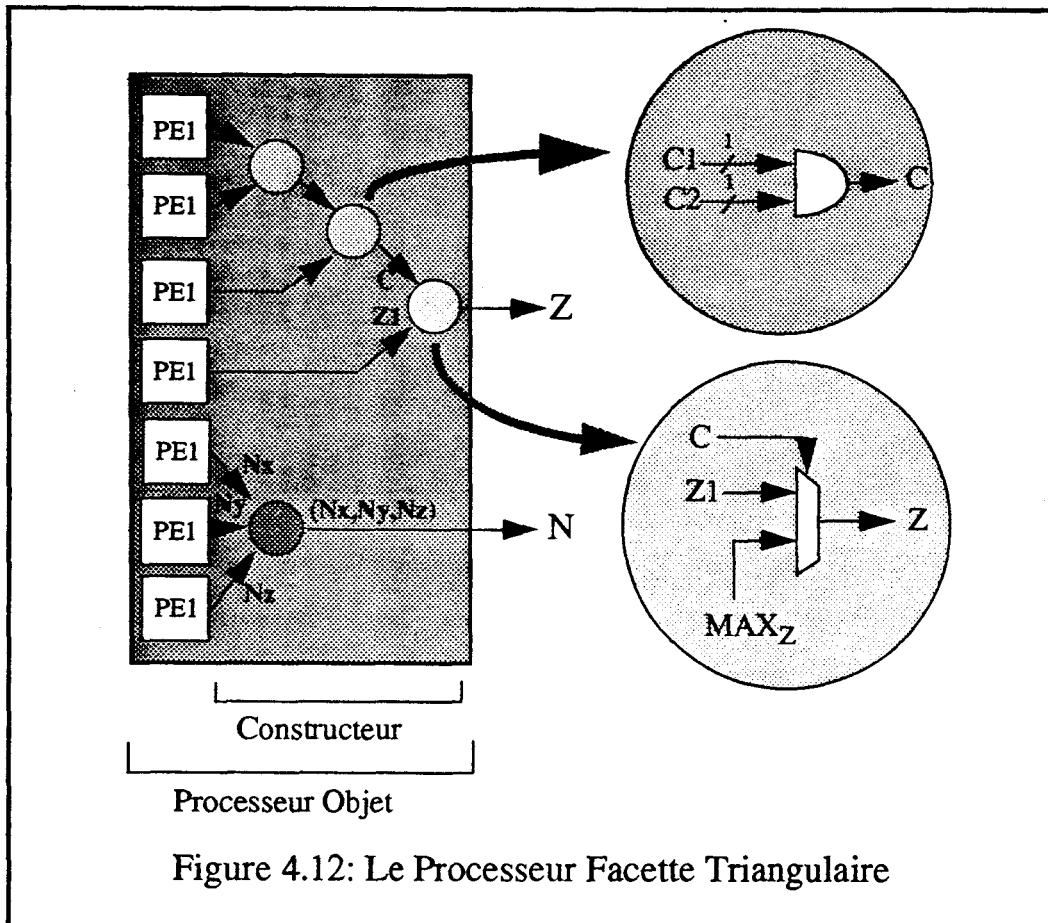
Le Processeur Objet doit aussi fournir les caractéristiques intrinsèques (couleur, coefficients de réflexion ..) de l'objet. Nous n'avons pas fait figurer ces valeurs sur notre schéma car ce sont des constantes.

IV.2.2 Le Processeur Facette plane

Une facette triangulaire est définie par (voir Figure 4.12):

- trois Processeurs Elémentaires PE1 permettant de déterminer le contour écran du triangle par intersection de 3 demi plans.

- un Processeur Élémentaire PE1 déterminant la profondeur Z en tout pixel (i.e. l'équation du plan 3D qui contient le triangle).
- trois Processeurs Élémentaires PE1 calculant les trois composantes (N_x, N_y, N_z) de la normale à la face en utilisant la méthode d'interpolation de Phong si la facette est une approximation de surface gauche. Si la facette est effectivement plane, N_x, N_y et N_z sont des constantes.



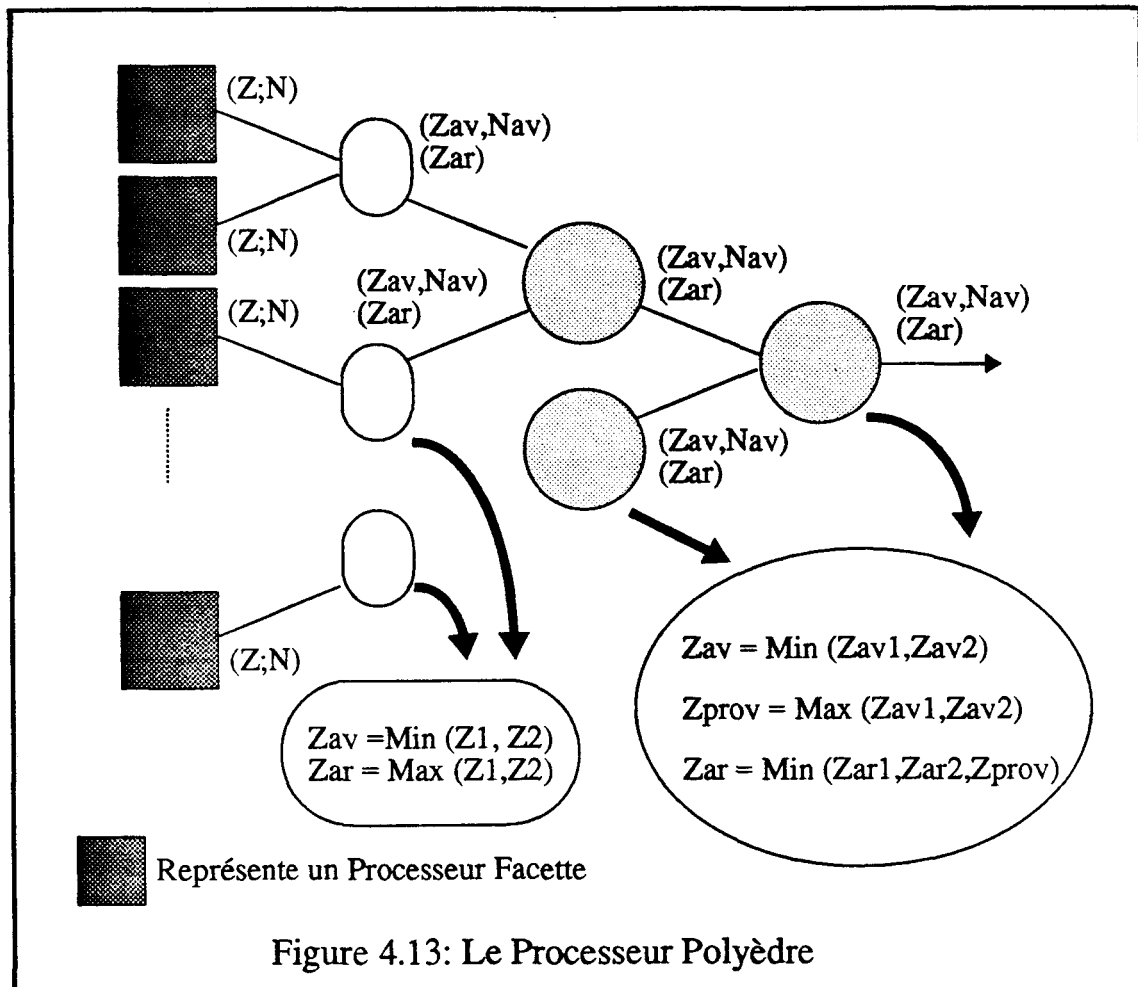
Un processeur permettant de visualiser une facette de plus de trois côtés se déduit facilement de celui d'une facette triangulaire. Toutefois faire des interpolations sur les normales et les profondeurs n'est alors envisageable que si celles-ci sont coplanaires.

IV.2.3 Le processeur Polyèdre

Il n'est pas nécessaire de construire explicitement les polyèdres convexes si la seule opération inter-objets du décideur est l'élimination des parties cachées. On peut alors considérer les différentes faces d'un polyèdre convexe comme des objets indépendants, le décideur faisant implicitement la construction du polyèdre en effectuant l'élimination des parties cachées.

Pour certaines opération inter-objets la construction du polyèdre doit être réalisée. En particulier cela est indispensable pour les volumes d'ombre et la visualisation directe

d'arbres CSG. Pour ces traitements la profondeur arrière des objets est nécessaire. Le processeur effectuant la construction du polyèdre est représenté sur la figure 4.13.



IV.2.4 Le Processeur Sphère

Une sphère est définie par (voir Figure 4.14) :

- un Processeur Élémentaire PE2 définissant le contour de la sphère.
- un Processeur Élémentaire PE2 approchant la profondeur Z en tout pixel.
- trois Processeurs Élémentaires calculant les trois composantes (N_x, N_y, N_z) de la normale à la face, un PE2 pour la composante N_z de la normale et deux composants PE1 pour les autres composantes.

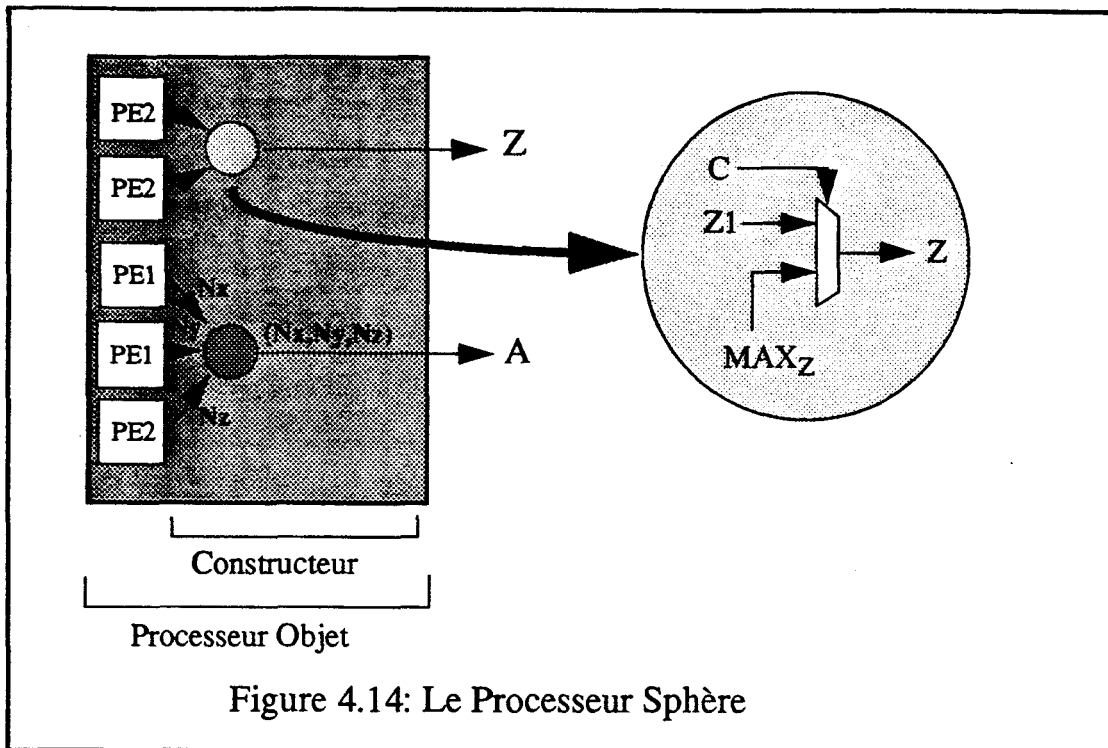


Figure 4.14: Le Processeur Sphère

Le Processeur sphère ne représente que la demi-sphère avant; le contour est exact, la profondeur et la normale approchées. On ne peut, en effet, calculer ni la profondeur exacte de la demi-sphère, celle-ci s'exprimant à l'aide d'une racine carrée, ni la vraie normale puisque sa composante N_z est fonction de la profondeur.

La profondeur de la demi-sphère avant s'exprime par la formule suivante:

$z = z_1 - \sqrt{(x - x_1)^2 + (y - y_1)^2 - R^2}$ où (x_1, y_1, z_1) sont les coordonnées du centre de la sphère et R son rayon.

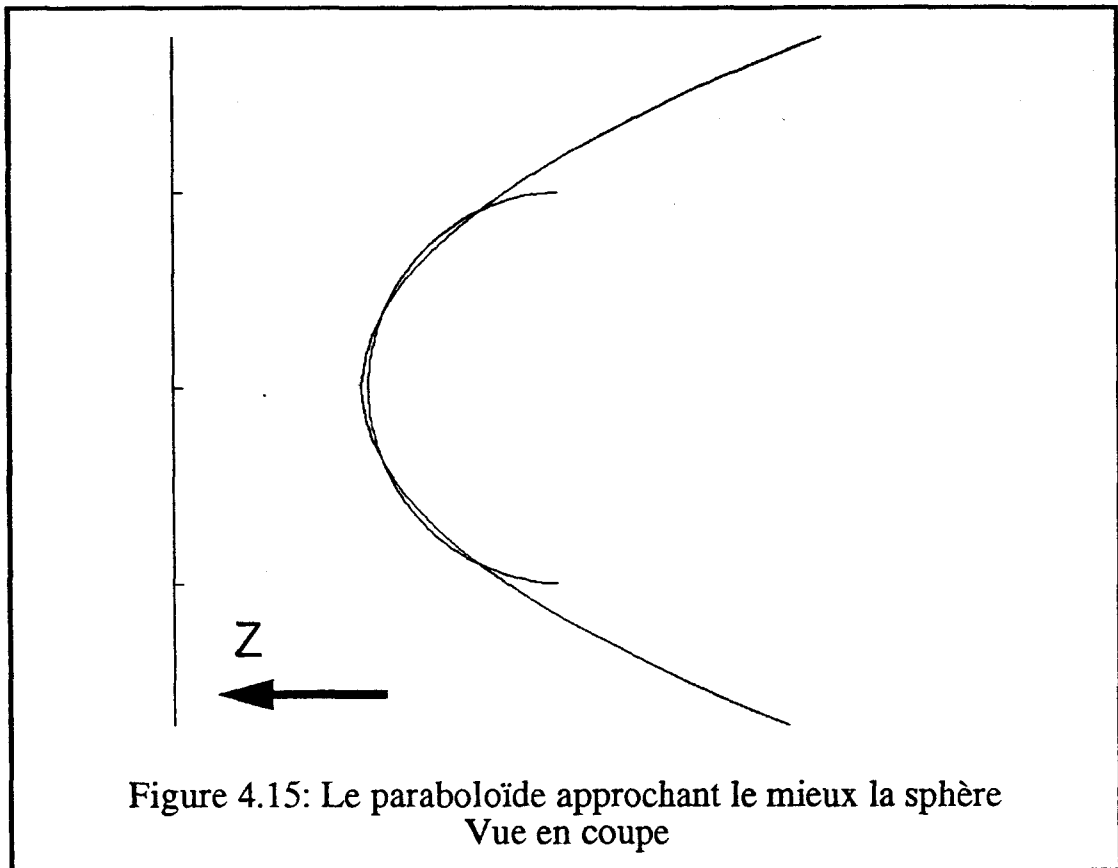
On approche la demi-sphère avant par un parabolôïde.

Nyiri [Nyiri90] a fait une étude et des simulations pour déterminer le meilleur parabolôïde. Le résultat le plus satisfaisant a été obtenu en utilisant la méthode d'approximation des moindres carrées.

Pour modéliser une sphère de centre $M_1(x_1, y_1, z_1)$ et de rayon R , nous utilisons donc le parabolôïde d'équation:

$$\frac{15\pi}{64R} \times (x - x_1)^2 + \frac{15\pi}{64R} \times (y - y_1)^2 + (z - z_1) + \left(-\frac{21\pi R}{64} \right) = 0$$

qui approche correctement la profondeur de la demi-sphère avant (voir Figure 4.15).



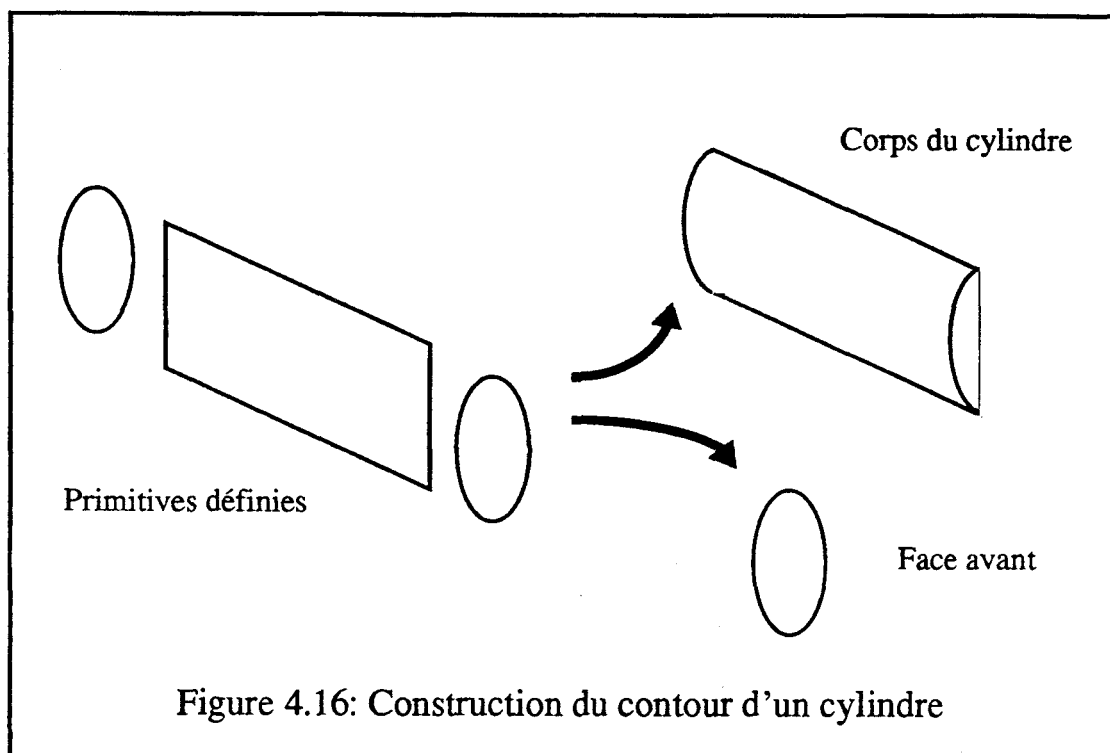
Cette approximation donne de bons résultats visuels sauf dans un cas particulier. L'intersection de deux paraboloïdes approchant des sphères de même taille est une parabole perpendiculaire à l'écran qui se projette en un segment sur l'écran. L'intersection réelle est une demi-ellipse, l'oeil perçoit ce défaut de façon très importante.

Nous avons tenté de résorber ce défaut en utilisant deux paraboloïdes pour approximer une demi-sphère. Dans certains cas, il a été possible de trouver deux paraboloïdes donnant un excellent résultat visuel. Utiliser deux paraboloïdes entraîne, le plus souvent, des résultats moins bons que l'utilisation de l'unique paraboloïde déterminé avec la méthode des moindres carrés.

Pour certaines opérations inter-objets il est nécessaire de connaître la profondeur arrière des objets. Pour une sphère il est très facile de définir la face arrière en ajoutant dans le processeur objet un processeur élémentaire du second degré calculant la profondeur arrière avec un procédé identique au calcul de la profondeur avant.

IV.2.5 Le Processeur Cylindre

La partie visible d'un cylindre est composée de deux parties distinctes: le corps du cylindre et la partie avant qui est une ellipse. Avec les primitives dont nous disposons nous devons les traiter séparément. Le contour du corps du cylindre est défini par une facette (parallélogramme) et deux ellipses (voir Figure 4.16), une de ces deux ellipses étant la face avant du cylindre.



Un cylindre est défini par (voir Figure 4.17):

- Six Processeurs Élémentaires pour le contour: 4 PE1 pour définir le parallélogramme et 2 PE2 pour les ellipses.
- Deux Processeurs Élémentaires pour la profondeur: 1 PE2 pour le corps du cylindre et 1 PE1 pour l'ellipse avant.
- Trois Processeurs Élémentaires PE2 calculant les trois composantes (N_x, N_y, N_z) de la normale au corps du cylindre. Les composantes de la normale pour la face avant sont des constantes.

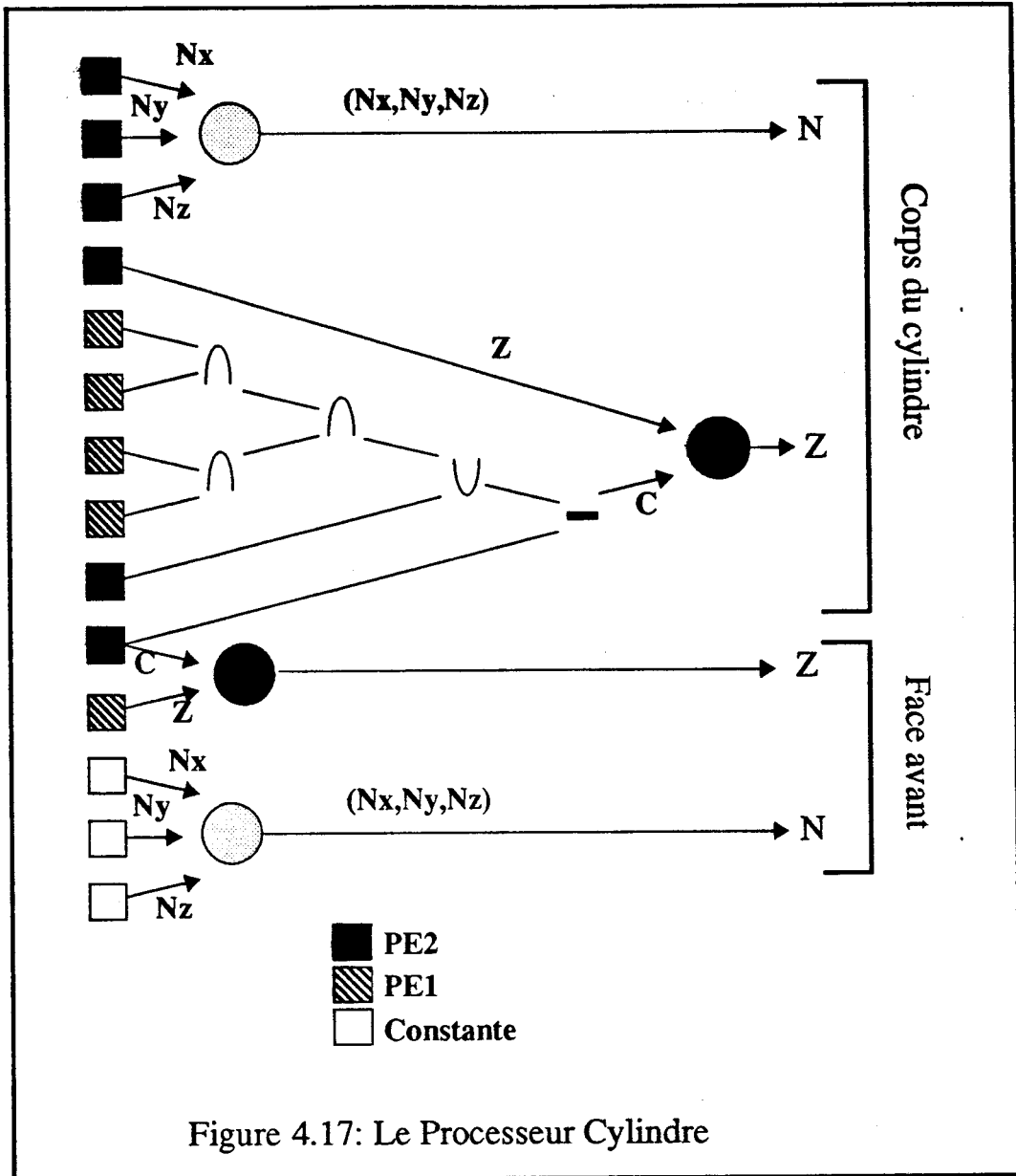


Figure 4.17: Le Processeur Cylindre

Les contours obtenus avec ce processeur sont exacts. La profondeur et la normale en chaque point du corps du cylindre sont approchées. Comme pour la sphère, on ne peut calculer la profondeur exacte, celle-ci s'exprimant avec une racine carrée.

Nyiri [Nyiri90] a mis au point une méthode calculant la surface parabolique approchant au mieux la profondeur du corps du cylindre. La même méthode permet d'approcher correctement les trois composantes de la normale qui sont des fonctions linéaires de la profondeur.

Définir la face arrière du cylindre est plus difficile que déterminer celle de la sphère. Il faut en effet définir le contour écran de la partie arrière du corps du cylindre. Il faut pour cela établir un nouveau constructeur à partir des processeurs élémentaires définis sur la figure 4.17 qui établit deux réseaux d'opérateurs, l'un travaillant pour les faces avants l'autre pour les faces arrières.

IV.2.6 Autres Processeurs Objets

IV.2.6.1 Le Processeur Facette non Plane

Nous pouvons, sur le modèle du Processeur Facette Plane, définir un processeur permettant de visualiser des facettes non planes en remplaçant les processeurs du premier ordre par des processeurs de second ordre.

Utiliser de tels processeurs induit de grosses difficultés de modélisation. Il est en effet difficile de décomposer un objet sous forme de facettes non planes convexes affichables quelle que soit leur position par un Processeur Facette non Plane. La détermination des coniques dont l'intersection délimite le contour écran de la facette est un problème délicat.

Des processeurs de ce type sont utilisables pour afficher des objets modélisés par des quadriques ou des courbes splines du second degré.

IV.2.6.2 Processeurs utilisant le processeur PEQ

Nous envisageons de définir de nouveaux processeurs objets en utilisant le Processeur Élémentaire Quadrique présenté précédemment.

Utiliser ce processeur remet en cause la structure -contour,profondeur,normale- que nous avons utilisée pour les processeurs présentés jusqu'ici. En effet si les valeurs Z1 et Z2 fournies par un Processeur Élémentaire Quadrique sont interprétées comme des profondeurs, le processeur fournit de fait le contour de la primitive. Les valeurs Z1 et Z2 valent la valeur Zmax quand la primitive n'est pas présente au pixel considéré.

Une quadrique représentable par un Processeur Élémentaire Quadrique est le plus souvent infinie, il faut donc étudier comment définir des primitives graphiques à l'aide de quadriques en trois dimensions.

Notre étude sur les processeurs de ce type ne fait que commencer et nous semble prometteuse. En effet, les processeurs Quadriques donnent des valeurs exactes qui permettent d'éviter de faire des approximations comme celles présentées pour la sphère et le cylindre. De plus ces primitives de haut niveau permettront de visualiser des objets complexes avec un nombre faible de Processeurs Élémentaires.

Les quadriques présentent une autre qualité jusqu'ici non évoquée: elles permettent d'envisager des animations avec peu de pré-calculs. Tout déplacement dans l'espace d'une quadrique peut, en effet, être assimilé à un changement de repère.

IV.2.7 Réalisation, contraintes physiques

IV.2.7.1 Nos objectifs

A terme notre objectif est de réaliser un circuit intégré composé de plusieurs Processeurs Élémentaires et d'un constructeur pouvant visualiser un objet graphique complexe.

Nous espérons pouvoir définir un processeur programmable afin de traiter des primitives graphiques de divers types.

IV.2.7.2 Difficultés de réalisation

Pour réaliser un Processeur Objet nous devons répondre à de nombreuses questions, les plus délicates étant:

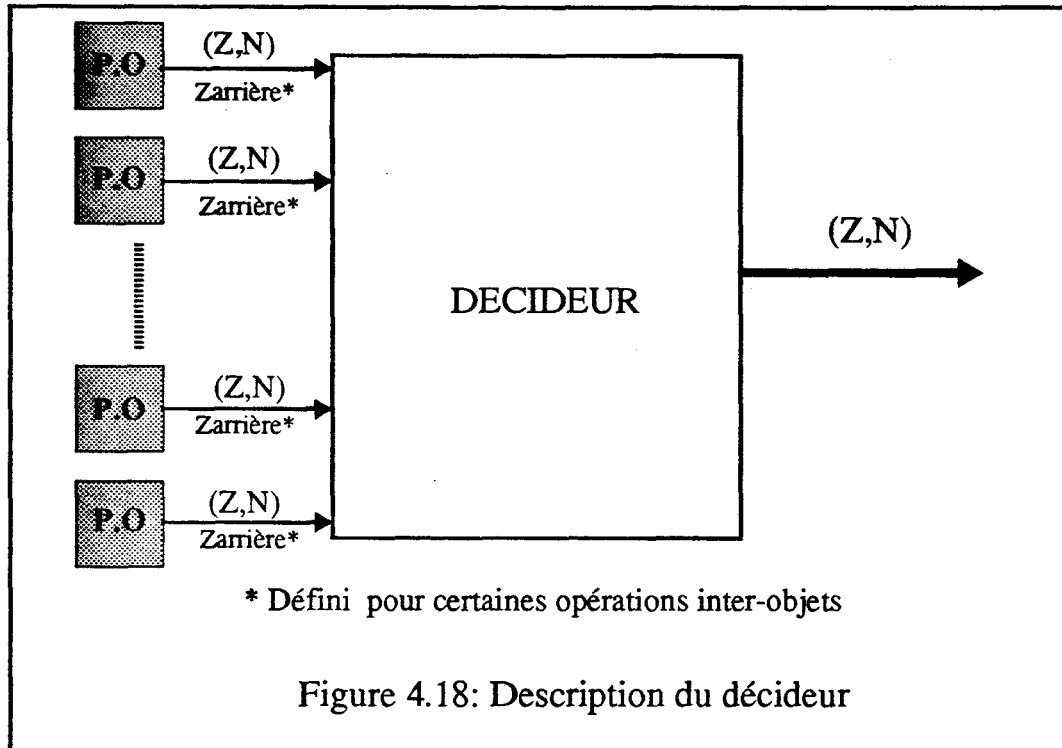
- Quelle liaison utiliser pour charger les séries de coefficients nécessaires à chaque processeur élémentaire ?
- Peut-on pour une image donnée afficher plusieurs objets disjoints verticalement avec un seul Processeur Objet ?
- Peut-on envisager d'intégrer plusieurs processeurs objets dans un même circuit ?

Technologiquement, il n'est pas possible de réaliser beaucoup de Processeurs Objets distincts. Nous devons soit définir des Processeurs capables de visualiser plusieurs primitives différentes, soit limiter le nombre de primitives.

IV.3 Le Décideur

IV.3.1 Présentation

Le décideur est la partie de la machine I.M.O.G.E.N.E la plus délicate à concevoir. Il doit regrouper les informations fournies par les Processeurs Objets et effectuer les opérations inter-objets (voir Figure 4.18). Les différents Processeurs Objets fournissent leurs valeurs au rythme du balayage écran, le décideur doit donc travailler à la même vitesse.



Les difficultés pour réaliser le décideur sont de deux ordres: Algorithmique et Matérielle

- Algorithmique

Les opérations faites dans le décideur sont fonction des traitements inter-objets souhaités. Dans les paragraphes suivants nous montrerons que certaines opérations sont aisément réalisables alors que d'autres ne le sont pas ou de façon très limitée.

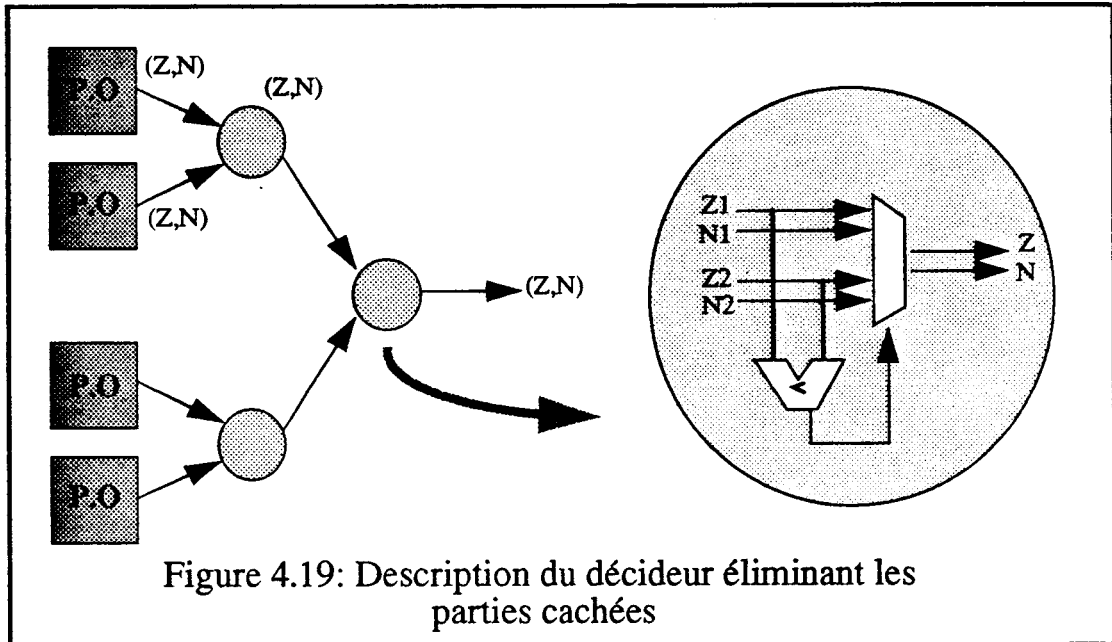
- Matérielle

La réalisation matérielle d'un mécanisme regroupant les sorties d'un nombre très important de processeurs pose un problème technologique difficile à résoudre. Nous présenterons quelques solutions à la fin de cette partie.

IV.3.2 Elimination des parties cachées

L'élimination des parties cachées est l'opération inter-objets la plus simple. La valeur Zarrière n'est pas nécessaire. Dans ce cas, le décideur peut être un arbre binaire pipeliné

implémentant une version distribuée de l'algorithme du Zbuffer. Tous les opérateurs de l'arbre sont identiques, ils gardent le couple (Z,N) ayant la plus petite profondeur (voir Figure 4.19). Bien évidemment, il est possible d'utiliser les différents processeurs en pipeline comme le font toutes les autres propositions exploitant un parallélisme massif objet (Weinberg, GSP-NVS et PROOF).



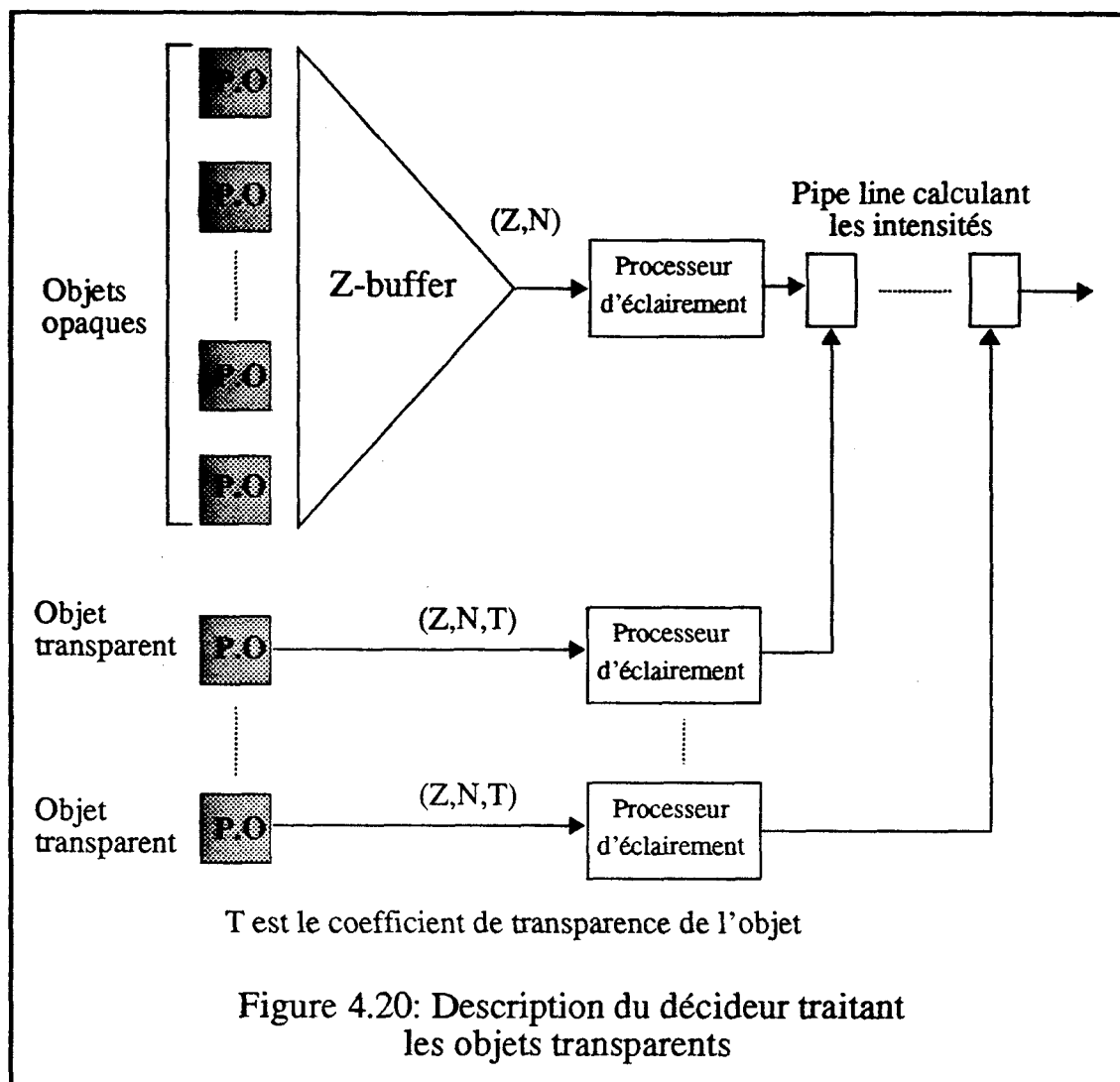
Dans le cas où tous les Processeurs Objets sont identiques, tout processeur peut traiter n'importe quel objet, ce qui facilite grandement la répartition, faite par l'ordinateur hôte, des primitives graphiques aux Processeurs Objets.

IV.3.3 Traitement des objets transparents

Le traitement des objets transparents se fait avec la méthode présentée au paragraphe III.1.1.3. Elle ne nécessite pas de définir les faces arrières des objets. Mais pour faire les calculs d'éclairage en un point, il faut connaître les intensités pour l'objet opaque visible et les différents objets transparents. Cette méthode impose donc d'avoir autant de post-processeurs d'éclairage que d'objets transparents (voir Figure 4.20).

Par ailleurs, les objets transparents doivent être ajoutés à l'image du plus éloigné au plus proche de l'observateur. Ceci nous impose deux contraintes:

- Les différents objets transparents doivent être traités en pipe-line par le décideur.
- Les objets transparents doivent être chargés dans l'ordre de leur classement en profondeur dans les différents processeurs spécifiques. Si le tri en profondeur des objets n'est pas faisable, il faut faire des découpages d'objets.



Les différents opérateurs du pipe-line calculant les intensités sont tous identiques et très simples. Si l'objet transparent est devant l'objet visible on effectue un calcul avec l'expression fournie au paragraphe III.1.1.3.

Sur la figure 4.20 apparaissent clairement les difficultés qu'implique le traitement des objets transparents. Il faut autant de post-processeurs d'éclairage que d'objets transparents présents dans une scène donnée. Les objets transparents doivent être triés par le processeur hôte et affectés ensuite à un processeur précis.

Il n'est pas envisageable matériellement de réaliser un décideur traitant plus de quelques objets transparents.

IV.3.4 Traitement des ombres portées

Avec les concepts que nous avons choisis, seule la méthode de Crow est utilisable (voir le paragraphe III.1.1.2). Les cônes d'ombre sont définis comme de nouveaux objets. Il est nécessaire pour ces objets de calculer la profondeur avant et la profondeur arrière. Le

principe de calcul est le suivant: si pour une source donnée l'objet visible est à l'ombre (quand sa profondeur est comprise entre les deux profondeurs de l'ombre) on n'effectue pas le calcul d'éclairage pour cette source.

Les différents volumes d'ombre pour une source donnée sont traités en pipe-line. Si l'un de ces volumes place l'objet visible à l'ombre, on considère la source comme n'existant pas en ce point et on court-circuite le processeur d'éclairage correspondant (voir Figure 4.21).

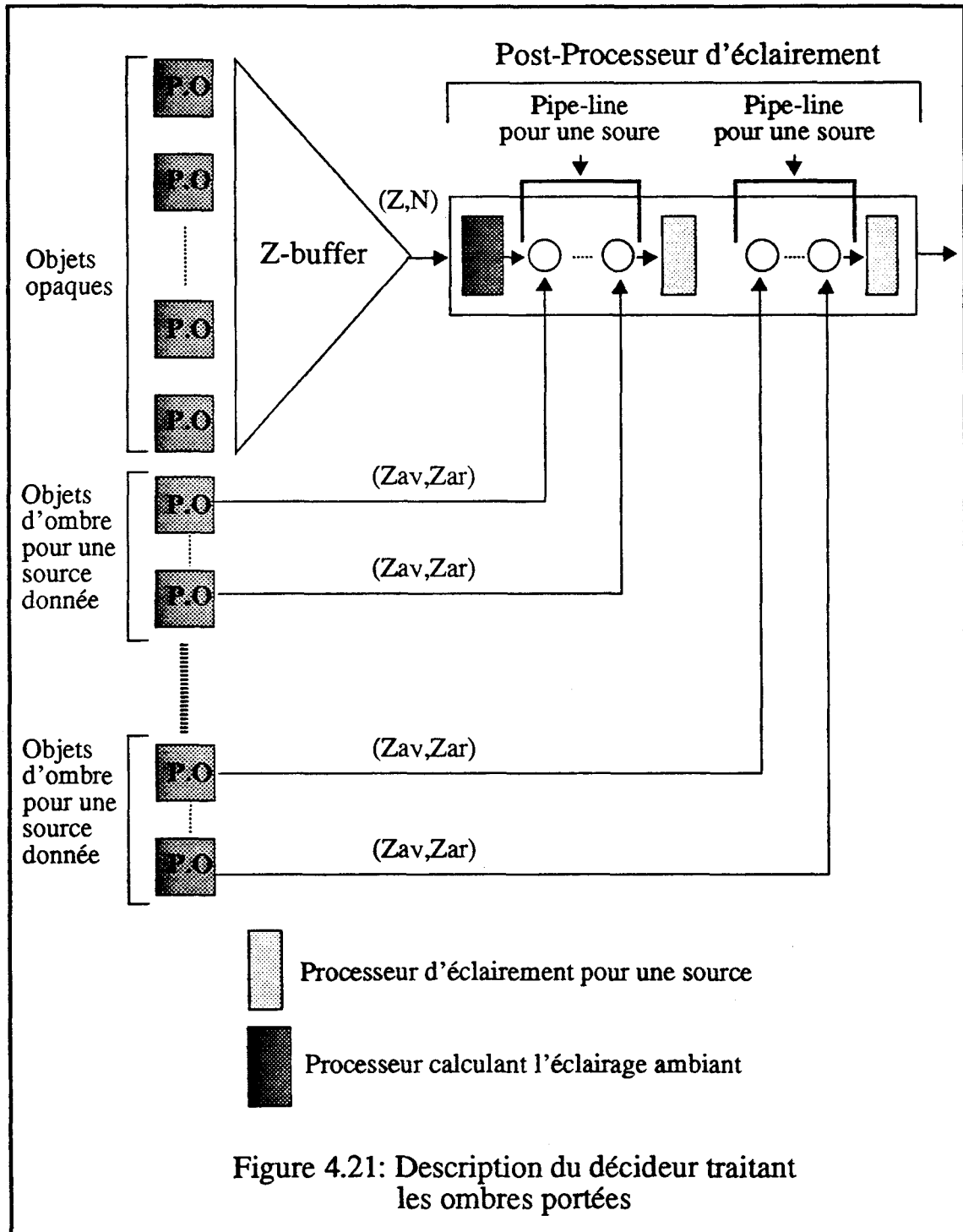


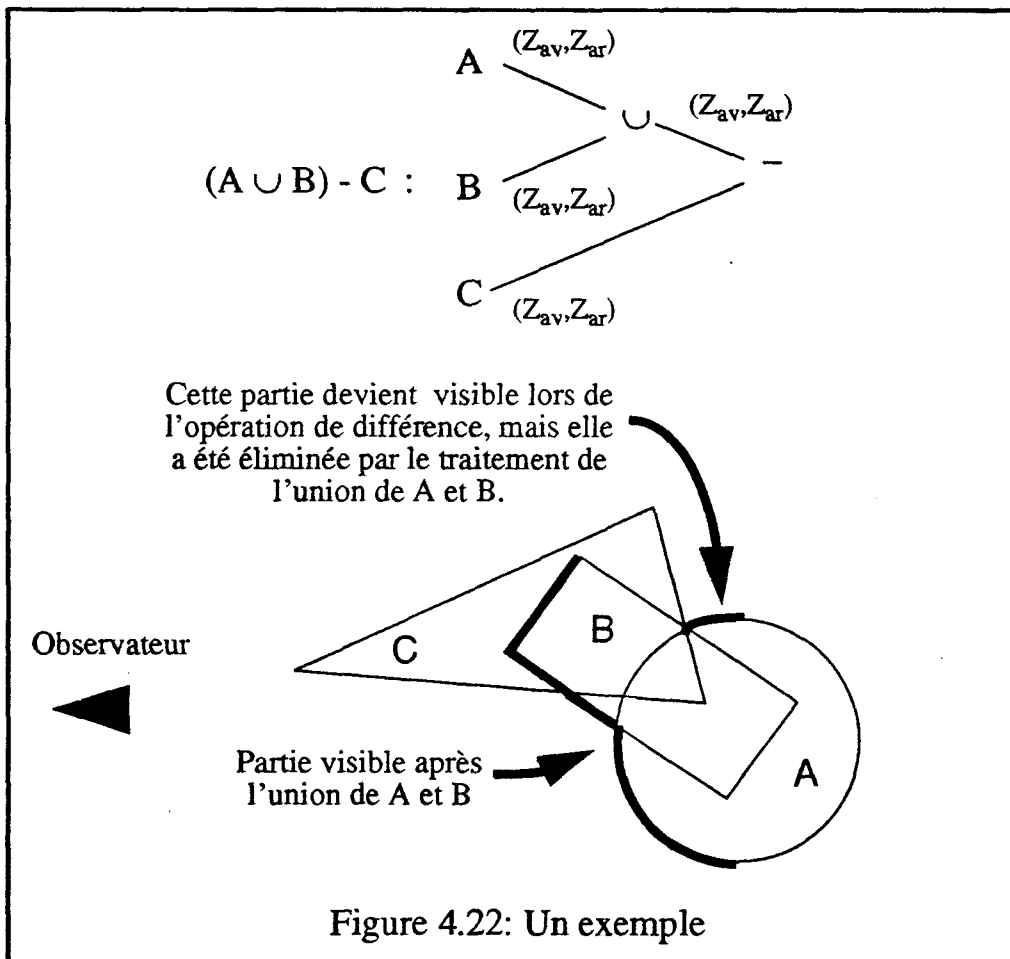
Figure 4.21: Description du décideur traitant les ombres portées

Les calculs d'éclairément pour les différentes sources lumineuses peuvent être effectués en parallèle.

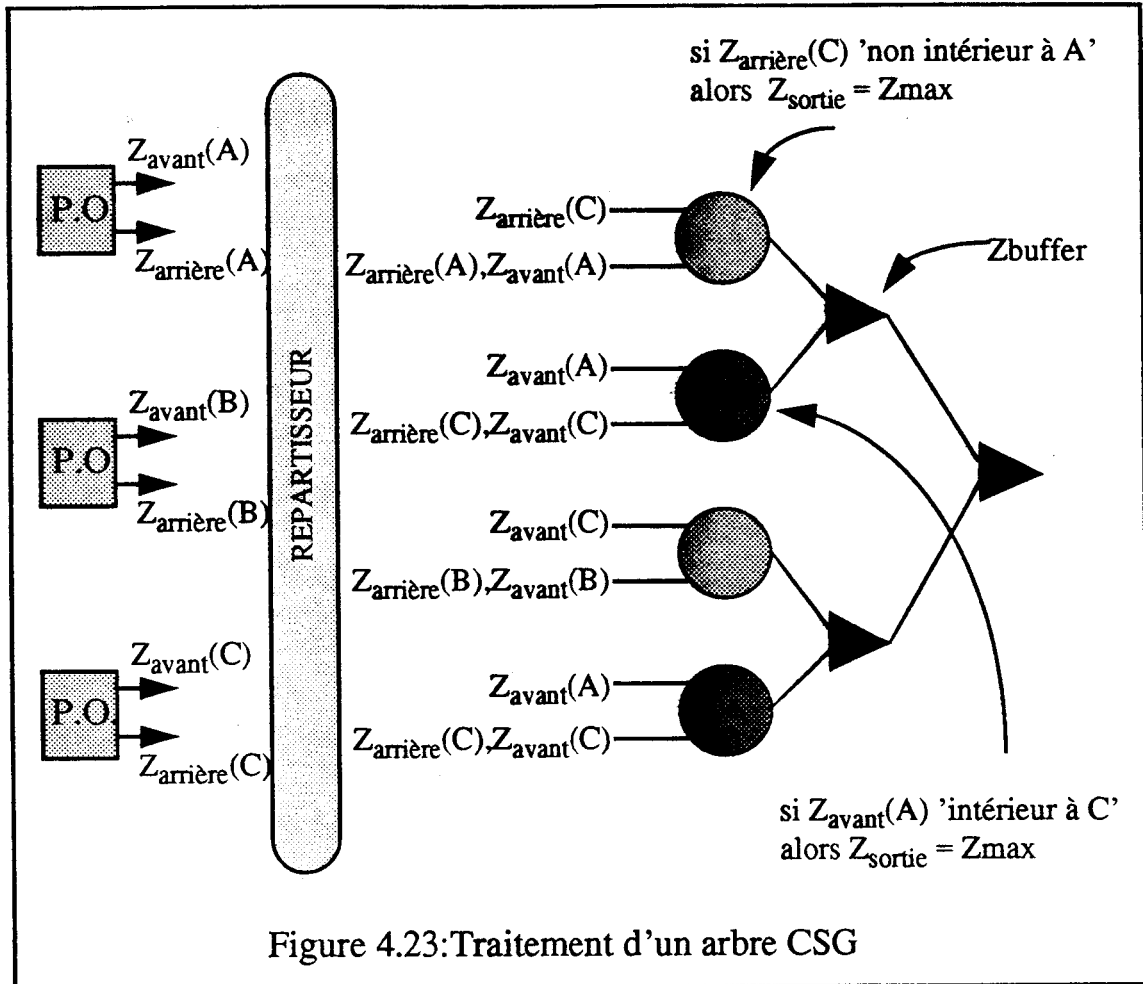
Cette méthode permet de prendre en compte correctement les ombres portées, mais en utilisant un nombre très important de Processeurs Objets.

IV.3.5 Visualisation directe d'arbre CSG

Il serait très intéressant de visualiser directement des arbres CSG. L'idée la plus simple est d'implanter directement l'arbre CSG dans le décideur. Malheureusement, ceci n'est pas envisageable. Une opération inter-objets peut nécessiter des valeurs qui ne sont plus disponibles vu les traitements déjà effectués. Un exemple est présenté Figure 4.22.



Une solution est d'utiliser la méthode, proposée par Goldfeather, présentée au paragraphe III.1.1.3.4. Elle nécessite un pré-traitement pour normaliser l'arbre CSG. Le traitement de l'arbre de la Figure 4.22 avec cette méthode est présenté Figure 4.23.



Avec l'architecture que nous avons choisie, cette solution présente une difficulté considérable: le Répartisseur. En effet réaliser matériellement un mécanisme capable de répartir un certain nombre d'entrées sur un nombre important de sorties à la vitesse du balayage écran est très difficile. Par contre le décideur affichant ensuite l'arbre est aisément réalisable.

La bonne solution est de définir un nouveau traitement de l'arbre qui permettrait d'afficher des arbres CSG sans utiliser de répartisseur ou avec un répartisseur beaucoup plus limité que celui présenté. Nous devons exploiter les caractéristiques de notre architecture, en particulier le fait que nos primitives de base sont convexes et que l'intersection de deux objets convexes est convexe.

Nous sommes à la recherche d'une solution permettant de visualiser efficacement des objets définis par des arbres CSG avec la machine I.M.O.G.E.N.E.

IV.3.6 Problèmes d'ordre matériel

La réalisation matérielle du décideur soulève les problèmes suivants:

- Doit-on réaliser un seul circuit intégré programmable capable de prendre en compte de nombreuses opérations inter-objets ou réaliser plusieurs circuits distincts ayant chacun une fonction déterminée ?

- Réaliser un circuit regroupant une partie importante de l'arbre de décision impose d'avoir un nombre de broches considérable pour entrer les informations. Par exemple un circuit intégrant un arbre binaire de trois niveaux nécessite huit entrées. Chaque entrée représente une cinquantaine de bits à acquérir en parallèle (une vingtaine pour la profondeur, autant pour les composantes de la normale et une dizaine pour les caractéristiques propres de l'objet). Une solution est d'utiliser un pipe-line plutôt qu'un arbre binaire. Le décideur peut alors être en partie réparti dans les différents Processeurs Objets. Cette solution permet de ne pas définir de nouveau circuit intégré, mais des problèmes de décalage entre les différents Processeurs Objets apparaissent.

Actuellement nous réalisons un décideur limité capable d'éliminer les parties cachées en utilisant un compromis entre l'arbre binaire et le pipe-line. Par exemple, la décision entre différents processeurs d'une même carte est faite en pipe-line, les différentes cartes fonctionnent en parallèle, un circuit spécifique sert de décideur entre les sorties des différentes cartes.

Résumons: étudier un décideur sous forme d'arbre binaire pose des difficultés matérielles. Toutefois l'arbre permet d'envisager de traiter d'autres opérations inter-objets que l'élimination des parties cachées (en particulier pour le CSG).

IV.4 Le module Graphique

IV.4.1 Schéma d'ensemble

La Figure 4.24 donne une vue globale du Module Graphique. Les Processeurs Objets et le Décideur ont déjà été décrits, restent à présenter le Processeur d'éclairage et l'Unité de Commande.

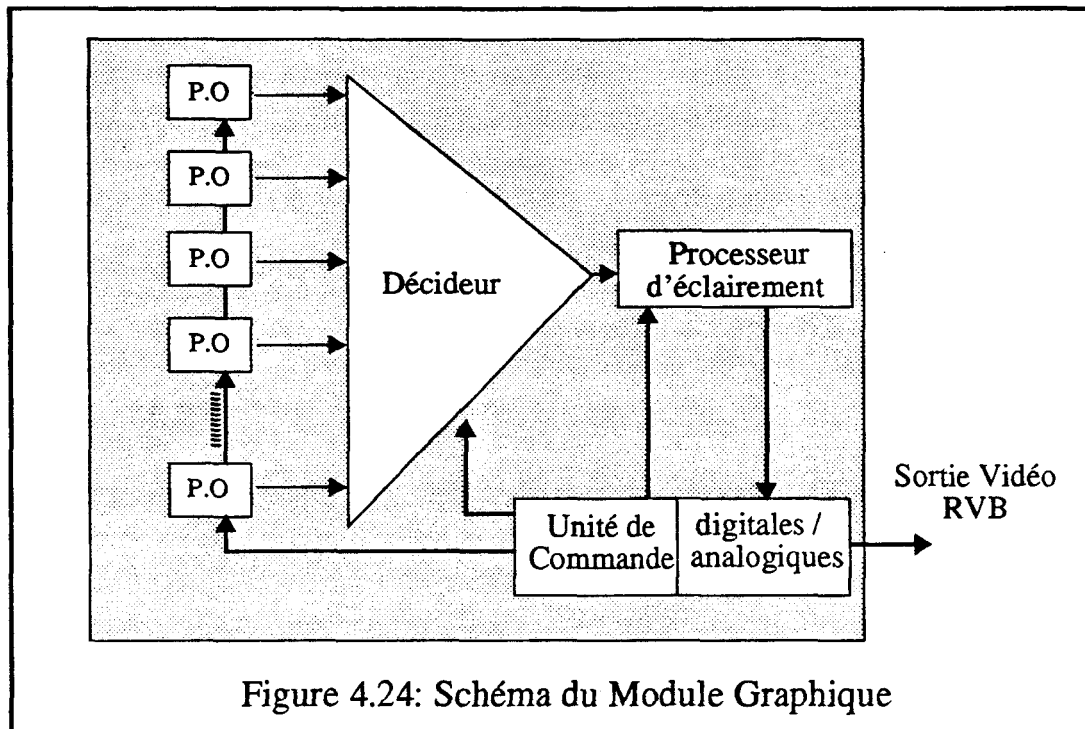


Figure 4.24: Schéma du Module Graphique

Le Module Graphique travaille intégralement au rythme du balayage écran. Le traitement des pixels est totalement pipe-liné. Si une unité ne respecte pas la vitesse il faut la découper en différentes parties effectuées en pipe-line ou en parallèle.

IV.4.2 Le Processeur d'Eclairage

Il fonctionne également au rythme du balayage écran. Il reçoit pour chaque pixel la profondeur, la normale et les caractéristiques intrinsèques (couleur de base, coefficient de réflexion...) de l'objet vu. En fonction de la position et des caractéristiques des sources lumineuses, il calcule les valeurs Rouge, Vert, Bleu à afficher.

Deux études ont déjà été menées pour réaliser un post-processeur d'éclairage pour des machines de visualisation exploitant une approche objets [Clauss90] [Clauss91] [Schnei90] [DeerWS88]. Ces travaux montrent que réaliser un post-processeur calculant l'éclairage en temps réel nécessite une partie matérielle très importante.

Nous commençons à définir un processeur d'éclairage à partir de l'expression présentée au paragraphe III.1.2.1.4. Il nous faut définir des structures matérielles permettant

de normaliser des vecteurs, de calculer des produits, des produits scalaires et des puissances, le tout très rapidement. La composante diffuse et la composante spéculaire peuvent être calculées en parallèle, les calculs pour les différentes sources lumineuses peuvent aussi être parallélisés. Nous pensons réaliser un composant intégré dont nous utiliserons plusieurs exemplaires en parallèle pour calculer l'éclairement en tout point. Une structure pipe-line est utilisée pour le composant élémentaire afin que l'ensemble du Module Graphique travaille au rythme du balayage de trame.

Il serait possible de ne pas utiliser de processeur d'éclairement en chargeant les Processeurs Elémentaires qui calculaient les composantes de la normale avec des expressions fournissant directement les valeurs RVB. Cependant la qualité du rendu est alors bien inférieure car on utilise des approximations et non des valeurs exactes. Dans le cas d'approximation de surfaces gauches par des facettes, cela revient à utiliser l'algorithme de Gouraud alors que le processeur d'éclairement utilise celui de Phong.

IV.4.3 L'Antialiassage

Nous avons montré dans le paragraphe III.1.5 qu'il existe deux méthodes pour antialiasser: le filtrage et le sur-échantillonnage.

Sur-échantillonner impose de traiter l'image avec une résolution au minimum quatre fois supérieure à celle de l'écran. Utiliser cette solution avec notre machine imposerait de faire travailler tous les Processeurs Elémentaires, les Constructeurs et le Décideur quatre fois plus vite. Pour un écran 512x512 en mode non entrelacé, le Module Graphique devrait travailler à une fréquence de 64 Mhz; ceci n'est pas envisageable technologiquement aujourd'hui.

Nous utiliserons donc un filtrage avec une matrice 2x2 ou 3x3 après le processeur d'éclairement. Cette solution impose de mémoriser, de façon provisoire, une ou deux lignes.

IV.4.4 L'Unité de Commande

Cette unité contrôle l'ensemble du Module Graphique. Elle gère l'horloge système, les signaux de synchronisation des Processeurs Objets, du Décideur, du Processeur d'Eclairement et de l'écran graphique (retour ligne et retour trame). Elle réalise les conversions digital/analogique en sortie du processeur d'éclairement.

Nous envisageons d'utiliser un contrôleur graphique commercial pour réaliser cette unité (par exemple IMS G300 INMOS). Les contrôleurs graphiques commerciaux fournissent les signaux gérant l'écran, et font la conversion digitales/analogiques.

IV.4.5 La liaison Module Pilote / Module Graphique

Pour garantir une animation temps réel, la mémoire de coefficients doit être rechargée par le module pilote au début de chaque trame. Par ailleurs, il faut garder à l'esprit que nous étudions une solution massivement parallèle : le module pilote aura donc à charger un très grand nombre de processeurs élémentaires. Il faut pour cela qu'il puisse disposer du maximum de temps possible.

Pour respecter ces fortes contraintes de temps, une solution est d'utiliser une mémoire

de chargement commune à tous les processeurs élémentaires, et un automate de chargement câblé chargé du transfert des coefficients de cette mémoire vers toutes les mémoires de travail pendant le retour trame. La mémoire de chargement (M2) est quant à elle chargée par le module pilote pendant l'image précédente. Par ailleurs, il est intéressant de câbler la mémoire de chargement (M2) comme extension de la mémoire de l'ordinateur hôte (M1): vue du module pilote, l'affectation des coefficients aux processeurs n'est alors rien d'autre qu'un transfert de mémoire à mémoire (voir Figure 4.25).

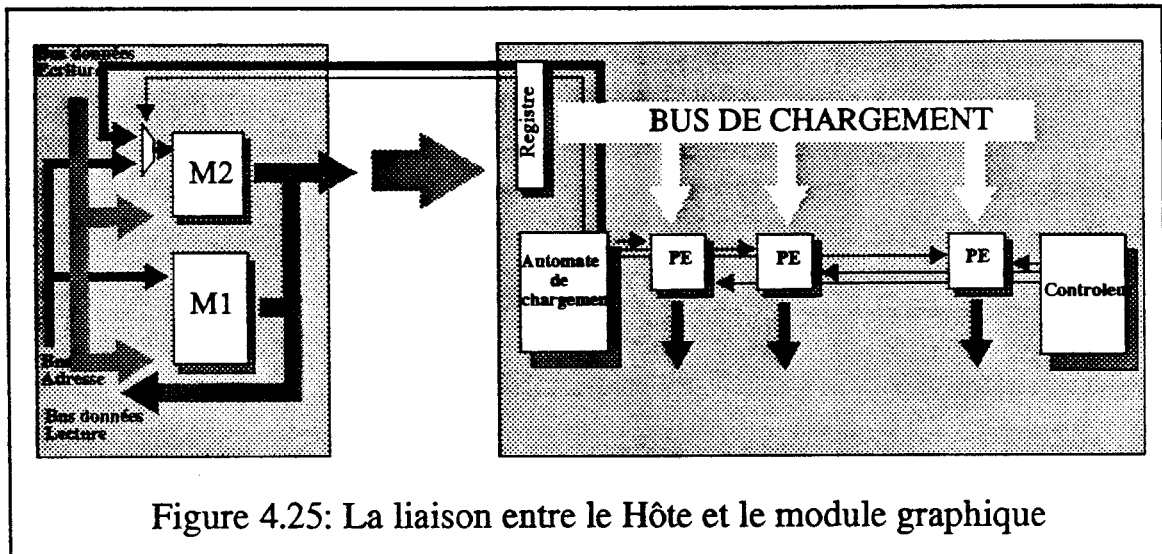


Figure 4.25: La liaison entre le Hôte et le module graphique

IV.4.6 Schéma d'implantation

La figure 4.26 présente l'organisation physique du Module Graphique.

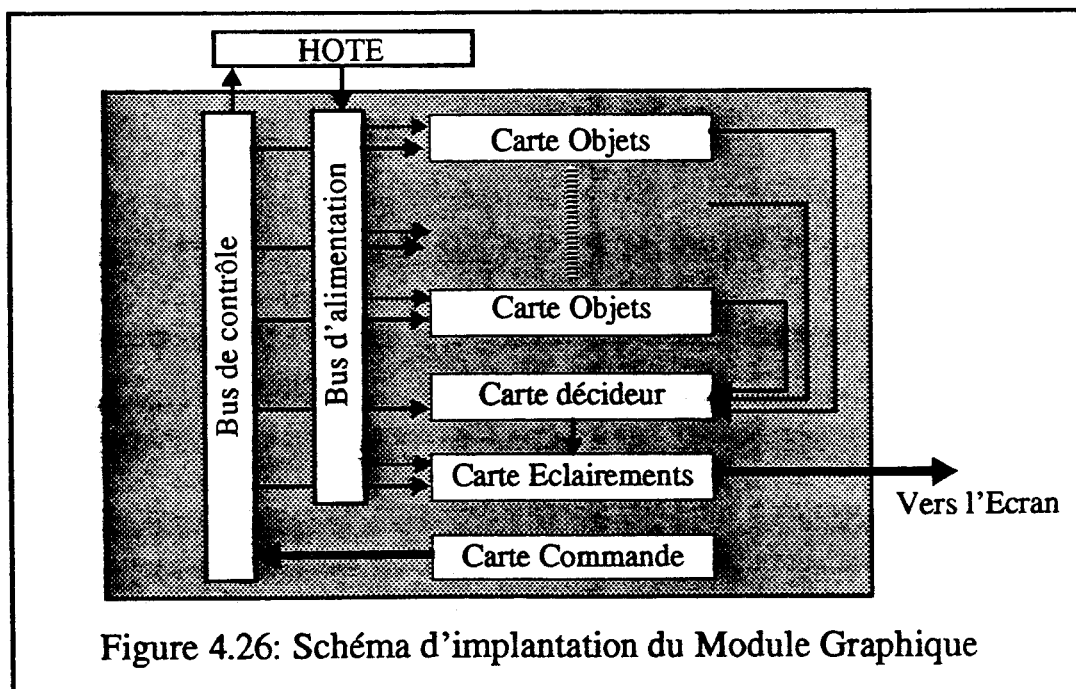


Figure 4.26: Schéma d'implantation du Module Graphique

Une carte objet contient non seulement un certain nombre de Processeurs Objets mais aussi la partie du décideur correspondant. C'est le nombre de Processeurs Objets intégrés par cartes qui déterminera les performances de la machine.

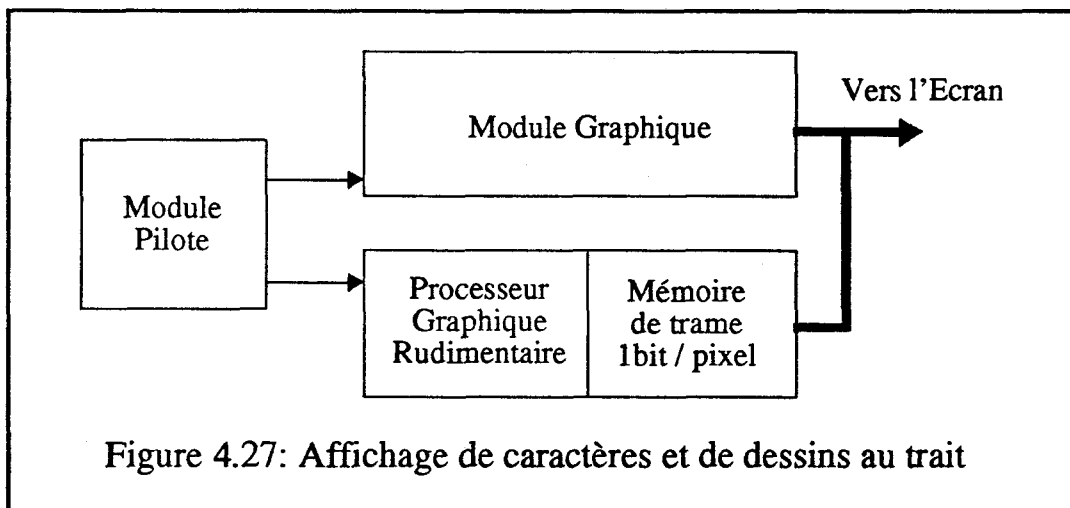
Sur le schéma précédent, apparaît une difficulté majeure: comment réaliser les liaisons entre toutes les cartes Objets et la carte Décideur ? Une réponse envisageable est de supprimer la carte décideur et de la remplacer par un fond de panier intelligent réalisant le regroupement des résultats fournies par les cartes Objets de façon distribuée.

IV.4.7 La mémoire de trame

La machine I.M.O.G.E.N.E permet de ne pas utiliser de mémoire de trame. Cependant ajouter une mémoire de trame peut être intéressant pour deux configurations, que nous allons présenter maintenant.

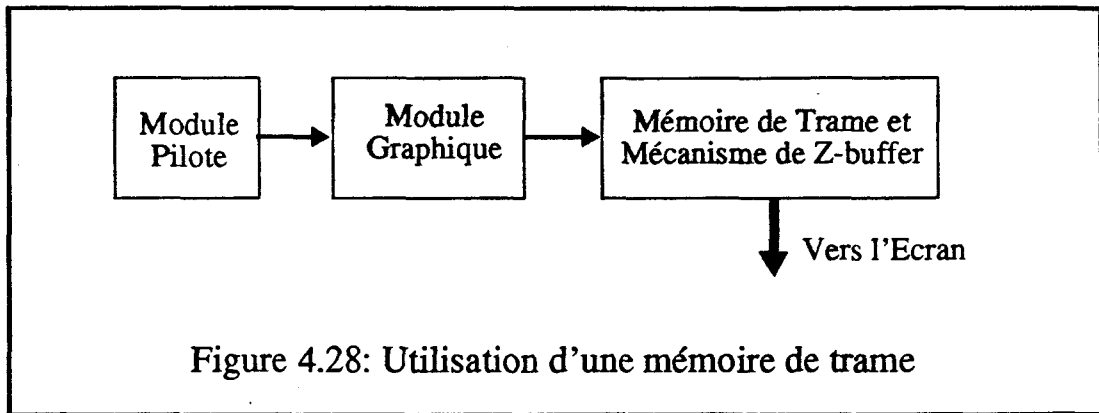
IV.4.7.1 Affichage de caractères et dessins au trait

Les Processeurs Objets ne permettent pas de visualiser des caractères ou de faire du dessin au trait. Nous pensons donc ajouter, en parallèle avec le module graphique, une mémoire de trame un bit par pixel et un processeur graphique rudimentaire effectuant ces tâches (voir Figure 4.27). La mémoire de trame est parcourue au rythme de travail du Module Graphique. Les valeurs qu'elle fournit sont superposées à l'image.



IV.4.7.2 Augmentation du nombre d'objets visualisables

Il est possible de ne pas afficher directement les valeurs fournies par le module graphique, mais de les stocker dans une mémoire de trame. En utilisant un mécanisme de Z-buffer sur cette mémoire, on peut visualiser une scène comprenant plus d'objets qu'il y a de Processeurs Objets dans le Module Graphique (voir Figure 4.28). Bien évidemment l'affichage n'est plus temps réel.



IV.4.8 Performances

Les performances d'I.M.O.G.E.N.E sont très facile à établir. Elle permet d'afficher un nombre d'objets égal au nombre de Processeurs Objets intégrés dans la machine, 50 ou 25 fois par seconde (selon que le balayage est ou non entrelacé).

Par contre la comparer aux autres machines graphiques est difficile. Toutes les performances sont données en polygones par seconde. Ces performances sont, le plus souvent, fonction de la taille (en nombre de pixels) des polygones. Or notre proposition affiche des primitives graphiques de plus haut niveau (sphère, cylindre, quadrique...) avec un temps constant quelle que soit leur taille.

Déterminer le nombre de facettes qu'il faudrait pour visualiser nos primitives est quasi-impossible. Admettons qu'une sphère, ou un cylindre, nécessite en moyenne 100 facettes. Avec 200 Processeurs Objets (de type sphère ou cylindre) I.M.O.G.E.N.E a pour performance l'équivalent de $200 \times 100 \times 50 = 1.000.000$ de facettes par seconde en mode non entrelacé.

IV.5 Le Module Pilote

IV.5.1 Présentation

Nous présentons figure 4.29 l'organisation fonctionnelle du Module Pilote. Ce module est un logiciel travaillant sur un processeur à usage général (hôte). Le travail effectué sur le hôte peut se décomposer en trois parties: une unité de configuration, une unité d'animation et une unité de calculs.

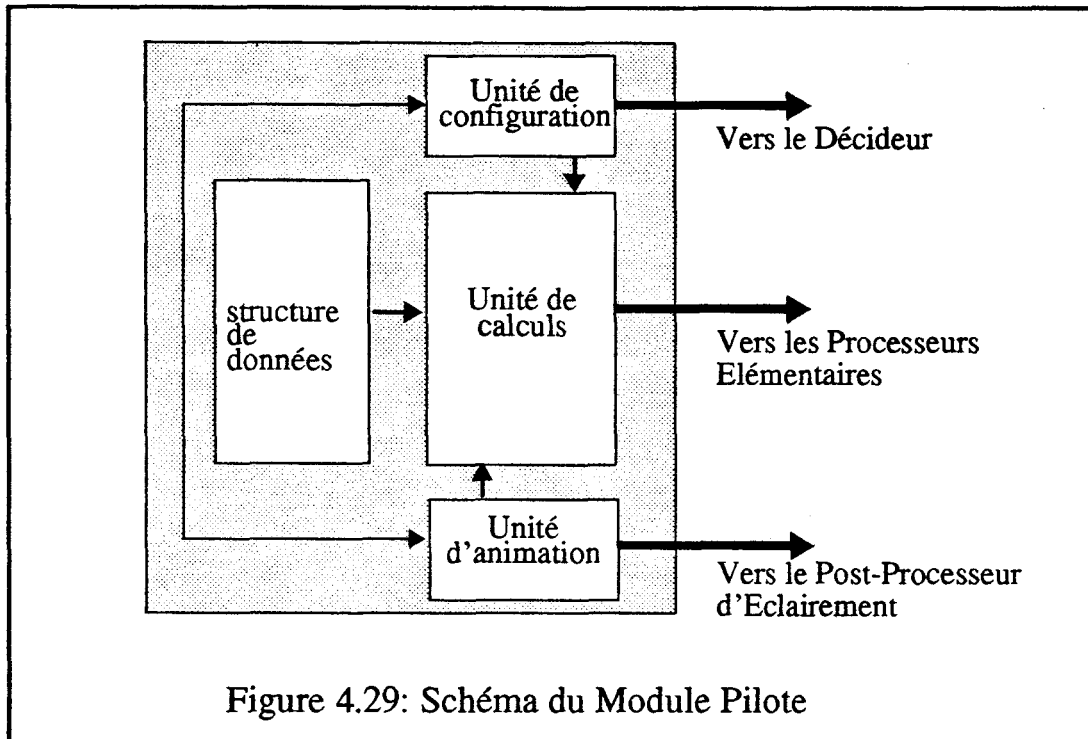


Figure 4.29: Schéma du Module Pilote

La création des objets ne sera pas étudiée dans ce travail. Nous supposons que l'ensemble des objets visualisables sont dans la structure de données. Bien évidemment les objets sont stockés sous une forme proche de celle nécessaire à la visualisation. Pour notre machine ils sont directement définis par leurs équations ou inéquations en trois dimensions. Avec nos concepts, utiliser les Brep (Boundary Representation) ou l'énumération spatiale n'aurait pas de sens. A terme la machine I.M.O.G.E.N.E nécessitera d'utiliser un modéleur dont les primitives graphiques seront adaptées aux Processeurs Objets réalisés.

Nous allons maintenant décrire les différentes unités composant le Module Pilote et leur fonctionnement pour animer en temps réel.

IV.5.2 L'unité de configuration

Cette unité a en charge les tâches suivantes:

- L'affectation des différentes expressions du premier ou second degré aux processeurs élémentaires

- La programmation du décideur dans le cas où certains noeuds du décideur peuvent avoir plusieurs fonctions.
- La programmation des processeurs objets si ceux-ci peuvent traiter des objets de plusieurs types.

Elle est commandée par l'utilisateur qui construit la scène, c'est-à-dire qui choisit les objets qu'il veut visualiser et éventuellement par l'unité d'animation qui détermine parmi les objets de la scène ceux qui sont dans le champ de vision. Quand le nombre d'objets de la scène est supérieur au nombre de processeurs objets, il est impossible de visualiser tous les objets dans une même vue.

Cette unité ne pourra pas faire l'intégralité des affectations et de la programmation du module graphique en temps réel (pendant le retour trame), elle travaillera donc en prétraitement de la visualisation. Effectuer un nombre limité d'opérations est possible en cours de visualisation, par exemple remplacer un objet par un autre de même type dans un Processeur Objet.

IV.5.3 L'unité d'animation

Cette unité calcule la position de tous les objets et de toutes les sources lumineuses de la scène en cours de visualisation.

Les informations, à partir desquelles elle travaille, peuvent avoir deux origines:

- Une interface qui lui fournit, au cours de l'animation, la position de l'observateur, les déplacements ou déformations des différents objets et les modifications des conditions d'éclairage (par exemple les commandes d'un simulateur).
- Une base de données où sont prédéfinis les différents mouvements qui régiront l'animation.

L'unité d'animation fournit à l'unité de calculs, les positions des objets, et au processeur d'éclairage les caractéristiques des sources lumineuses. Elle effectue un calcul tous les vingt-cinquième de seconde.

L'essentiel de son travail est de calculer des positions ce qui revient à effectuer des changements de repère par opérations matricielles.

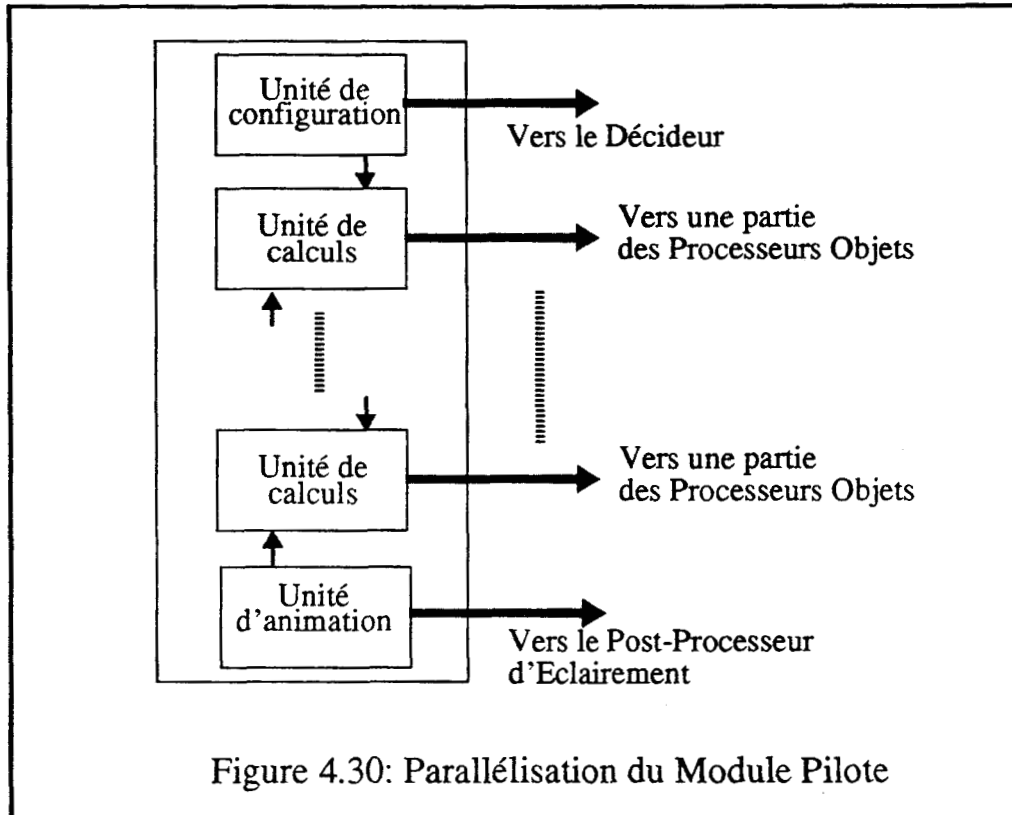
IV.5.4 L'unité de calculs

Pour chaque objet, à partir de ses caractéristiques prises dans la base de donnée et de sa position fournie par l'unité d'animation, l'unité de calculs détermine les coefficients de toutes les expressions du premier et du second degré qui permettent de visualiser cet objet. Elle envoie ces séries de coefficients aux processeurs élémentaires choisis par l'unité de configuration. Cet envoi se fait durant le retour trame.

Cette unité a un volume très important de calculs à effectuer et un débit élevé de communication avec le module graphique. Pour une sphère elle doit calculer et envoyer 24 coefficients 25 fois par seconde. A partir d'un certain nombre de processeurs élémentaires il sera nécessaire de paralléliser cette unité.

IV.5.5 Parallélisation du Module Hôte

Il est tout à fait possible d'utiliser plusieurs processeurs dans le module pilote. Un processeur fait office d'unité de configuration, un second d'unité d'animation. L'unité de calcul est répartie sur les processeurs restants. Chacun de ces processeurs est en communication avec une fraction des processeurs objets, et effectue les calculs pour les objets correspondants. Ainsi la liaison entre le Module Pilote et le Module Graphique est parallélisée (voir Figure 4.30). Ceci sans introduire aucun conflit d'accès, les différentes voies de communication étant indépendantes.



Conclusion

Le travail que nous venons de présenter est une étude de faisabilité d'un module graphique ayant un processeur par objet graphique et aucune mémoire d'image. Cette thèse est le premier élément d'I.M.O.G.E.N.E. Elle situe notre proposition parmi les machines graphiques existantes et précise les spécifications à partir desquelles nous poursuivons ce projet.

Résumons les principales originalités de notre proposition:

- Nous proposons de définir un module graphique ayant un processeur par primitive graphique mais pas de mémoire de trame. Tous les processeurs fonctionnent au rythme du balayage de l'écran.
- Nous avons défini de nouvelles primitives graphiques de plus haut niveau que les facettes polygonales utilisées par tous les autres modules graphiques.
- Nous traitons les interactions entre objets après la conversion des objets en pixels. Ce choix permet d'utiliser un parallélisme massif et d'effectuer d'autres opérations inter-objets que l'élimination des parties cachées.

Notre approche impose d'utiliser une quantité de matériel considérable. Nous avons montré, au cours de cette thèse, que c'est la seule solution pour arriver à animer des images avec possibilité d'interaction en cours de visualisation.

Le projet I.M.O.G.E.N.E se développe maintenant suivant trois axes:

- La réalisation des processeurs objets et l'organisation physique du décideur. Nous nous orientons vers la réalisation d'un processeur objet construit avec des processeurs élémentaires PEQ décrits au paragraphe IV.1.8.
- L'étude du post-processeur d'éclairage permettant de prendre en compte les réflexions diffuses et spéculaires pour plusieurs sources lumineuses.
- L'étude des problèmes de modélisation. Cette étude a pour objectifs:
 - de déterminer les séries de coefficients à fournir à chaque processeur élémentaire du module graphique.
 - de définir de nouveaux processeurs objets.
 - d'étudier les interactions entre objets.

Nous espérons avoir, dans deux ans, un prototype complet (logiciel et matériel) permettant de visualiser quelques objets au rythme de 25 images par seconde. Seule une intégration des processeurs objets en full-custom permettra de réaliser un module graphique affichant un grand nombre d'objets.

Le développement du projet I.M.O.G.E.N.E. m'a conduit à envisager les perspectives suivantes:

Il serait intéressant de faire une analyse du parallélisme matériel utilisable avec les machines exploitant les méthodes globales d'éclairage comme le Lancer de rayon et la Radiosité, ceci en utilisant la grille définie dans cette thèse. En effet, les architectures utilisant des processeurs dédiés et un parallélisme massif semble prometteuses, vu les volumes énormes de calcul que ces méthodes nécessitent. De plus une telle analyse théorique permettrait de comparer leurs résultats à ceux des multiprocesseurs à usages généraux sur lesquels on commence à implémenter des algorithmes de visualisation d'images de synthèse.

Une seconde voie à explorer est la recherche de primitives graphiques autres que les facettes planes. Dans un premier temps, il faudrait se poser la question suivante: dans les modélisations actuellement utilisées (surfaces de Bézier ou B-splines, Arbres C.S.G. ou Constructive Solid Geometry), quels sont les paramètres liés au rendu par facettes, peut-on à partir de ces modèles afficher d'autres primitives que la facette plane? Ensuite seulement on pourra déterminer une alternative aux facettes planes qui permettra d'augmenter la qualité des images tout en diminuant le nombre d'objets élémentaires constituant les scènes. Le travail à effectuer dans ce domaine doit commencer par l'étude des fondements mathématiques de la modélisation et il remettra probablement en cause les méthodes utilisées en C.A.O. depuis son origine.

V COMPLEMENTS

Nous allons, pour terminer ce travail, construire trois index: un sur les machines citées, un sur les algorithmes présentés et un des mots-clés. La bibliographie clôt cette définition du projet I.M.O.G.E.N.E.

V.1 Index des Machines citées

Nom de la Machine (constructeur)	Références bibliographiques	Paragraphe
CAP	[SatoIS85]	II.1.8
CM ²	[Cleary86]	I.2.2.2
DN 10000 (Apollo)	[Voorhi89][KirkVo90]	II.1.6
Expert	[NiimIM84]	II.1.9
Geometry Engine	[Clark82]	II.1.1
GS1000, GS2000 (Stellar)	[ApgaBM89]	II.1.5
GSP-NVS (Schlumberger)	[DeerWS88]	II.1.11
GX 4000 IMU (Raster Technologies)	[DenaRT88][Torbor87]	II.1.10
IRIS 4D/240 GTX (Silicon Graphics)	[Akeley89][AkelJe88]	II.1.4
Pixel Machine (AT&T Bell)	[PotmHo89][PotmMH89]	II.1.7
Pixel-Planes	[FuchPo81][FuchGH85][EyleAF88] [FuchPE89a][GoldFu86][GoldHF86] [GoldMT89][FuchPE89b]	II.1.12 II.1.13
PRC (Hewlett-Packard)	[SwanTh86]	II.1.2
PROOF	[Schnei88][SchnCl89][Clauss89] [Schnei90][Clauss90]	II.1.11
Ray Casting Machine	[KedeEl89]	IV.1.8
RC ²	[AtamML91]	I.2.2.2
SAGE (IBM)	[GharGH88]	II.1.3
Titan (Ardent)	[Borden89]	II.1.6
Voxar	[PitoCD89][CaubDG88]	I.2.2.2
Weinberg	[Weinbe81]	II.1.11

V.2 Index des Algorithmes décrits

V.2.1 Eclairage

Interpolation de Gouraud	[Gourau71]	III.1.2.2.1
Interpolation de Phong	[Phong75]	III.1.2.2.2
Loi de Lambert		III.1.2.1.2
Modèle de Phong	[Phong75]	III.1.2.1.3

V.2.2 Elimination parties cachées

Lignes de Balayage	[Boukni70][Watkin70]	III.1.1.1.3
Listes de Priorité	[NeweNS72][FuchAG83]	III.1.1.1.2
Par subdivision	[Warnoc68][Catmul75][Griffi75]	III.1.1.1.1
Z-buffer	[FoleVF90]	III.1.1.1.4

V.2.3 Ombres portées

Découpage géométrique	[Appel68][AtheWG78]	III.1.1.2.1
Méthode de Crow	[Crow77][Berger86]	III.1.1.2.2
Méthode de Williams	[Willia78]	III.1.1.2.3

V.2.4 Visualisation directe d'arbres C.S.G.

Algorithme d'Atherton	[Athert83]	III.1.1.4.1
Algorithme de Goldfeather	[GoldFH86][GoldMT89]	III.1.1.4.4
Algorithme de Jansen	[Jansen86]	III.1.1.4.3
Algorithme de Rossignac et Requicha	[RossRe86]	III.1.1.4.2

V.3 Index des mots-clés

Accélération	I.3.3.3			
Algorithmes	II.2.3	III.1	V.2	
Anti-aliasage	III.1.5	IV.4.3		
Approche objet	I.3.2	II.2.2	III.2.3	III.3
Approche pixel	I.3.2	II.2.2	III.2.2	
Balayage de trame	I.2.4.2	III.2.3.1		
Communications	II.2.6	III.2.4.3	IV.5.5	
Constructive Solid Geometry	III.1.1.4	IV.3.5		
Conversion des objets en pixels	I.2.1	II.2.4	III.1.3	
Eclairage	II.2.3.3	III.1.2		
Ecrans	I.2.4.2			
Efficacité	I.3.3.3			
Elimination des parties cachées	III.1.1.1	IV.3.2		
Filtrage	III.1.5			
Granularité du parallélisme	I.3.3.1	III.2.1		
Interpolation	III.1.1.2			
Lancer de rayons	I.2.2.1	I.2.2.2		
Objets transparents	III.1.1.3	IV.3.3		
Ombres portées	III.1.1.2	IV.3.4		
Opérations inter-objets	III.1.1	IV.3.2		
Organisation des machines	I.2.3	II.2.1	III.1.3	
Parallélisme	I.3.2	II.2.2		
Parallélisme massif	I.3.3.1	III.2.1		
Performances des machines	II.2.5	III.2.4.2	IV.4.8	V.1
Pipe-line	I.3.2	II.2.2		
Primitives graphiques	III.1.6	IV.2		
Processus de visualisation	I.2.1	I.2.3	I.3.2	III.1.3
Quadrique	IV.1.8	IV.2.6.2		
Matériel	I.2.3	II.2.1	IV	
Mémoire de trame	I.2.3	III.1.4	IV.4.7	
MIMD	I.3.3.2	II.2.1		
Module graphique	I.2.3	III.2.4.3	III.3.1	IV.5
Module pilote	I.2.3	III.2.4.3	III.3.2	IV.4
Radiosité	I.2.2.3			
Ratio-coopération	I.3.3.3	III.2.1.2		
Rendement	I.3.3.3	III.2.1.2		
SIMD	I.3.3.2	II.2.1		
Suréchantillonnage	III.1.5			
Temps réel	I.1.2	III.3		
Tubes à Rayons Cathodiques	I.2.4.1			

V.4 BIBLIOGRAPHIE

- [AtamML91] ATAMENIA A., MERIAUX M. and al.
"A Cellular Architecture for Ray Tracing"
Advances in Computer Graphics Hardware V, Springer Verlag, 1991, to appear in 1991
- [Akeley89] AKELEY K.
"The Silicon Graphics 4D/240GTX Superworkstation"
IEEE Computer Graphics and Applications, vol. 9 num. 4, july 89, pp 71-83
- [AkeJe88] AKELEY K. and JERMOLUK T.
"High-Performance Polygon Rendering"
ACM Computer Graphics, vol. 22 num. 4, august 1988, pp 239-246
- [ApgaBM88] APGAR B., BERSACK B. and MAMMEN A.
"A Display System for the Stellar Graphics Supercomputer Model GS1000"
ACM Computer Graphics, vol. 22 num. 4, august 1988, pp 255-262
- [Appel 68] APPEL A.
"Some Techniques for Shading Machine Rendering of Solids"
Proceeding SJCC 1968, Thompson Books, Washington, 1968, pp37-45
- [Athert83] ATHERTON P.
"A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry"
ACM Computer Graphics, vol. 17 num. 3, july 1983, pp 73-82
- [AtheWG78] ATHERTON P., WEILER K. and GREENBERG D.
"Polygon Shadow Generation"
ACM Computer Graphics, vol. 12 num. 3, july 1978, pp 275-281
- [BadoPr91] BADOUEL D. and PRIOL T.
"An Efficient Parallel Ray Tracing Scheme for Highly Parallel Architecture"
Advances in Computer Graphics Hardware V, Springer Verlag, 1991, to appear in 1991
- [Berger86] BERGERON P.
"A General Version of Crow's Shadow Volumes"
IEEE Computer Graphics and Applications, vol. 6 num. 5, september 86, pp 17-28
- [BishWe86] BISHOP G. and WEINER D.
"Fast Phong Shading"
ACM Computer Graphics, vol. 20 num. 4, august 1986, pp 103-106
- [Borden89] BORDEN B.
"Graphics Processing on a Graphics Supercomputer"
IEEE Computer Graphics and Applications, vol. 9 num. 4, july 89, pp 56-62
- [BonaPr88] BOUATOUCH K. and PRIOL T.
"Parallel Space Tracing: An Experience on an iPSC Hypercube"
Proc. of Computer Graphics International'88, Springer-Verlag, 1988, pp170-188

- [Boukni70] BOUKNIGHT W.
"A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Representations"
 Communication ACM, vol. 13, 1970, pp 527-536
- [BourNo90] BOURRUST I. et NORMAND F.
"Etude et Réalisation du Processeur Élémentaire d'I.M.O.G.E.N.E"
 Mémoire d'Ingénieur, EUDIL Lille, mars 1990
- [CaubDG88] CAUBET R., DUTHEN Y. and GAILDRAT V.
"VOXAR: a Tridimensional Architecture for Fast Realistic Image Synthesis"
 Proc. of Computer Graphics International '88, Springer-Verlag, 1988, pp135-149
- [Catmul75] CATMULL E.
"Computer Display of Curved Surfaces"
 Proc. of IEEE Conf. Comput. Graphics Pattern Recognition Data Struct., may 75, p 11
- [ChaiMC90] CHAILLOU C., MERIAUX M., CORDONNIER V. and KARP F. S.
"A Real Time Image Generator"
 Proc. Eighth IASTED International Symposium Applied Informatics, ACTA PRESS, 1990, pp 140-143
- [ChaiKM90a] CHAILLOU C., KARP F. S., MERIAUX M. and CORDONNIER V.
"Parallel Architecture for Real-Time Image Generation"
 Proc. VAPP IV-CONPAR 90, vol on Special Technical Contributions, 1990, pp C45-C48
- [ChaiKM90b] CHAILLOU C., KARP F. S., MERIAUX M. and CORDONNIER V.
"I.M.O.G.E.N.E: An Object Oriented Architecture for Real Time Image Generation"
 Proc. ISMM Int. Conf. on Parallel and Distributed Computing and Systems, New York, 1990, to appear
- [ChaiKM91] CHAILLOU C., KARP F. S. and MERIAUX M.
"I.M.O.G.E.N.E: A Solution to the Real Time Animation Problem"
 Advances in Computer Graphics Hardware V, Springer Verlag, 1991, to appear in 1991
- [ChaiKN90a] CHAILLOU C., KARP F. S., NIYIRI E. et MERIAUX M.
"I.M.O.G.E.N.E: Images au Moyen d'Objets Générés par Expressions Numériques"
 Actes des Journées Graphiques GROPLAN 90, E. M. Saint Etienne, Novembre 1990
- [ChaiKN90b] CHAILLOU C., KARP F. S., NIYIRI E. et MERIAUX M.
"Présentation du Projet I.M.O.G.E.N.E"
 Rapport de Recherche num. 87, LIFL USTLFA, Décembre 1990
- [Clark82] CLARK J.
"The Geometry Engine: A VLSI Geometry System for Graphics"
 ACM Computer Graphics, vol. 16 num.3, july 1982, pp127-133
- [Clause89] CLAUSSEN U.
"On Reducing the Phong Shading Method"
 Proceeding Eurographics '89, Elsevier Science Publishers B. V., pp 333-344

- [Clausse91] CLAUSSEN U.
"Real Time Phong Shading"
 Advances in Computer Graphics Hardware V, Springer Verlag, 1991, to appear in 1991
- [Clausse90] CLAUSSEN U.
"Verfahren zur Schnellen Beleuchtungs- und Schattierungsberechnung"
 Phd Thesis, Universität Tübingen, 1990
- [Cleary86] CLEARY J.G. and al.
"Multiprocessor Ray Casting"
 Computer Graphics Forum, vol. 5 num. 1, march 86, pp3-12
- [CohenGr85] COHEN M. and GREENBERG D.
"The Hemi-cube a Radiosity Solution for Complex Environments"
 ACM Computer Graphics, vol. 9 num.3, july 1985, pp31-40
- [CordMe90] CORDONNIER V. and MERIAUX M.
"Image Display from Multiprocessors"
 Proc. SID International Symposium, Digest of Technical Papers, vol. 21, 1990, pp49-52
- [Crow77] CROW F.
"Shadow Algorithms for Computer Graphics"
 ACM Computer Graphics, vol. 11 num.3, july 1977, pp242-248
- [Crow81] CROW F.
"A Comparison of Antialiasing Techniques"
 IEEE Computer Graphics and Applications, vol. 1 num. 1, january 81, pp 40-48
- [DeerWS88] DEERING M., WINNER S., SCHEDIWIY B. and al.
"The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics"
 ACM Computer Graphics, vol. 22 num. 4, august 1988, pp 21-30
- [Degran89] DEGRANDE S.
"Architecture à Transputers à Vidéo Répartie"
 Mémoire de DEA, USTL Lille, 1989
- [Degran90] DEGRANDE S.
"Réalisation d'un P. E. Second Degré d'I.M.O.G.E.N.E avec le Logiciel SOLO14000"
 Rapport de Recherche numéro 90, LIFL Lille, novembre 90
- [DenaRT88] DENAULT D., RYHERD E., TORBORG j. and al.
"VLSI Drawing Processor Utilizing Multiple Parallel Scan-Line Processors"
 Advances in Computer Graphics Hardware II, Springer Verlag, 1988, pp 167-182
- [EyleAF88] EYLES J., AUSTIN J., FUCHS H. and al
"Pixel-Planes 4: A Summary"
 Advances in Computer Graphics Hardware II, Springer Verlag, 1988, pp 183-208
- [Flynn72] FLYNN M. J.
"Some Computer Organizations and their Effectiveness"
 IEEE Transaction on Computers, vol. C21 num. 9, september 72, pp 948-960

- [FoleVF90] FOLEY J., VAN DAM A., FEINER S. and HUGHES J.
"Computer Graphics: Principles and Practice (second edition)"
 The systems Programming Series, Addison Wesley 12110, 1990
- [FuchAG83] FUCHS H., ABRAM G. and GRANT E.
"Near Real-Time Shaded Display of Rigid Objects"
 ACM Computer Graphics, vol. 17 num. 3, july 1983, pp 65-72
- [FuchGH85] FUCHS H., GOLDFEATHER J., HULTQUIST J. and al
"Fast Spheres, Shadows, Textures, Transparencies and Image Enhancements in Pixel-Planes"
 ACM Computer Graphics, vol. 19 num. 3, july 85, pp 111-120
- [FuchPE89a] FUCHS H., POULTON J., EYLES J. and GREER T.
"Coarse-grain and Fine-grain Parallelism in the Next Generation Pixel-Planes Graphics System"
 Parallel Processing for Computer Vision and Display, Addison Wesley ,pp 241-253
- [FuchPE89b] FUCHS H., POULTON J., EYLES J. and al
"Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories"
 ACM Computer Graphics, vol. 23 num. 3, july 89, pp 79-88
- [FuchPo81] FUCHS H. and POULTON J.
"Pixel-Planes: A VLSI-Oriented Design for a Raster Graphics Engine"
 VLSI Design, Troisième Trimestre 81, pp20-28
- [GharGH88] GHARACHORLOO N., GUPTA S., HOKENEK E. and al
"Subnanosecond Pixel Rendering with Million Transistor Chips"
 ACM Computer Graphics, vol. 22 num.4, august 1988, pp41-49
- [GharGS89] GHARACHORLOO N., GUPTA S., SPROULL R. and al
"A Characterization of Ten Rasterization Techniques"
 ACM Computer Graphics, vol. 23 num. 3, july 89, pp 355-368
- [GoldFu86] GOLDFEATHER J. and FUCHS H.
"Quadratic Surface Rendering on a Logic-Enhanced Frame-Buffer Memory"
 IEEE Computer Graphics and Applications, vol. 6 num. 1, january 86, pp 48-59
- [GoldHF86] GOLDFEATHER J., HULTQUIST J. and FUCHS H.
"Fast Constructive Geometry Display in the Pixel-Powers Graphics System"
 ACM Computer Graphics, vol. 20 num. 4, august 1986, pp 107-116
- [GoldMT89] GOLDFEATHER J., MOLNAR S., TURK G. and FUCHS H.
"Near Real-Time CSG Rendering Using Tree Normalisation and Geometric Pruning"
 IEEE Computer Graphics and Applications, vol. 9 num. 3, may 89, pp 20-28
- [GoraTG84] GORAL C., TORRANCE K., GREENBERG D. and BATTAILE B.
"Modeling the Interaction of Light between Diffuse Surfaces"
 ACM Computer Graphics, vol. 18 num. 3, july 1984, pp 213-222
- [Gourau71] GOURAUD H.
"Continuous Shading of Curved Surfaces"
 IEEE Transaction on Computers, vol. C-20 num. 6, june 1971, pp 623-629

- [Griffi75] GRIFFITHS J.
"A Data Structure for the Elimination of Hidden Surface by Patch Division"
Computer Aid Design, vol. 7 num.3, 1975, pp 171-178
- [GrimKB89] GRIMES J., KOHN L. and BHARADHWAJ R.
"The Intel i860 64-Bit Processor: A General-Purpose CPU with 3D Graphics Capabilities"
IEEE Computer Graphics and Applications, vol. 9 num. 4, july 89, pp 85-94
- [Hashem90] HASHEMIAN R.
"Square Rooting Algorithms for Integer and Floating-Point Numbers"
IEEE Transaction on Computers, vol. 39 num 8, August 1990, pp 1025-1029
- [Jansen85] JANSEN F.
"A CSG List Priority Hidden Surface Algorithm"
Proceeding Eurographics '85, Elsevier Science Publishers B.V., pp 51-62
- [Jansen86] JANSEN F.
"A Pixel-Parallel Hidden Surface Algorithm for Constructive Solid Geometry"
Proceeding Eurographics '86, Elsevier Science Publishers B.V., pp 29-40
- [Jansen87] JANSEN F.
"CSG Hidden Surface Algorithms for VLSI Hardware Systems"
"Advance on Graphics Hardware I", Eurographics Seminars, Springer-Verlag, 1987, pp 75-82
- [Karpf89] KARPf S.
"Elément du Projet I.M.O.G.E.N.E, Etude et Réalisation du Processeur Elémentaire"
Mémoire de DEA, LIFL Lille, septembre 1989
- [KayGre79] KAY D.S. and GREENBERG D.
"Transparency for Computer Synthesized Images"
ACM Computer Graphics, vol. 13 num. 2, august 79, pp158-164
- [KedeEl89] KEDEM G. and ELLIS J.L.
"the Ray Casting Machine"
Parallel Processing for Computer Vision and Display, Addison Wesley, pp 378-401
- [KirkVo90] KIRK D. and VOORHIES D.
"The Rendering Architecture of the DN10000VS"
ACM Computer Graphics, vol. 24 num. 4, august 90, pp 299-307
- [KuijB189] KUIJK A. and BLAKE E.
"Faster Phong Shading via Angular Interpolation"
Computer Graphics Forum, vol. 8 num. 4, december 89, pp 315-324
- [Lepret89] LEPRETRE E.
"Algorithmique Parallèle et Architectures Spécialisées pour la Synthèse d'Images"
Thèse de Doctorat, USTLFA Lille, 1989
- [Mammen89] MAMMEN A.
"Transparency and Antialiasing Algorithms Implemented with Virtual Pixel Maps Technique"
IEEE Computer Graphics and Applications, vol. 9 num. 4, july 89, pp 43-55

- [Martin82] MARTINEZ F.
"Vers une Approche Systématique de la Synthèse d'Image: Aspects Logiciel et Matériel"
 Thèse d'Etat, INPG Grenoble, 1982
- [Martin86] MARTINEZ F.
"Architectures Spécialisées et Technologie en Synthèse d'Image"
 Actes du Second Colloque Image, Avril 86, pp 353-359
- [Meriau84] MERIAUX M.
"Contribution à l'Imagerie Informatique: Aspects Algorithmiques et Architecturaux"
 Thèse d'Etat, USTLFA Lille, 1984
- [NeweNS72] NEWELL M., NEWELL R. and SANCHA T.
"A New Approach to the Shaded Picture Problem"
 Proc. ACM Natl. Conf., 1972, pp 443-450
- [NiimIM84] NIIMI H., IMAI Y., MURAKAMI M. and al
"A Parallel Processor System for Three-Dimensional Color Graphics"
 ACM Computer Graphics, vol. 18 num. 3, july 84, pp 67-76
- [Nyiri90] NYIRI E.
"Modélisation et Simulation d'Objets 3D à l'Aide d'Expressions du Second Degré"
 Mémoire de DEA, LIFL Lille, septembre 1990
- [Phong75] PHONG B. T.
"Illumination for Computer Generated Pictures"
 Communications ACM, vol. 18 num. 18, june 1975, pp 311-317
- [PitoCD89] PITOT P., CAUBET R., DUTHEN Y. et GAILDRAT V.
"Le Suivi Analytique de Rayons: un Algorithme Incrémental Rapide pour la Machine VOXAR"
 Actes de MICAD 89, 1989, pp 653-664
- [PotmHo89] POTMESIL M. and HOFFERT E.
"The Pixel Machine: A Parallel Image Computer"
 ACM Computer Graphics, vol. 23 num. 3, july 89, pp 69-78
- [PotmMH89] POTMESIL M., McMILLAN L., HOFFERT E. and al
"A Parallel Image Computer with a Distributed Frame Buffer: System Architecture and Programming"
 Proceeding Eurographics'89, Elsevier Science Publishers B.V., pp 197-208
- [Priol89] PRIOL T.
"Lancer de Rayon sur des Architectures Parallèles: Etude et Mise en Oeuvre"
 Thèse de Doctorat, Université de Rennes I, 1989
- [Rogers85] ROGERS D. F.
"Procedural Elements for Computer Graphics"
 Mac Graw-Hill, 1985, chapter 4
- [RossRe86] ROSSIGNAC J. and REQUICHA A.
"Depth-Buffering Display Techniques for Constructive Solid Geometry"
 IEEE Computer Graphics and Applications, vol. 6 num. 5, september 86, pp 29-39

- [RossVo89] ROSSIGNAC J and VOELCKER H.
"Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection, and Shading Algorithms"
 ACM Transaction on Graphics, vol. 8 num 1, january 1989, page 51-87
- [Roth82] ROTH D. S.
"Ray Casting for Modeling Solids"
 Computer Graphics and Image Processing, vol. 18 num. 2, february 82, pp51-55
- [SatoIS85] SATO H., ISHII M., SATO k. and al
"Fast Image Generation of Constructive Solid Geometry Using a Cellular Array Processor"
 ACM Computer Graphics, vol.19 num. 3, july 85, pp 95-102
- [SchnCI89] SCHNEIDER B.O. and CLAUSSEN U.
"PROOF; An Architecture for Rendering in Object Space"
 Advances in Computer Graphics Hardware II, Springer Verlag, 1989
- [Schnei88] SCHNEIDER B.O.
"A Processor for an Object-Oriented Rendering System"
 Computer Graphics Forum, num. 7, 1988, pp301-310
- [Schnei90] SCHNEIDER B.O.
"Eine Objektorientierte Architektur für Hochleistungs-Display-Prozessoren"
 Phd Thesis, Universität Tübingen, 1990
- [SuthSS74] SUTHERLAND I., SPROULL R. and SCHUMACKER R.
"A Characterization of Ten Hidden-Surface Algorithms"
 Computing Survey, vol. 6 num. 1, 1974, pp 1-55
- [SwanTh86] SWANSON R. and THAYER L.
"A Fast Shaded-Polygon Renderer"
 ACM Computer Graphics, vol. 20 num. 4, august 1986, pp 95-101
- [Tilove 80] TILOVE R.
"Set Membership Classification: A Unified Approach to Geometric Intersection Problems"
 IEEE Transaction on Computers, vol. C.29 num. 10, october 1980, pp 874-883
- [Torbor87] TORBORG J.
"A Parallel Processor for Graphics Arithmetic Operations"
 ACM Computer Graphics, vol. 21 num. 4, july 87, pp 197-204
- [TorrSp67] TORRANCE K. and SPARROW E.
"Theory for Off-Specular Reflection from Roughened Surfaces"
 Journal of the Optical Society of America, vol. 57, 1967, pp 1105-1114
- [Voorhi89] VOORHIES D.
"Reduced Complexity Graphics"
 IEEE Computer Graphics and Applications, vol. 9 num. 4, july 89, pp 63-70

[Warn83] WARN D.

"Lighting Control for Synthetic Images"

ACM Computer Graphics, vol. 17 num. 3, july 83, pp 13-21

[Warnoc68] WARNOCK J.

"A Hidden Line Algorithm for Half-tone Picture Representation"

University of Utah Computer Science Dept. Rep. TR 4-5, may 68, NTIS AD 761 995

[Watkin70] WATKINS G.

"A Real-Time Visible Surface Algorithm"

University of Utah Computer Science Dept. Rep., UTEC-CSC-70-101, june 70, NTIS AD 762 004

[WeilAt77] WEILER K. and ATHERTON P.

"Hidden Surface Removal Using Polygon Area Sorting"

ACM Computer Graphics, vol. 11 num. 3, july 77, pp214-222

[Weinbe81] WEINBERG R.

"Parallel Processing Image Synthesis and Anti-Aliasing"

ACM Computer Graphics, vol. 15 num. 3, august 1981, pp 55-61

[Whitte80] WHITTED T.

"An Improved Illumination Model for Shaded Display"

Communication ACM, vol. 23, 1980, pp 343-349

[Willia78] WILLIAMS L.

"Casting Curved Shadows on Curved Surfaces"

ACM Computer Graphics, vol. 12 num. 3, july 1978, pp 270-274



Résumé

Le Projet I.M.O.G.E.N.E consiste en la définition d'une machine de rendu d'image de synthèse en temps réel. Nous envisageons le temps réel au sens strict: l'image est intégralement recalculée à chaque trame, c'est à dire 50 fois par seconde avec un balayage non-entrelacé (moitié moins si le balayage est entrelacé). Notre approche est radicalement différente de celle des modules graphiques existants que nous présentons et classons en fonction du parallélisme qu'ils exploitent. Nous n'utilisons pas de mémoire de trame. Toutes les opérations inter-objets sont effectuées après la conversion des objets en pixels (ce choix impose des limitations que nous mettons en évidence). Nous exploitons un parallélisme massif objet, le système de rendu comprend un grand nombre de processeur-objets. Pour une image donnée, chacun de ces processeurs traite une seule primitive graphique au rythme du balayage écran, les différents pixels sont traités en pipe-line. Le processeur-objet est composé d'unités très simples (additionneurs, registres, ...) pour être aisément intégrable en VLSI. Pour limiter le nombre de processeurs-objets, nous définissons des processeurs traitant des primitives plus complexes que les facettes triangulaires, avec interpolations sur les profondeurs et l'éclairage, généralement utilisées.

Mots-Clés

Synthèse d'image, Parallélisme massif, Architecture pour le Graphique
Circuit Intégré, Approche Objet

Abstract

The goal of the I.M.O.G.E.N.E. Project is the definition of a real time graphics system. We focus on true real time display, images being computed at frame rate, i.e. 50 times a second with a non interlaced mode (half with an interlaced one). Our approach drastically differs from current graphics system that we analyse and sort according to the kind of parallelism they use. Our solution uses no frame buffer. All inter-object processing are made after the object-to-pixels conversion (this choice involves some limitations which we highlight). We uses a massive object parallelism; the graphics module is made of a large number of object-processors. Each object-processor handles one graphics primitive at pixel rate all over the screen, with pixels being processed in pipe-line. The object-processors are made of very simple units (adders, registers, ...) allowing an easy integration. In order to restrict the number of object-processors, we define more powerful graphics primitives than classical 3D triangular facets with depth and lighting interpolations.