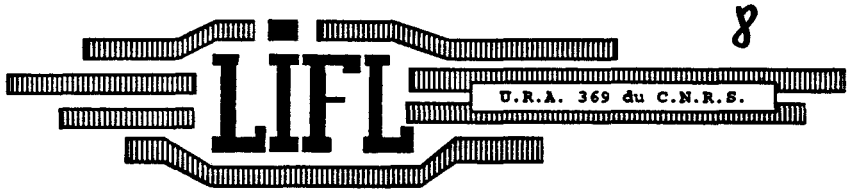
NUMERO D'ORDRE : **1054**

ANNEE : 1993



LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

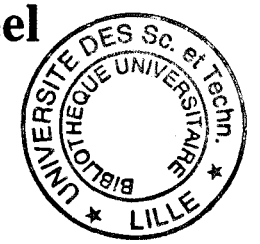
pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

Sylvain KARPF

Architectures Massivement Parallèles pour la Synthèse d'Images Temps Réel



Thèse soutenue le 26 janvier 1993, devant la commission d'examen :

Président :	V. CORDONNIER	LIFL
Directeur de Thèse	M. MERIAUX	LIFL
Rapporteurs :	M. LUCAS	Ecole Centrale de Nantes
	C. PUECH	IMAG
Examinateurs :	C. CHAILLOU	LIFL
	S. COUVET	Thomson-CSF DSI
	A. KAUFMAN	SUNY at Stony Brook
	M. WEINFELD	LIX

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.R. d'I.E.E.A. Bât M3. 59655 Villeneuve d'Ascq CEDEX
Tél. 20.43.47.24 Fax. 20.43.65.66

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE

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. PARREAU, J. LOMBARD, M. MIGEON. J. CORTOIS. A.DUBRULLE

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

M. P. LOUIS

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CHAMLEY Hervé	Géotechnique
M. CONSTANT Eugène	Electronique
M. ESCAIG Bertrand	Physique du solide
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. LABLACHE COMBIER Alain	Chimie
M. LOMBARD Jacques	Sociologie
M. MACKE Bruno	Physique moléculaire et rayonnements atmosphériques

M. MIGEON Michel	EUDIL
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. BOSCQ Denis	Probabilités
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BRASSELET Jean Paul	Géométrie et topologie
M. BREZINSKI Claude	Analyse numérique
M. BRIDOUX Michel	Chimie Physique
M. BRUYELLE Pierre	Géographie
M. CARREZ Christian	Informatique
M. CELET Paul	Géologie générale
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. CROSNIER Yves	Electronique
Mme DACHARRY Monique	Géographie
M. DAUCHET Max	Informatique
M. DEBOURSE Jean Pierre	Gestion des entreprises
M. DEBRABANT Pierre	Géologie appliquée
M. DECLERCQ Roger	Sciences de gestion
M. DEGAUQUE Pierre	Electronique
M. DESCHEPPER Joseph	Sciences de gestion
Mme DESSAUX Odile	Spectroscopie de la réactivité chimique
M. DHAINAUT André	Biologie animale
Mme DHAINAUT Nicole	Biologie animale
M. DJAFARI Rouhani	Physique
M. DORMARD Serge	Sciences Economiques
M. DOUKHAN Jean Claude	Physique du solide
M. DUBRULLE Alain	Spectroscopie hertzienne
M. DUPOUY Jean Paul	Biologie
M. DYMENT Arthur	Mécanique
M. FOCT Jacques Jacques	Métallurgie
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie structurale
M. FRONTIER Serge	Ecologie numérique
M. GLORIEUX Pierre	Physique moléculaire et rayonnements atmosphériques
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie-Physique
M. GRANELLE Jean Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILBAULT Pierre	Physiologie animale
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. HENRY Jean Pierre	Génie mécanique
M. HERMAN Maurice	Physique spatiale
M. LACOSTE Louis	Biologie Végétale
M. LANGRAND Claude	Probabilités et statistiques

M. LATTEUX Michel	Informatique
M. LAVEINE Jean Pierre	Paléontologie
Mme LECLERCQ Ginette	Catalyse
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique atomique et moléculaire
M. LEROY Jean Marie	Spectrochimie
M. LHENAFF René	Géographie
M. LHOMME Jean	Chimie organique biologique
M. LOUAGE Françis	Electronique
M. LOUCHEUX Claude	Chimie-Physique
M. LUCQUIN Michel	Chimie physique
M. MAILLET Pierre	Sciences Economiques
M. MAROUF Nadir	Sociologie
M. MICHEAU Pierre	Mécanique des fluides
M. PAQUET Jacques	Géologie générale
M. PASZKOWSKI Stéfan	Mathématiques
M. PETIT Francis	Chimie organique
M. PORCHET Maurice	Biologie animale
M. POUZET Pierre	Modélisation - calcul scientifique
M. POVY Lucien	Automatique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. RAMAN Jean Pierre	Sciences de gestion
M. SALMER Georges	Electronique
M. SCHAMPS Joël	Spectroscopie moléculaire
Mme SCHWARZBACH Yvette	Géométrie
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
M. SLIWA Henri	Chimie organique
M. SOMME Jean	Géographie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. THIEBAULT François	Sciences de la Terre
M. THOMAS Jean Claude	Géométrie - Topologie
M. THUMERELLE Pierre	Démographie - Géographie humaine
M. TILLIEU Jacques	Physique théorique
M. TOULOTTE Jean Marc	Automatique
M. TREANTON Jean René	Sociologie du travail
M. TURRELL Georges	Spectrochimie infrarouge et raman
M. VANEECLOO Nicolas	Sciences Economiques
M. VAST Pierre	Chimie inorganique
M. VERBERT André	Biochimie
M. VERNET Philippe	Génétique
M. VIDAL Pierre	Automatique
M. WALLART Françis	Spectrochimie infrarouge et raman
M. WEINSTEIN Olivier	Analyse économique de la recherche et développement
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ABRAHAM Francis	
M. ALLAMANDO Etienne	Composants électroniques
M. ANDRIES Jean Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BALL Steven	Génétique
M. BART André	Biologie animale
M. BASSERY Louis	Génie des procédés et réactions chimiques
Mme BATTIAU Yvonne	Géographie
M. BAUSIERE Robert	Systèmes électroniques
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique atomique et moléculaire
M. BERNAGE Pascal	Physique atomique, moléculaire et du rayonnement
M. BERTHOUD Arnaud	Sciences Economiques
M. BERTRAND Hugues	Sciences Economiques
M. BERZIN Robert	Analyse
M. BISKUPSKI Gérard	Physique de l'état condensé et cristallographie
M. BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie végétale
M. BOHIN Jean Pierre	Biochimie métabolique et cellulaire
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie civil
M. BOIVIN Jean Claude	Spectrochimie
M. BOUCHER Daniel	Physique
M. BOUQUELET Stéphane	Biologie appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BROCARD Jacques	Chimie
Mme BROUSMICHE Claudine	Paléontologie
M. BUISINE Daniel	Mécanique
M. CAPURON Alfred	Biologie animale
M. CARRE François	Géographie humaine
M. CATTEAU Jean Pierre	Chimie organique
M. CAYATTE Jean Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie structurale
M. CHIVE Maurice	Composants électroniques optiques
M. COMYN Gérard	Informatique théorique
Mme CONSTANT Monique	Composants électroniques et optiques
M. COQUERY Jean Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique nucléaire et corpusculaire
M. COUTURIER Daniel	Chimie organique
M. CRAMPON Norbert	Tectolique géodynamique
M. CURGY Jean Jacques	Biologie
M. DANGOISSE Didier	Physique théorique
M. DE PARIS Jean Claude	Analyse
M. DECOSTER Didier	Composants électroniques et optiques
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean Paul	Informatique
M. DELORME Pierre	Physiologie animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
Mme DEMUYNCK Claire	Physique atomique, moléculaire et du rayonnement
M. DENEL Jacques	Informatique
M. DEPREZ Gilbert	Physique du solide - christallographie

M. LE MAROIS Henri	Vie de la firme
M. LEMOINE Yves	Biologie et physiologie végétales
M. LESCURE François	Algèbre
M. LESENNE Jacques	Systèmes électroniques
M. LOCQUENEUX Robert	Physique théorique
Mme LOPES Maria	Mathématiques
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU François	Sciences économiques
M. MAHIEU Jean Marie	Optique - Physique atomique
M. MAIZIERES Christian	Automatique
M. MANSY Jean Louis	Géologie
M. MAURISSON Patrick	Sciences Economiques
M. MERIAUX Michel	EUDIL
M. MERLIN Jean Claude	Chimie
M. MESMACQUE Gérard	Génie mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MOCHE Raymond	Modélisation,calcul scientifique,statistiques
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie organique
M. MORE Marcel	Physique de l'état condensé et cristallographie
M. MORTREUX André	Chimie organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
M. NIAY Pierre	Physique atomique,moléculaire et du rayonnement
M. NICOLE Jacques	Spectrochimie
M. NOTELET Francis	Systèmes électroniques
M. PALAVIT Gérard	Génie chimique
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. PERTUZON Emile	Physiologie animale
M. PETIT Daniel	Biologie des populations et écosystèmes
M. PLIHON Dominique	Sciences Economiques
M. PONSOLLE Louis	Chimie physique
M. POSTAIRE Jack	Informatique industrielle
M. RAMBOUR Serge	Biologie
M. RENARD Jean Pierre	Géographie humaine
M. RENARD Philippe	Sciences de gestion
M. RICHARD Alain	Biologie animale
M. RIETSCH François	Physique des polymères
M. ROBINET Jean Claude	EUDIL
M. ROGALSKI Marc	Analyse
M. ROLLAND Paul	Composants électroniques et optiques
M. ROLLET Philippe	Sciences Economiques
Mme ROUSSEL Isabelle	Géographie physique
M. ROUSSIGNOL Michel	Modélisation,calcul scientifique,statistiques
M. ROY Jean Claude	Psychophysiologie
M. SALERNO Françis	Sciences de gestion
M. SANCHOLLE Michel	Biologie et physiologie végétales
Mme SANDIG Anna Margarette	
M. SAWERYSYN Jean Pierre	Chimie physique
M. STAROSWIECKI Marcel	Informatique
M. STEEN Jean Pierre	Informatique
Mme STELLMACHER Irène	Astronomie - Météorologie
M. STERBOUL François	Informatique
M. TAILLIEZ Roger	Génie alimentaire
M. TANRE Daniel	Géométrie - Topologie
M. THERY Pierre	Systèmes électroniques
Mme TJOTTA Jacqueline	Mathématiques
M. TOURSEL Bernard	Informatique
M. TREANTON Jean René	Sociologie du travail

M. DERIEUX Jean Claude	Microbiologie
M. DERYCKE Alain	Informatique
M. DESCAMPS Marc	Physique de l'état condensé et cristallographie
M. DEVRAINNE Pierre	Chimie minérale
M. DEWAILLY Jean Michel	Géographie humaine
M. DHAMELINCOURT Paul	Chimie physique
M. DI PERSIO Jean	Physique de l'état condensé et cristallographie
M. DUBAR Claude	Sociologie démographique
M. DUBOIS Henri	Spectroscopie hertzienne
M. DUBOIS Jean Jacques	Géographie
M. DUBUS Jean Paul	Spectrométrie des solides
M. DUPONT Christophe	Vie de la firme
M. DUTHOIT Bruno	Génie civil
Mme DUVAL Anne	Algèbre
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FARVACQUE Jean Louis	Physique de l'état condensé et cristallographie
M. FAUQUEMBERGUE Renaud	Composants électroniques
M. FELIX Yves	Mathématiques
M. FERRIERE Jacky	Tectonique - Géodynamique
M. FISCHER Jean Claude	Chimie organique, minérale et analytique
M. FONTAINE Hubert	Dynamique des cristaux
M. FORSE Michel	Sociologie
M. GADREY Jean	Sciences économiques
M. GAMBLIN André	Géographie urbaine, industrielle et démographie
M. GOBLOT Rémi	Algèbre
M. GOURIEROUX Christian	Probabilités et statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie organique
M. GUELTON Michel	Chimie physique
M. GUICHAOUA André	Sociologie
M. HAIMAN Georges	Modélisation,calcul scientifique, statistiques
M. HOUDART René	Physique atomique
M. HUEBSCHMANN Johannes	Mathématiques
M. HUTTNER Marc	Algèbre
M. ISAERT Noël	Physique de l'état condensé et cristallographie
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et statistiques
M. JEAN Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KOENIG Gérard	Sciences de gestion
M. KOSTRUBIEC Benjamin	Géographie
M. KREMBEL Jean	Biochimie
Mme KRIFA Hadjila	Sciences Economiques
M. LANGEVIN Michel	Algèbre
M. LASSALLE Bernard	Embryologie et biologie de la différenciation
M. LE MEHAUTE Alain	Modélisation,calcul scientifique,statistiques
M. LEBFEVRE Yannic	Physique atomique,moléculaire et du rayonnement
M. LECLERCQ Lucien	Chimie physique
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Marc	Composants électroniques et optiques
M. LEFEVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
M. LEGRAND Michel	Astronomie - Météorologie
M. LEGRAND Pierre	Chimie
Mme LEGRAND Solange	Algèbre
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne

M. TURREL Georges	Spectrochimie infrarouge et raman
M. VANDIJK Hendrik	
Mme VAN ISEGHEM Jeanine	Modélisation,calcul scientifique,statistiques
M. VANDORPE Bernard	Chimie minérale
M. VASSEUR Christian	Automatique
M. VASSEUR Jacques	Biologie
Mme VIANO Marie Claude	
M. WACRENIER Jean Marie	Electronique
M. WARTEL Michel	Chimie inorganique
M. WATERLOT Michel	géologie générale
M. WEICHERT Dieter	Génie mécanique
M. WERNER Georges	Informatique théorique
M. WIGNACOURT Jean Pierre	
M. WOZNIAK Michel	Spectrochimie
Mme ZINN JUSTIN Nicole	Algèbre

Remerciements

Vincent Cordonnier m'a donné le goût de l'architecture et des voyages. Il m'a aussi proposé mon premier sujet de recherche, et je suis très heureux qu'il préside ce jury.

Michel Mériaux a dirigé mes travaux. Il m'a toujours encouragé et écouté, et a su assurer à l'ensemble de l'équipe un excellent environnement de travail.

Je remercie Michel Lucas et Claude Puech de m'avoir fait l'honneur d'être rapporteurs de cette thèse.

Il me semble que la recherche universitaire en architecture ne peut ignorer le monde de l'industrie. J'apprécie que Serge Couvet s'intéresse à nos travaux et y apporte son regard critique.

Je remercie également Arie Kaufman et Michel Weinfeld pour avoir accepté de juger ce travail.

Je travaille en étroite collaboration avec Christophe Chaillou depuis quatre ans. Ce travail est en partie le fruit de nos nombreuses discussions, et me semble bien refléter notre complémentarité.

Les bureaux 320 (malgré son brouillard permanent) et 322 abritent une faune éclectique et accueillante. Je tiens à remercier tout particulièrement Samuel Degrande pour sa compétence, sa gentillesse et sa disponibilité. Toute l'équipe lui doit beaucoup.

Le LIFL est bien plus qu'un simple lieu de travail ; merci à toutes les personnes qui, de jour comme de nuit et même le dimanche, y entretiennent une ambiance chaleureuse et décontractée.

Merci enfin à Henri Glanc pour avoir assuré la reproduction de cette thèse avec son efficacité et sa célérité habituelles.

Last, but not least, merci à mes parents qui m'ont toujours donné un environnement propice à la poursuite de mes études.

Sommaire

Introduction 1

Chapitre 1 Notions de base 3

1.1 Méthodes de génération d'images de synthèse 3
1.1.1 Le lancer de rayons 3
1.1.2 Les techniques de radiosité 3
1.1.3 Le rendu volumique 4
1.1.4 Le rendu géométrique 4
1.2 Technologie des systèmes graphiques 4
1.2.1 Les écrans 4
1.2.2 La mémoire de trame 5
1.3 Le pipeline de rendu classique 6
1.3.1 La phase de préparation 6
1.3.2 La conversion des objets en pixels 8
1.3.3 L'élimination des parties cachées 9
1.4 Vers plus de réalisme 10
1.4.1 Méthode de Phong 10
1.4.2 Antialiassage 11
1.4.3 Placage de textures 11
1.4.4 Ombres portées 12
1.5 Les stations de travail actuelles 13
1.6 Au coeur du temps réel 14
1.6.1 Fréquence d'animation 14
1.6.2 Définition de la latence d'un système 15

Chapitre 2 Architectures massivement parallèles pour la synthèse d'images 17

2.1 Définitions des niveaux de parallélisme 17
2.1.1 Parallélisation des calculs géométriques 17
2.1.2 Parallélisation de la conversion 19
2.1.3 Conclusion 23
2.2 Parallélisme massif total 23
2.2.1 Approche pixel 23
2.2.2 Approche objet 28

2.2.3 Comparaison des approches pixel et objet 37
2.2.4 Conclusion 38
2.3 Parallèlisme massif partiel séquentiel 39
2.3.1 Introduction 39
2.3.2 Approche pixel : SAGE 41
2.3.3 Approche objet : GSP-NVS 43
2.3.4 Comparaison des deux approches 46
2.3.5 Equivalent en monoprocesseur 46
2.4 Parallélisme image haut niveau 47
2.4.1 Rappel 47
2.4.2 Parallélisme image bas niveau : Pixel-Planes 5 48
2.5 Parallélisme objet haut niveau 52
2.5.1 Rappel 52
2.5.2 Parallélisme pixel bas niveau : PixelFlow 52
2.6 Conclusion 55

Chapitre 3 Unités de conversion massivement parallèles. 57

3.1 Présentation de l'étude 57
3.2 Caractéristiques des unités de conversion 58
3.2.1 Unité de conversion objet 58
3.2.2 Unité de conversion pixel 61
3.2.3 Unité de conversion séquentielle 62
3.3 Caractérisation de scènes 62
3.3.1 Définitions des paramètres caractéristiques 62
3.3.2 Modification des paramètres 63
3.4 Implémentation matérielle des unités de conversion 64
3.4.1 Choix de la méthode de génération des facettes 64
3.4.2 Unité de conversion objet 65
3.4.3 Unité de conversion pixel 65
3.5 Performances intrinsèques des unités de conversion 66
3.5.1 Unité de conversion objet 66
3.5.2 Unité de conversion pixel 69
3.5.3 Unité de conversion séquentielle 70
3.5.4 Conclusions 70
3.6 Accélération intrinsèque des unités de conversion 71
3.6.1 Unité de conversion objet 71
3.6.2 Unité de conversion pixel 73
3.7 Performances d'un système complet 74

3.7.1 Facteur de duplication 75
3.7.2 Performances globales du système 76
3.7.3 Fréquence maximale d'animation 78
3.8 Retour sur SAGE, GSP-NVS et Pixel-Planes 81
3.8.1 SAGE. 81
3.8.2 GSP-NVS. 84
3.8.3 Pixel-Planes 85
3.9 Conclusions. 85

Chapitre 4 Conversion de facettes 87

4.1 Définition d'une facette 87
4.2 Calcul incrémental d'expressions linéaires 87
4.2.1 Unité de conversion objet 87
4.2.2 Unité de conversion pixel 89
4.3 Processeur facette pour unité de conversion objet 90
4.3.1 Description générale du processeur 91
4.3.2 Les opérateurs de calcul. 92
4.3.3 Contrôle du processeur 102
4.3.4 Conclusion 106
4.4 Cellule de base d'un réseau multi-pipeline 107
4.4.1 Principe de base 107
4.4.2 Vers une réalisation VLSI. 108

Chapitre 5 Affichage de primitives de haut niveau. 109

5.1 Etat de l'art. 109
5.1.1 Pixel-Planes 4 109
5.1.2 Pixel-Planes 5 111
5.1.3 L'Apollo DN 10000. 113
5.1.4 La Ray-Casting Machine 114
5.1.5 Conclusion 115
5.2 Opérateurs incrémentaux pour le calcul de quadriques 115
5.2.1 Définitions mathématiques 115
5.2.2 Calcul d'une expression du second degré. 116
5.2.3 Calcul de la racine carrée 117
5.2.4 le Processeur Elémentaire Quadrique 119
5.3 Première génération de quadriques 120
5.3.1 Génération du contour 120
5.3.2 Calcul de la profondeur et de la normale. 120

5.3.3 Construction d'une quadrique 121
5.4 Deuxième génération de quadriques 122
5.5 Conclusion 124

Conclusion 127

Annexe A 129

Bibliographie 137

Introduction

Le domaine de la synthèse d'images par ordinateur est en constante évolution depuis quelques années, et peut être divisé en deux grandes catégories d'applications :

- *les images hyper-réalistes* popularisées de nos jours par le cinéma et la publicité. Ces images, d'un réalisme de plus en plus parfait, utilisent des méthodes très gourmandes en temps de calcul (radiosité, lancer de rayon). Actuellement, la recherche en ce domaine est orientée vers l'amélioration des algorithmes afin de réduire les temps de calcul tout en augmentant le réalisme des images (par exemple par une meilleure prise en compte des phénomènes lumineux).
- *les images interactives calculées en temps réel.* Les principales applications visées sont les simulateurs de conduite (type simulateurs de vol), la simulation scientifique (par exemple la simulation moléculaire) et les mondes virtuels. De nombreuses recherches ont été menées afin de concevoir des ordinateurs dédiés à ce type d'applications et augmenter ainsi le réalisme et la complexité des images générées en temps réel. Toutes ces machines utilisent comme primitives graphiques des facettes triangulaires, les plus puissantes annonçant des performances de l'ordre d'un million de facettes affichées par seconde.

Pour atteindre de telles performances, tous ces systèmes sont fortement parallélisés. Dans cette thèse, nous nous intéressons essentiellement aux systèmes utilisant un parallélisme massif (plusieurs centaines de processeurs utilisés en parallèle). Dans le domaine des ordinateurs de calcul, le parallélisme massif (popularisé par la CM-1 de Thinking Machine Corporation) est arrivé au stade commercial (CM-2, CM-5, Intel Paragon, DEC Maspar), permettant de depasser en puissance les supercalculateurs vectoriels (type CRAY) des années 80. Dans le domaine des ordinateurs graphiques, si des machines massivement parallèles existent, elles restent cependant à l'état de prototype dans les laboratoires de recherche.

En règle générale, une machine massivement parallèle utilise un très grand nombre de processeurs sur lesquels sont réparties les données à traiter. Dans le domaine de la synthèse d'images, ces données sont de deux types : les objets de la scène à visualiser et les pixels de l'écran. Il est alors naturel d'envisager deux types de parallélisme suivant l'association données/processeurs [MERI84] :

- si les processeurs traitent les objets de la scène, le parallélisme sera appelé parallélisme objet.
- si les processeurs traitent les pixels de l'image, le parallélisme sera appelé parallélisme pixel.

Le but de cette thèse est de comparer de manière formelle ces deux types de parallélisme, et de déterminer les puissances potentielles des machines graphiques basées sur ces concepts.

L'organisation de ce travail est la suivante :

- Le premier chapitre propose un exposé des notions de base de la synthèse d'images, tant d'un point de vue algorithmique que matériel. Il permet également de faire le point sur les performances des stations de travail graphiques commerciales actuelles.

- Le deuxième chapitre propose une description et une analyse de tous les systèmes graphiques massivement parallèles proposés ces dix dernières années. Ceux-ci sont classifiés suivant les différents types de parallélisme utilisés (objet ou pixel, haut niveau ou bas niveau). Par ailleurs, nous décrivons leurs organisations architecturales, leurs complexités matérielles et les performances atteintes (ou espérées si les systèmes n'ont pas été construits).
- Le troisième chapitre est consacré à une comparaison formelle des unités de conversion massivement parallèles objets et pixel (uniquement dans le cas de l'affichage de facettes). Les puissances intrinsèques développées par ces unités sont estimées et comparées. Par ailleurs, nous proposons une méthode générale pour analyser les systèmes graphiques utilisant de telles unités de conversion, en généralisant la notion de facteur de duplication.
- Le quatrième chapitre présente d'une part la réalisation VLSI d'un processeur de conversion de facettes triangulaires destiné à une machine objet, et d'autre part le principe général du processeur équivalent pour une machine pixel. Il permet ainsi d'évaluer la complexité de ces processeurs.
- Le dernier chapitre présente plusieurs solutions pour réaliser un processeur capable d'afficher directement des primitives quadriques (sans passer par l'étape de facettisation). Nous montrons en particulier l'intérêt que présente la parallélisme objet pour afficher des quadriques.

Chapitre 1
Notions de base

1.1 Méthodes de génération d'images de synthèse

Les méthodes de génération d'images de synthèse utilisées de nos jours peuvent être divisées en quatre catégories distinctes : le lancer de rayon, la radiosité, le rendu volumique et le rendu géomètrique. Nous présentons ici une brève description de leurs principes de base. Dans la suite de cette thèse, nous nous consacrerons uniquement à l'étude du rendu géomètrique qui est aujourd'hui la seule méthode permettant d'atteindre le temps réel.

1.1.1 Le lancer de rayons

La technique du lancer de rayons [WHIT80] est basée sur une simulation des rayons lumineux parcourant la scène à afficher. Elle calcule, en chaque pixel de l'écran, la lumière qui lui parvient en traitant les rayons lumineux dans le sens inverse des rayons réels. Les différentes étapes de la génération d'une image sont les suivantes :

A partir de l'oeil de l'observateur est lancé un rayon (appelé rayon primaire) vers chaque pixel de l'écran. On détermine ensuite le premier objet rencontré par chacun des rayons, ce qui permet de déterminer l'objet visible en chaque pixel.

Des nouveaux rayons sont lancés à partir de chaque point d'intersection ainsi calculé :

- un rayon (appelé rayon d'ombre) vers chaque source lumineuse permettant de déterminer si l'objet est éclairé ou non (création des ombres portées).
- un rayon (appelé rayon réfléchi) dans la direction symétrique de celle du rayon primaire par rapport à la normale à l'objet. Ce rayon permet de traiter les surfaces réfléchissantes.
- un rayon (appelé rayon réfracté ou transmis) dans le cas où l'objet est transparent.

Chaque rayon réfléchi ou transmis est ensuite traité comme un rayon primaire, et ainsi de suite. On obtient ainsi une structure arborescente de rayons dont il faut limiter la hauteur (on limite ainsi le nombre de réflexions multiples correctement traitées).

Cette technique produit des images reproduisant parfaitement les effets d'ombres portées, de reflets spéculaires, de transparences et de surfaces réfléchissantes. Malgré de nombreux travaux visant à optimiser le nombre de calculs effectués et à paralléliser cet algorithme, le lancer de rayon reste une méthode gourmande en temps de calculs (de plusieurs secondes à plusieurs dizaines de minutes suivant la complexité de la scène et le nombre de sources lumineuses).

1.1.2 Les techniques de radiosité

La radiosité, initialement proposée par Goral et al. [GORA84], permet de calculer des images de synthèse en modélisant les interactions lumineuses entre les différents objets composant

la scène. Elle est basée sur la propriété physique de conservation de l'énergie dans un espace clos. Cette méthode, qui nécessite de décomposer la scène en petites facettes, calcule, pour chaque couple de facettes, la quantité d'énergie lumineuse diffuse échangée. Les volumes de calculs à effectuer sont donc considérables. La première méthode implémentant efficacement la technique de radiosité, appelée méthode de l'hémi-cube, a été proposée par Cohen et al. en 1985 [COHE85]. Depuis, de nombreux travaux ont été menés soit pour optimiser les temps de calculs imposés par la méthode, soit pour ajouter des éclairements spéculaires (généralement en combinant radiosité et lancer de rayon [SILL89]).

1.1.3 Le rendu volumique

Le rendu volumique (appelé également rendu voxel) permet d'afficher des scènes définies directement par un ensemble de points tridimensionnels (repérés par leur coordonnées (x, y, z). Les points ainsi définis sont appelés des voxels, équivalents en trois dimensions de la notion de pixels en deux dimensions. Historiquement, les recherches sur le rendu voxel ont débuté avec l'apparition d'appareil médicaux fournissant directement des informations 3D (scanners, RMN[1]). De nos jours, de nombreux travaux de recherche sont menés afin d'utiliser le rendu voxel en synthèse d'images. Un aperçu des principaux travaux de recherche effectués dans ce domaine est proposé dans [KAUF91].

1.1.4 Le rendu géométrique

Le rendu géométrique permet de visualiser des images de synthèse en effectuant directement la projection sur l'écran des objets de la scène. Toutes les machines graphiques interactives, de la simple carte pour PC aux plus puissantes stations de travail graphiques utilisent cette technique. Les architectures graphiques massivement parallèles que nous étudions dans cette thèse sont également basées sur cette méthode. La suite de ce chapitre est entièrement consacrée au rendu géomètrique.

1.2 Technologie des systèmes graphiques

1.2.1 Les écrans

Les écrans utilisés actuellement sur toutes les stations graphiques haut de gamme, ainsi que sur les simulateurs[2], sont des écrans TRC (Tubes à Rayons Cathodiques) utilisant un balayage de trame. A cet effet, un faisceau balaie l'ensemble de l'écran ligne par ligne, pixel par pixel.

La fréquence de rafraîchissement (exprimée en Hertz) de l'écran indique le nombre de balayage complets effectués par seconde. Plus cette fréquence est élevée, plus le confort visuel de l'écran est grand (absence de scintillement notamment).

Les écrans couramment utilisés sur les systèmes graphiques possèdent une définition de 1280×1024 pixels et une fréquence de rafraîchissement dépassant les 60 Hz.

- écran 1280×1024, 76Hz. Fréquence pixel 135 MHz
- écran 1152×900, 66 Hz. Fréquence pixel 93 MHz

1. Résonnance Magnétique Nucléaire.
2. Certains simulateurs de vol de nuit utilisent encore des écrans vectoriels car ce sont les seuls capables de reproduire fidèlement les lumières ponctuelles.

1.2.2 La mémoire de trame

En règle générale, un système graphique utilise une mémoire appelée mémoire de trame dans laquelle est mémorisée l'image à afficher. Dans les systèmes haut de gamme, chaque pixel est codé sur 24 bits (8 bits pour la composante Rouge, 8 pour la composante Vert et 8 pour la composante Bleu). Cette mémoire est utilisée d'une part par le (ou les) processeur graphique qui y mémorise l'image synthétisée, et d'autre part par un circuit spécialisé appelé contrôleur vidéo chargé de transférer les pixels de la mémoire vers l'écran (via des convertisseurs numériques/analogiques).

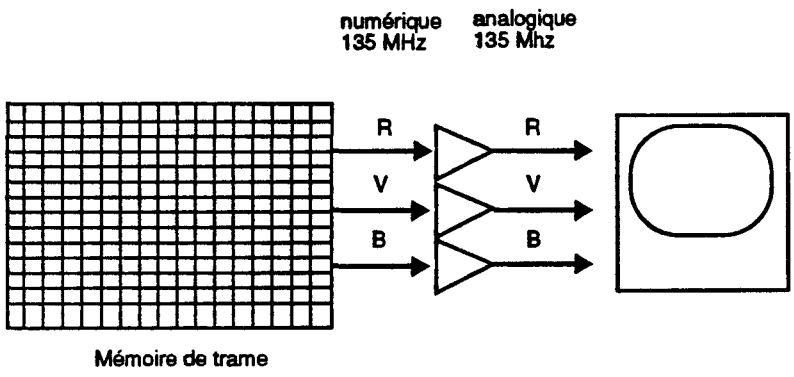


Figure 1.1 : mémoire de trame et écran

Tout système graphique performant repose donc sur des mémoires rapides. En effet, pour rafraîchir un écran 1280 x 1024, 76 Hz, vraies couleurs, le débit de sortie de la mémoire doit être de 405 Moctets par seconde.

1.2.2.1 Les DRAMs

Bien évidemment, aucune mémoire dynamique n'est capable de soutenir un tel rythme. La solution consiste alors à utiliser plusieurs bancs mémoires en parallèle. Les DRAMs actuelles stockent les mots sur 8 bits et possèdent un temps d'accès en lecture d'environ 70 ns (en mode page, c'est-à-dire en lecture sur une colonne donnée, ce qui est le cas d'un rafraîchissement écran). Une mémoire de trame utilisant ces composants doit donc utiliser dix boîtiers en parallèle pour chaque composante de couleur.

1.2.2.2 Les VRAMs

Pour mieux répondre aux besoins spécifiques de la visualisation graphique, des composants mémoires spécifiques, appelés VRAM (comme Vidéo RAM) ont été développés. Un composant VRAM est identique à un composant DRAM, mais possède en sortie un registre à décalage pouvant être utilisé indépendamment du port d'entrées/sorties classique.

Dans le cas d'une mémoire de trame, l'avantage offert par les VRAM est double :

- le temps d'accès du port série est inférieur au temps d'accès d'une DRAM (même en mode page), ce qui permet de réduire le nombre de boîtiers utilisés en parallèle. Par ailleurs, il n'est pas nécessaire de fournir l'adresse lorsqu'on utilise le registre série.

- L'utilisation d'un registre de sortie permet de disposer d'un double accès à la mémoire (le contrôleur utilise le registre tandis que le processeur graphique utilise le port standard).

1.3 Le pipeline de rendu classique

Dans les méthodes de rendu géométrique, les objets à représenter sont en général modélisés à l'aide d'un maillage de facettes triangulaires. La méthode de visualisation la plus courament utilisée (du moins sur les stations graphiques) est appelée "pipeline de rendu Gouraud" (Figure 1.2).

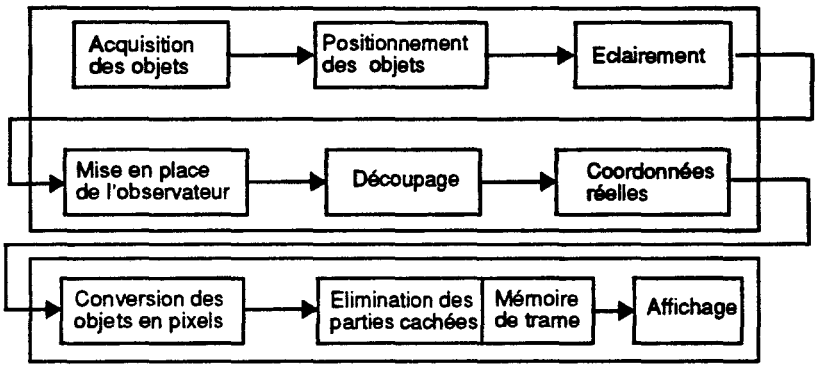


Figure 1.2 : le pipeline de rendu classique

Il est composé de trois étapes : la phase de préparation des objets, la conversion des objets en pixels et l'élimination des parties cachées.

1.3.1 La phase de préparation

1.3.1.1 Acquisition, positionnement

Les objets sont extraits de la base de données, et positionnés dans un repère absolu.

1.3.1.2 Eclairement

Pour générer une image de synthèse, il faut déterminer en chaque pixel de l'écran la couleur (RVB) à afficher. Pour cela, de nombreux modèles d'éclairement ont été définis (souvent empiriquement) pour modéliser le comportement d'un objet éclairé par une source lumineuse. Nous ne présentons ici que les deux modèles les plus souvent utilisés dans le pipeline de rendu classique : le modèle pour l'éclairement diffus, et le modèle pour l'éclairement spéculaire.

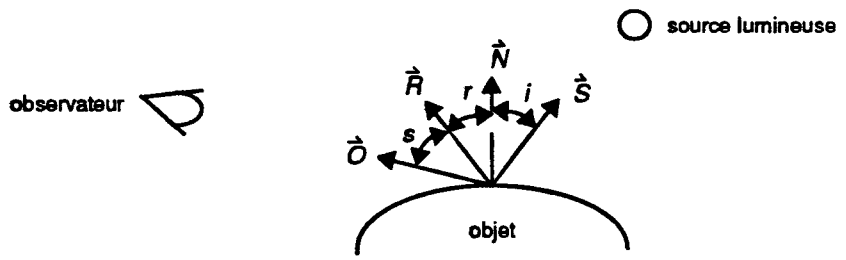


Figure 1.3 : vecteurs utilisés

Les vecteurs utilisés dans les modèles d'éclairement simples sont les suivants (voir Figure 1.3) :

- $\vec{S}$: vecteur source.
- $\vec{N}$: vecteur normal.
- $\vec{R}$: vecteur réfléchi.
- $\vec{O}$: vecteur observateur.

Les angles utilisés sont les suivants :

- i : angle d'incidence.
- r : angle de reflexion.
- s : angle spéculaire.

Eclairement diffus

L'éclairement diffus est calculé par la loi de Lambert

$$I_{diffus} = k_d \cdot \cos(i) \cdot I_S = k_d \cdot (\vec{N} \cdot \vec{S}) \cdot I_s, \text{ avec}$$

- k_d : coefficient diffus.
- I_S : intensité de la source lumineuse.

Eclairement spéculaire

L'éclairement spéculaire est calculé en général par la formule de Phong [PHON75] :

$$I_{speculaire} = k_s \cdot \cos^n s \cdot I_s = k_s \cdot (\vec{R} \cdot \vec{O})^n \cdot I_s, \text{ avec}$$

- k_s : coefficient spéculaire.
- n : exposant spéculaire (défini l'aspect métallique de la surface)

Dans le cas du pipeline de rendu classique, l'éclairement est calculé aux sommets des facettes.

1.3.1.3 positionnement de l'observateur

Les objets, jusque là définis dans le repère absolu, sont positionnés dans le repère de l'observateur (calcul matriciel 4×4).

1.3.1.4 Découpage, projection

Les facettes sont découpées par le polyèdre de vision afin d'éliminer les parties susceptibles de "déborder" de l'écran. Les facettes, définies par leurs sommets en coordonnées homogènes, sont enfin transformées en coordonnées réelles.

1.3.1.5 Coût de la phase de préparation

L'ensemble des calculs de la phase de préparation doivent être effectués en nombres flottants. Si l'on considère des facettes triangulaires indépendantes, chacune est définie par trois sommets. On trouve dans [AKEL89] une estimation de la puissance de calcul demandée par la phase de préparation pour des facettes à quatre sommets (éclairées par une source lumineuse située à l'infini). Transformer 100.000 facettes par seconde requiert une puissance de calcul d'environ 50 MFlops.

Le processeur flottant le plus utilisé aujourd'hui, le i860XP, délivre une puissance théorique de 80 MFlops. En pratique, il est difficile de dépasser les 50-60 MFlops (par programmation optimisée en assembleur). Si on désire réaliser une machine graphique affichant 1.000.000 de facettes par seconde, il faut donc utiliser une dizaine de processeurs. Pour atteindre les 10.000.000 de facettes par seconde, il faudrait en utiliser une centaine.

1.3.2 La conversion des objets en pixels

La première méthode utilisée pour représenter les objets a été la représentation "fil-de-fer"[1]. Dans cette méthode, seules les arêtes des polygones sont dessinées. L'étape suivante, appelé ombrage constant[2], a consisté à calculer un éclairement constant sur chacune des facettes composant un objet. Les objets apparaissaient donc en couleur, mais leur aspect facettisé demeurait visible.

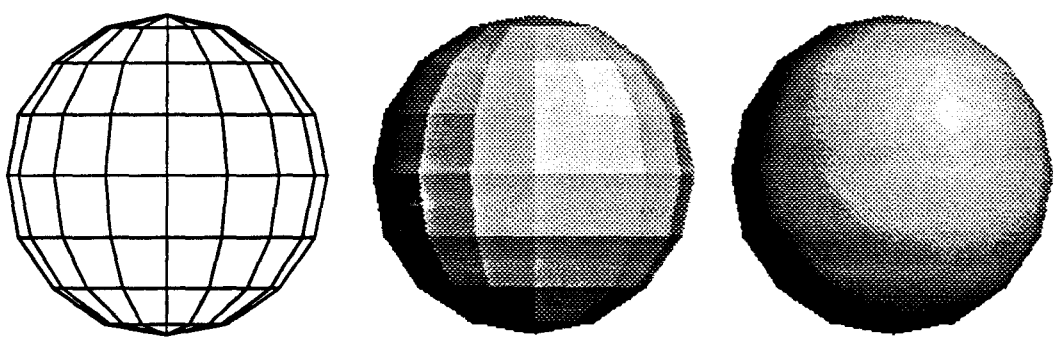
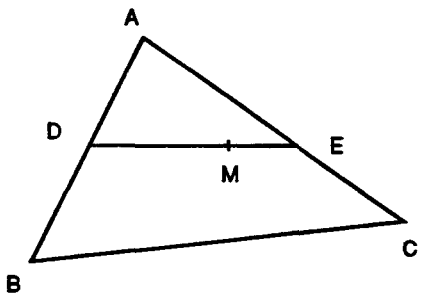


Figure 1.4 : les trois modèles de représentation (courtesy of S. R. Degrande, LIFL)

Afin de reproduire l'effet d'ombrage naturel des objets courbes, et donc de faire disparaître leur aspect facettisé, Gouraud [GOUR71] a proposé de calculer la couleur en chaque point d'une facette triangulaire par double interpolation des valeurs aux sommets. Par ailleurs, la

1. Wireframe.
2. Flat-shading.

normale au sommet d'une facette est calculée en faisant la moyenne des normales des faces adjacentes (ceci permet d'approcher la normale à la surface courbe).



$$ID = \frac{AD}{AB} \times IA + \frac{BD}{AB} \times IB$$

$$IE = \frac{AE}{AC} \times IA + \frac{CE}{AC} \times IC$$

$$IM = \frac{DM}{DE} \times ID + \frac{EM}{DE} \times IE$$

Figure 1.5 : double interpolation linéaire

Cette méthode permet un rendu correct de l'éclairement diffus. Cependant, si la couleur aux sommets de la facette est calculée par le modèle d'éclairement de Phong, des défauts visuels importants apparaissent lors de l'interpolation. Par ailleurs, ces défauts sont fortement accentués en cas d'animation (déplacements saccadés des reflets spéculaires).

1.3.3 L'élimination des parties cachées

Si l'élimination des parties cachées a longtemps été un sujet de recherche prolifique dans la communauté graphique, il semble aujourd'hui clos. La méthode la plus utilisée (pour ne pas dire la seule) est appelée méthode du Z-buffer. Celle-ci nécessite l'utilisation, en plus de la mémoire de trame, d'une mémoire de profondeur permettant de stocker en chaque pixel la profondeur de l'objet visible. Lorqu'un nouvel objet arrive en un pixel, sa profondeur est comparée avec celle du Z-buffer[1]. Si elle est inférieure, cela signifie que le nouvel objet est situé devant. Sa profondeur remplace alors l'ancienne valeur dans le Z-buffer, tandis que sa couleur RVB remplace l'ancienne valeur dans la mémoire de trame (voir Figure 1.6).

La simplicité de cet algorithme facilite grandement son implémentation matérielle. Le seul problème délicat est de disposer de mémoires suffisamment rapides pour garantir des performances élevées (le Z-buffer repose sur un cycle lire-comparer-écrire). Toute les stations graphiques actuelles sont basées sur cet algorithme, et résolvent le problème du débit de la mémoire en parallélisant les accès au Z-buffer.

1. On appelle Z-buffer à la fois la méthode et la mémoire de profondeur.

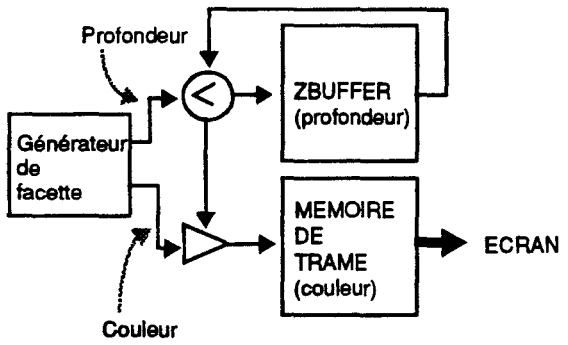


Figure 1.6 : schématisation d'un Z-buffer cablé

1.4 Vers plus de réalisme

Le pipeline de Gouraud est encore utilisé dans de nombreuses machines graphiques en raison de sa simplicité, facilitant ainsi une réalisation matérielle. Cependant, la qualité des images produites est alors moyenne. Nous présentons maintenant plusieurs techniques proposées depuis plusieurs années pour améliorer notablement la qualité des images (au prix bien sûr d'une augmentation de la complexité des calculs à effectuer). Certaines sont devenues indispensables, d'autres le seront probablement d'ici quelques années.

1.4.1 Méthode de Phong

Le principal défaut de la méthode de Gouraud est la médiocre prise en compte des reflets spéculaires lorsque ceux-ci sont pris en compte dans le modèle d'éclairement utilisé (défaut accentué en cas d'animation interactive). Afin de pallier ces défauts, Phong [PHON75] a proposé d'interpoler en chaque point d'une facette non plus la couleur (définie par les trois composantes R, V et B), mais le vecteur normal (défini par les trois composantes Nx, Ny et Nz). L'éclairement doit alors être effectué après la conversion des objets en pixels.

Cette méthode donne de bien meilleurs résultats que celle de Gouraud, mais au prix d'une augmentation importante du nombre de calculs. En effet, les calculs d'éclairement ne sont pas effectués aux sommets des facettes (trois calculs par facette triangulaire), mais en chaque pixel de l'image finale. A titre d'exemple, considérons une scène composée de 100.000 facettes triangulaires, éclairée par une seule source lumineuse et affichée sur un écran 1280×1024. La méthode de Gouraud nécessite 300.000 calculs d'éclairement (100.000 seulement dans le cas de facettes maillées car les sommets sont alors partagés), alors que celle de Phong en nécessite 1,3 millions.

Notons enfin qu'il ne faut pas confondre modèle d'éclairement de Phong et interpolation de Phong.

Toutes les stations graphiques commerciales actuelles utilisent l'interpolation de Gouraud et non celle de Phong[1], car celle-ci requiert un processeur spécialisé après l'étape de conversion (i.e. après les interpolations et l'élimination des parties cachées) pour effectuer tous les calculs d'éclairement. Il est cependant probable que, les technologies VLSI progressant à

1. par contre, elles utilisent souvent le modèle d'éclairement de Phong avec interpolation de Gouraud, produisant ainsi des résultats visuels médiocres.

grand pas, les futures stations utiliseront la méthode de Phong.

De nombreux projets de recherche ont tenté d'intégrer un processeur d'éclairement après l'étape de conversion des objets [DEER88][CLAU91][LEFE91]. A ce jour, seules les machines Pixel-Planes 5 [FUCH89][ELLS91] et PixelFlow [MOLN92] (en cours de construction) l'ont fait avec succès.

1.4.2 Antialiassage

L'aliassage est un phénomène résultant de la discrétisation d'un signal continu. En synthèse d'images, il se manifeste par des défauts dans les images produites dont les principaux sont :

- marches d'escalier[1] sur les bords des objets. Ce phénomène est d'autant plus visible (et donc d'autant plus génant) que la résolution de l'écran est faible.
- phénomènes de moiré dans le cas d'utilisation de textures.
- clignotement des petits objets dans le cas d'images animées.

Pour remédier à ces phénomènes, on distingue deux grandes familles de méthodes d'antialiassage (autres que l'augmentation de la résolution de l'écran).

Les algorithmes de pré-filtrage considèrent le pixel comme une entité surfacique, et calculent pour chaque objet le taux d'occupation du pixel. Cette information est utilisée pour calculer la couleur effective de chaque pixel (en fonction de tous les objets présents). Le célèbre algorithme du A-buffer [CARP84] appartient à cette catégorie. Une implémentation matérielle de cette méthode est proposée dans [SCHN88b] et [WEIN81]. [MOLN91] présente un excellent aperçu des problèmes liés à cette méthode.

Les algorithmes de suréchantillonage échantillonent l'image à une résolution supérieure à celle de l'écran, puis calculent l'image finale à l'aide d'un filtre passe-bas. Notons que suréchantilloner l'image revient également à l'échantillonner plusieurs fois à la résolution standard, en appliquant à chaque passe une légère modification du point de vue (inférieure au pixel). Cette méthode est donc coûteuse en temps, mais elle possède plusieurs avantages déterminants. Tout d'abord, elle est compatible avec la méthode du Zbuffer, et est donc facilement implémentable sur les machines utilisant cette technique d'élimination des parties cachées. Elle est également générale, car elle traite l'image après la conversion des objets en pixel. Elle peut donc être utilisée quel que soit le type de primitives graphiques utilisées (facettes ou autres). Enfin, elle traite naturellement les arêtes implicites produites par les intersections d'objets.

Pour toutes ces raisons, le suréchantillonage est la méthode utilisée actuellement dans les stations de travail graphiques haut de gamme [HAEB90][SILI92][EVA92] et les machines expérimentales [FUCH85][DEER88][FUCH89][MOLN92].

1.4.3 Placage de textures

Pour améliorer le réalisme des images produites sans augmenter le nombre de facettes à traiter, une solution aujourd'hui couramment utilisée est le placage de textures[2]. En général, les textures sont stockées dans une mémoire auxiliaire (appelée table de textures), l'adresse dans cette table étant générée par interpolation au moment de la conversion des facettes (précisons que pour obtenir une mise en perspective correcte des textures, il est nécessaire de travailler en coordonnées homogènes et donc d'interpoler les valeurs u, v et w. L'adresse

1. Jaggies en anglais
2. texture mapping

exacte dans la table de textures est alors donnée par u/w et v/w).

Jusqu'en 1990, les textures n'étaient exploitées en temps réel que sur les simulateurs de vol professionnels. Depuis l'apparition de la gamme Silicon Graphics VGX [SILI90], et plus récemment la gamme Reality Engine [SILI92], elles sont maintenant utilisables sur les stations de travail graphiques.

1.4.4 Ombres portées

L'ajout des ombres portées permet d'augmenter considérablement le réalisme des images générées par ordinateur. De plus, les ombres peuvent, dans certaines applications telles que les simulateurs de vol, apporter à l'utilisateur des informations précieuses quant à la position des objets les uns par rapport aux autres (par exemple l'altitude de son avion en fonction de son ombre sur le sol). Le coût de génération des ombres est malheureusement si élevé que peu de systèmes les proposent en temps réel.

Il existe trois grandes familles d'algorithmes de génération des ombres portées :

1.4.4.1 Le lancer de rayon et la radiosité

Le lancer de rayon et la radiosité sont des méthodes générales qui prennent en compte naturellement les ombres portées. Dans le cas du lancer de rayon, les ombres sont générées en lançant, lors de l'intersection d'un rayon primaire avec un objet, un rayon vers chaque source lumineuse. Il est aujourd'hui impossible, même sur les plus puissants supercalculateurs, de calculer en temps réel des scènes en lancer de rayon.

Dans le cas de la radiosité, les ombres sont naturellement prises en compte lors du calcul de l'énergie échangée entre deux facettes. Notons toutefois que la précision de la facettisation détermine la précision des ombres générées (une facettisation grossière engendre des ombres crénelées). Pour remédier à ce problème, des méthodes de facettisation adaptative ont été proposées. Notons qu'une scène calculée par la méthode de radiosité peut être affichée en temps réel sur toute machine spécifiquement cablée pour le pipeline de Gouraud (par exemple les stations de travail graphiques). Les ombres portées sont alors également affichées en temps réel lors du déplacement de l'observateur dans la scène. Cependant, il n'est pas possible de modifier la position des objets, car cela nécessiterait un recalcul (éventuellement partiel) de l'image (impossible en temps réel).

1.4.4.2 La méthode de Williams

La méthode de Williams [WILL78] est une méthode en deux passes basée sur le Z-buffer. Elle nécessite cependant l'utilisation d'une mémoire de profondeur supplémentaire. La première passe effectue l'élimination des parties cachées en prenant la source lumineuse comme observateur (on détermine ainsi les objets visibles depuis la source, c'est à dite ceux directement éclairés). Les profondeurs ainsi calculées sont mémorisées dans une mémoire intermédiaire. La deuxième passe effectue un rendu classique de la scène (par Z-buffer) du point de vue de l'observateur. Les profondeurs sont cette fois mémorisées dans la mémoire principale (ainsi que les valeurs RVB). A chaque pixel de la mémoire de trame est ensuite associé (via un changement de repère) la profondeur correspondante dans la mémoire auxiliaire, permettant ainsi de déterminer si le pixel est à l'ombre de la source. Si cela est le cas, sa couleur est atténuée en conséquence.

Le principal problème de cette méthode est lié aux importants phénomènes d'aliassage qu'elle engendre, principalement dus au changement de repère de la mémoire de trame vers la mémoire auxiliaire.

Ce problème mis à part, cette méthode est avantageuse car elle s'appuie uniquement sur la technique du Z-buffer, et peut donc être facilement implémentée sur les stations de travail actuelles (si elles disposent de suffisamment de mémoire). Par ailleurs, le changement de repère peut être assimilé à celui effectué lors du plaquage de textures. Les stations actuelles bénéficiant de textures cablées peuvent donc utiliser la méthode de Williams sans dégrader exagèrement leurs performances [SEGA92].

1.4.4.3 La méthode de Crow

La méthode de Crow [CROW77] est basée sur la technique des volumes d'ombres. Développée initialement pour des facettes convexes, elle est suffisament générale pour pouvoir être adaptée à d'autres primitives plus complexes telles que les quadriques [NYIR92b].

A chaque objet est associé un volume d'ombre délimité par la source lumineuse et l'objet lui-même. Pour une facette triangulaire, le volume d'ombre ainsi généré est un polyèdre (infini dans un sens) à quatre cotés. Bien évidemment, chaque objet peut ainsi générer plusieurs volumes d'ombres (autant que de sources lumineuses).

Le calcul des ombres est ensuite effectué après l'élimination des parties cachées. En chaque pixel, l'algorithme détermine les volumes d'ombres qui contiennent l'objet visible. Si celui-ci est à l'intérieur d'un volume d'ombre, cela signifie qu'il n'est pas éclairé par la source considérée.

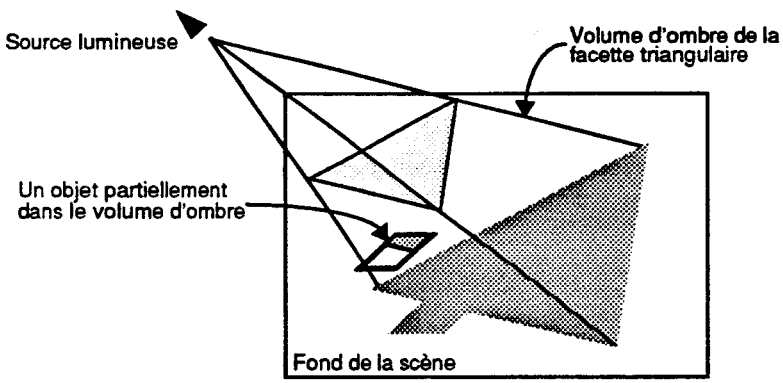


Figure 1.7 : méthode de Crow

Cette méthode a été utilisée sur la machine Pixel-Planes 4 [FUCH85] pour afficher interactivement des scènes avec ombres portées.

1.5 Les stations de travail actuelles

Depuis le début des années 80, de très nombreux constructeurs ont développé leurs propres machines graphiques. Parmi ceux-ci, on peut citer les sociétés proposant des stations de travail graphiques telles que AT&T [POTM89], Apollo [KIRK90], HP [SWAN86][McLE88], Silicon Graphics [AKEL88][AKEL89][HAEB90][SILI92], ainsi que celles proposant des solutions de visualisation pour supercalculateurs [APGA88][DENA88][DIED88][TORB87].

Dans le domaine des stations de travail graphiques, la société Silicon Graphics (SGI) est aujourd'hui le leader incontesté du marché. Créée par J. Clark au début des années 80, elle a bâti son hégémonie sur une gamme de machines graphiques puissantes et en constante évolution. La liste suivante présente un aperçu de l'évolution de la puissance de ces machines

et de la qualité des images produites.

- 1988. Gamme IRIS 4D/GTX. 100.000 facettes (100 pixels) par seconde avec interpolation de Gouraud et Z-buffer.
- 1990. Gamme VGX et VGXT. 1,1 millions de triangles par seconde avec Z-buffer. 100.000 triangles par seconde avec Z-buffer et textures.
- 1992. Gamme Reality Engine. 1,1 millions de triangles par seconde avec Z-buffer. 600,000 triangles par seconde avec Z-buffer, textures et anti-aliassage. Résolution écran 1280 x 1024, 1600 x 1200 et HDTV.

Par ailleurs, SGI propose également le système SkyWriter pour les applications de type simulation de vol. Celui-ci est composé de deux systèmes VGX (maintenant Reality Engine) permettant de garantir des fréquences d'affichage élevées (60 Hz).

1.6 Au coeur du temps réel

Tout au long de cette thèse, nous étudierons différents systèmes graphiques destinés à l'animation temps réel. La notion de temps réel en synthèse d'images recouvre de nombreuses notions, souvent liées les unes aux autres. Nous présentons ici les différents paramètres permettant de juger de l'adéquation d'un système graphique aux applications temps réel.

1.6.1 Fréquence d'animation

L'élément déterminant est le nombre d'images affichées par seconde, appelé aussi la fréquence d'animation du système.

- une application interactive (visualisation CAO, visualisation moléculaire interactive) nécessite une fréquence d'animation supérieure ou égale à 10 images par seconde (10 Hz).
- une application temps réel classique (simulateur bas de gamme) nécessite une fréquence d'animation de 30 images par seconde.
- une application temps réel haut de gamme (simulateur haut de gamme, mondes virtuels) nécessite une fréquence d'animation de 60 images par seconde.
- certaines applications peuvent nécessiter des fréquences d'animation encore plus élevées. Plus la fréquence d'affichage est élevée, plus le confort visuel est grand.

Il est intéressant de comparer ces chiffres avec ceux utilisés dans le cinéma. Le cinéma classique utilise une fréquence d'animation de 24 images par seconde. Historiquement, cette fréquence s'est trouvée être celle présentant le meilleur rapport confort visuel/coût. Certains procédés utilisés dans des films de démonstration utilisent une fréquence d'animation beaucoup plus élevée. Ainsi, le système Showscan, développé en 1986 par D. Trumbull (qui fut notamment responsable des effets spéciaux du film *2001 Odysée de l'espace*), utilise une fréquence d'animation de 60 images par seconde, augmentant ainsi considérablement la qualité des images et la reproduction des mouvements (en particulier les mouvements très rapides). Malheureusement, une telle qualité se paie : à format égal, un film Showscan utilise deux fois et demi plus de pellicule qu'un film classique. Nous verrons dans cette thèse que ce qui est vrai pour le cinéma l'est aussi pour les machines graphiques (mis à part le fait que le surcôut se chiffre non plus en surface de pellicule, mais en surface de silicium !). A titre d'exemple, les seuls systèmes actuellement capables de générer des images à 60 Hz sont les simulateurs de vol professionnels.

1.6.2 Définition de la latence d'un système

Lorsqu'un système graphique est utilisé dans des applications interactives, les images qu'il produit sont le fruit de la volonté de l'utilisateur. En effet, toute application interactive utilise la boucle d'interaction suivante :

(1) l'ordinateur présente une image à l'utilisateur.

(2) celui-ci agit sur les commandes (clavier, souris, joystick,...) en fonction de l'action qu'il désire effectuer.

(3) l'ordinateur recalcule la nouvelle image.

(4) l'ordinateur affiche la nouvelle image.

(5) retour en (1).

Le temps passé entre la fin de l'étape (2) et la fin de l'étape (4) est appelé la latence du système. C'est la somme de la latence des périphériques d'entrée (temps passé entre le début de l'étape (2) et le début de l'étape (3)) et de celle de la machine graphique (temps passé entre le début de l'étape (3) et la fin de l'étape (4)). La latence du système graphique s'exprime souvent en nombre d'images.

Chapitre 2

Architectures massivement parallèles pour la synthèse d'images

Nous allons dans ce chapitre proposer une classification des architectures graphiques massivement parallèles proposées ces dernières années. Le processus de création d'une image de synthèse peut clairement se décomposer en deux grandes partie : les calculs géométriques et la conversion des objets en pixels. Pour obtenir des performances élevées, chacune de ces étapes doit être parallélisée. Nous commençons ce chapitre en présentant rapidement (car ce n'est pas le sujet de cette thèse) les solutions envisageables pour paralléliser les calculs géométriques. Nous proposons ensuite une classification des architectures graphiques en fonction du type de parallélisme utilisé sans l'étape de conversion. Cette classification nous permet ensuite d'analyser les différents systèmes graphiques massivement parallèles proposés ces dernières années. Pour chaque machine, nous proposons une évaluation de la complexité matérielle, des performances, ainsi que des possibilités d'évolution.

2.1 Définitions des niveaux de parallélisme

2.1.1 Parallélisation des calculs géométriques.

Le terme "calculs géométriques" recouvre tous les calculs effectués sur les objets de la base de données, et précédant la phase de conversion de ces objets en pixels (cf. le chapitre 1 pour une classification détaillée).

Cette étape requiert uniquement une énorme puissance de calcul flottant dépendant principalement du nombre d'objets à afficher par image. Une estimation effectuée dans [AKEL89] montre qu'une unité graphique affichant 100.000 facettes par seconde (méthode de Gouraud) doit être alimentée par un hôte délivrant une puissance d'environ 50 MFlops (500 MFlops pour 1.000.000 de facettes par seconde).

Ces chiffres montrent que pour atteindre des performances graphiques satisfaisantes, l'unité de transformation géométrique doit être parallélisée. On distingue alors trois principales méthodes : pipeliner les opérations, utiliser plusiers ALU fonctionnant en mode SIMD, ou distribuer la base de données sur plusieurs unités de transformation fonctionnant en mode MIMD. Cette progression (pipeline puis SIMD puis MIMD) est parfaitement illustrée par l'évolution des unités géométriques des stations Silicon Graphics.

2.1.1.1 Pipeline

Cette solution est historiquement la première proposée pour paralléliser le processus de transformation géométrique. Les quatre composantes (x, y, z, w) sont traitées en parallèle, tandis que les différentes étapes (changement de repère, clipping, projection) sont effectuées en pipeline (Figure 2.1).

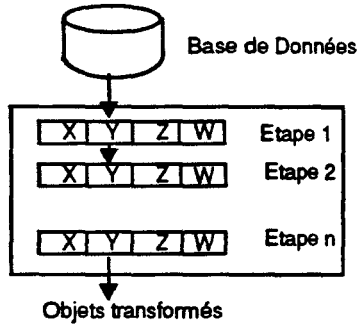


Figure 2.1 : pipeline sur les opérations et parallélisme sur les composantes

Le premier système VLSI utilisant ce type de parallélisme est le célèbre Geometry Engine [CLAR82] utilisé sur les premières stations graphiques de la firme Silicon Graphics. Il est important de noter qu'à l'époque (début des années 80), les processeurs généraux (ou les processeurs de signaux) n'offraient pas une puissance suffisante (en fait très peu de processeurs permettaient d'effectuer des calculs flottants), ce qui a conduit au développement de circuits spécifiques. Par la suite, le circuit Geometry Engine a été remplacé par des processeurs flottants Weitek sur la 4D/GTX [AKEL88].

Notons qu'un des principaux avantages du pipeline réside dans l'uniformité des chemins de données utilisés. En effet, le pipeline possédant une entrée (un objet) et une sortie, la liaison entre le module de transformation et le module de conversion en est grandement simplifiée. Malheureusement, cette approche ne peut pas être "indéfiniment " extensible, et montre ses limites quand le nombre d'objets à traiter augmente.

2.1.1.2 SIMD

Afin de pallier les défauts du pipeline, les concepteurs se sont orientés vers des unités géométriques composées de plusieurs unités de calcul fonctionnant en mode SIMD. Ainsi, l'étage de transformation géométrique de la Silicon Graphics VGX [SILI90] est composé de quatre processeurs flottants (des Texas Instruments 74ACT8867) permettant d'effectuer en parallèle des calculs sur les quatre sommets d'une facette (et non plus sur les quatre composantes d'un même sommet).

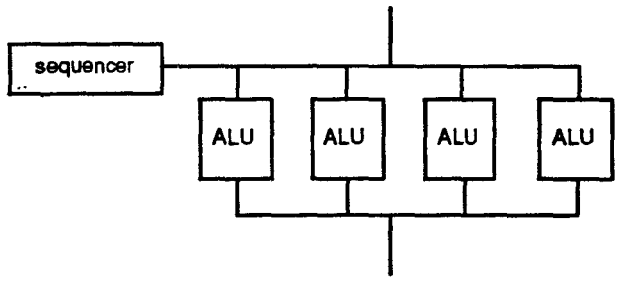


Figure 2.2 : organisation SIMD

2.1.1.3 MIMD

La dernière solution consiste à utiliser un parallélisme objet pour réaliser les calculs géométriques : la base de données est répartie sur plusieurs processeurs (généralement des processeurs généraux tels que le i860) effectuant chacun l'ensemble des transformations géométriques sur les objets qui lui sont attribués (Figure 2.3).

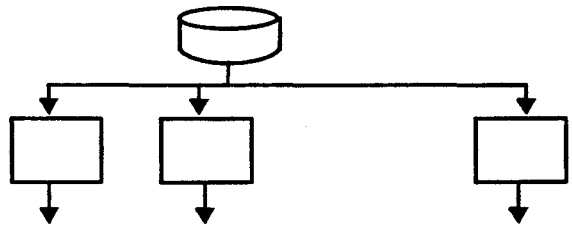


Figure 2.3 : distribution de la base de données

Cette solution, implémentée par exemple sur la machine Pixel-Planes 5 [FUCH89] et la SGI Reality Engine (huit i860XP), est très séduisante car elle permet un gain de performance linéaire en fonction du nombre de processeurs utilisés, et permet a priori d'atteindre des puissances (en MFLOPs) considérables. Cependant se posent alors des problèmes tels que la répartition des objets entre les différents processeurs [ELLS90] et le transfert des objets vers le(s) module(s) de conversion.

2.1.2 Parallélisation de la conversion

Dans l'ensemble du processus de création d'une image, l'étape suivant la phase de transformation géométrique est appelée conversion des objets en pixels[1].

Nous avons présenté dans le chapitre 1 la gamme de machines Silicon Graphics, qui propose aujourd'hui les machines graphiques commerciales les plus performantes, atteignant des puissances de l'ordre de 1 ou 2 millions de facettes par seconde. Ces machines sont bien évidemment fortement parallèles, le degré de parallélisme augmentant généralement d'année en année (du moins sur les machines haut de gamme). Malgré cela, le degré de parallélisme utilisé peut toujours être qualifié de moyen (quelques dizaines à quelques centaines d'unités).

Parallèlement aux systèmes commerciaux, de nombreux projets visant à définir des machines graphiques hautes performances à degré de parallélisme plus élevé ont été menés dans les laboratoires de recherche des universités ou des grands constructeurs. Nous ne nous intéresserons dans cette étude qu'aux systèmes utilisant un parallélisme massif. Ainsi, si la classification que nous proposons s'applique bien à ce type de machines, certains systèmes "classiques" (à parallélisme moyen ou faible) peuvent s'avérer difficiles à classifier suivant nos critères. Le lecteur intéressé pourra trouver une description de ces machines dans [FOLE90][LUCA91][CHAI92b].

Pour chacune des architectures, nous évoquons brièvement les principaux avantages et inconvénients. Une étude plus détaillée sera fournie ultérieurement lors de la description précise des machines.

---

1. scan-conversion en anglais. Dans la suite, nous utiliserons simplement le terme conversion

2.1.2.1 Les différents niveaux de parallélisme

Dans la suite, nous distinguerons deux niveaux de parallélisme :

- un parallélisme de haut niveau associant plusieurs unités de conversion travaillant en parallélisme MIMD. Ce parallélisme est toujours faible ou moyen;
- un parallélisme de bas niveau correspondant au degré de parallélisme interne d'une unité de conversion. Nous ne nous intéresserons dans notre étude qu'aux unités de conversion massivement parallèles (en général à contrôle SIMD).

D'une façon générale, le parallélisme utilisé dans l'étape de conversion peut être qualifié de parallélisme objet ou image. Nous définissons ainsi quatre types de parallélismes permettant de classifier toute machine graphique massivement parallèle :

- parallélisme objet haut niveau (MIMD)
- parallélisme image haut niveau (MIMD)
- parallélisme objet bas niveau (massivement parallèle SIMD)
- parallélisme image bas niveau (massivement parallèle SIMD)

2.1.2.2 Parallélisme objet de haut niveau

Le parallélisme objet consiste à générer en parallèle les différents objets composant la scène à afficher. Un système objet est composé de plusieurs unités de conversion, chaque unité effectuant sur tout l'écran la conversion et le traitement inter-objets d'une partie de la base de donnée. On peut ainsi considérer qu'une unité calcule une image complète (en taille) composée uniquement d'une partie des objets. Les images de toutes les unités de conversion sont ensuite composées pour obtenir l'image finale[1].

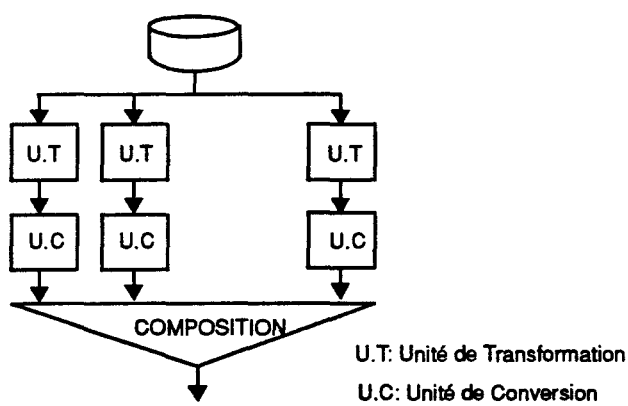


Figure 2.4 : parallélisme objet haut niveau

La Figure 2.4 présente les principaux avantages de ce type d'architecture :

- un parallélisme objet étant utilisé à la fois pour la transformation et la conversion, la liaison entre les modules de transformation et les modules de conversion ne pose aucun problème.

1. pour cette raison, les architectures à parallélisme objet sont également appelées architectures à composition [MOLN91]

- une architecture objet est facilement extensible, les performances croissant linéairement avec le nombre d'unités de conversion.

Les principaux problèmes posés par le parallélisme objet sont les suivants :

- la complexité du mécanisme de composition dépend du type d'opérations inter-objets que l'on souhaite implémenter (la plus simple étant l'élimination des parties cachées par l'algorithme du Zbuffer).
- le mécanisme de composition doit posséder un débit très élevé. A titre d'exemple, le réseau de composition de la machine PixelFlow est implémenté sur un fond de panier possèdant 256 lignes parallèles fonctionnant à 132 MHz, capable de transférer 30 Gbits/s !
- quelle stratégie choisir pour répartir les objets entre les différentes unités de conversion ?

2.1.2.3 Parallélisme image de haut niveau

Le parallélisme image consiste à diviser l'écran en zones, et à générer en parallèle les différentes zones. Chaque unité de conversion s'occupe uniquement des objets se situant dans la zone dont elle a la charge. Les images des différentes unités sont ensuite regroupées pour former l'image finale (Figure 2.5).

Les zones choisies sont généralement soit des lignes écran, soit des zones rectangulaires (la ligne écran pouvant être considérée comme une zone rectangulaire de hauteur 1 pixel), soit des zones rectangulaires entrelacées.

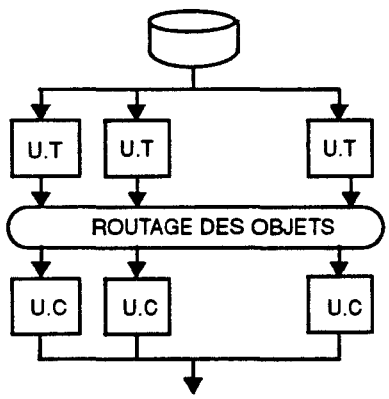


Figure 2.5 : parallélisme image haut niveau

Les principaux avantages du découpage image sont :

- efficacité des unités de conversion
- toutes les opérations inter-objets sont réalisées dans les unités de conversion.

Les principales difficultés rencontrées dans ce genre d'architecture sont :

- Les objets de la base de données doivent être triés en fonction du découpage de l'écran au début de chaque image (ceci augmente la complexité des algorithmes implémentés dans les unités de transformation).

- le mécanisme de routage est complexe (chaque unité de transformation doit pouvoir communiquer avec chaque unité de conversion) et doit être puissant. A titre d'exemple, le mécanisme de routage de la machine Pixel-Planes 5 est implémenté sous la forme d'un anneau possédant 8 voies indépendantes fonctionnant chacune à 80 Moctets/s, pour un débit total de 640 Moctets/s.
- des problèmes d'équilibrage de charge peuvent apparaître entre les différentes unités de conversion (en fonction du découpage choisi et de la scène à afficher).

2.1.2.4 La notion de réalisation partielle[1] en parallélisme image

Comme nous venons de le constater, le parallélisme image implique de diviser l'écran en plusieurs zones distinctes. L'allocation des unités de conversion aux zones écran peut être soit statique, soit dynamique.

Dans le cas d'une allocation statique, le système possède autant d'unités de conversion qu'il y a de zones, chaque unité s'occupant d'une zone et d'une seule. Nous dirons qu'un tel système utilise un parallélisme total (ou encore réalisation totale).

Dans le cas d'une allocation dynamique, chaque unité de conversion s'occupe de plusieurs zones. Il n'est donc pas nécessaire de disposer d'autant d'unités que de zones écran. Nous dirons donc qu'un tel système utilise un parallélisme partiel (ou encore réalisation partielle). L'allocation des unités de conversion se fait dynamiquement tout au long de la création de l'image (dès qu'une unité a fini de convertir une zone, une nouvelle zone lui est attribuée). Notons que cette solution impose de classer tous les objets de la base de données avant de commencer la conversion de la première zone, ce qui contribue à augmenter la latence du système.

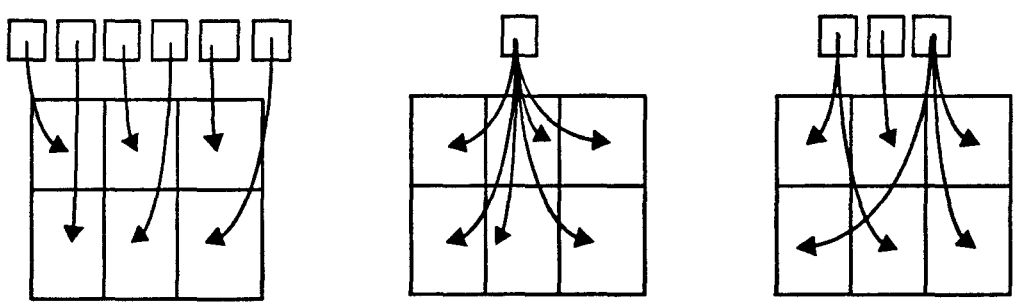


Figure 2.6 : réalisation totale, réalisation partielle séquentielle et parallèle

Dans le cas d'une réalisation partielle, nous distinguerons deux cas : soit le système utilise une seule unité de conversion[2] qui traite successivement toutes les zones écran (nous l'appellerons réalisation partielle séquentielle), soit il utilise plusieurs unités travaillant en parallélisme MIMD[3] (nous l'appellerons réalisation partielle parallèle).

1. Virtual en anglais
2. Virtual Buffer
3. Parallel Virtual Buffers

2.1.3 Conclusion

La classification que nous venons de présenter s'appuie sur la notion d'unité de conversion massivement parallèle objet ou pixel. Nous avons ainsi défini trois grandes familles de systèmes : les systèmes utilisant une seule unité de conversion associée à l'écran, les systèmes utilisant une seule unité de conversion associée à une zone de l'écran (réalisation partielle), et enfin les systèmes utilisant plusieurs unités de conversion en parallèle.

Dans la suite de ce chapitre, nous classons l'ensemble des machines massivement parallèles proposées depuis 1980 en fonction de la catégorie à laquelle elles appartiennent. Les machines de la première catégorie sont appelées systèmes à parallélisme massif total (SIMD). Celles de la deuxième catégorie sont appelées systèmes massivement parallèles à réalisation partielle séquentielle (SIMD). Enfin, la dernière catégorie regroupe les systèmes utilisant un double niveau de parallélisme, c'est-à-dire l'utilisation en parallélisme MIMD d'unités de conversion à parallélisme massif SIMD.

2.2 Parallélisme massif total

Les machines de cette catégorie utilisent une seule unité de conversion permettant de convertir l'ensemble des objets de la base de données vers l'ensemble des pixels de l'écran. Cette unité de conversion utilise un parallélisme massif SIMD. Dans le cas d'un parallélisme pixel, l'unité dispose d'un processeur par pixel de l'écran. Dans le cas d'un parallélisme objet, l'unité dispose d'un processeur par objet de la scène.

2.2.1 Approche pixel

Le parallélisme total pixel consiste à associer un processeur (appelé processeur pixel) par pixel de l'écran. Le nombre de processeurs utilisés dépend donc uniquement de la résolution choisie. Les architectures pixel se différencient principalement par le type de processeur pixel utilisé et le mécanisme de distribution des objets vers les processeurs pixel.

On distingue principalement deux méthodes de distribution des objets [LEPR89] :

- la diffusion : un objet est envoyé simultanément à tous les processeurs pixel qui fonctionnent en mode SIMD. Aucune communication n'est nécessaire entre les processeurs pixels.

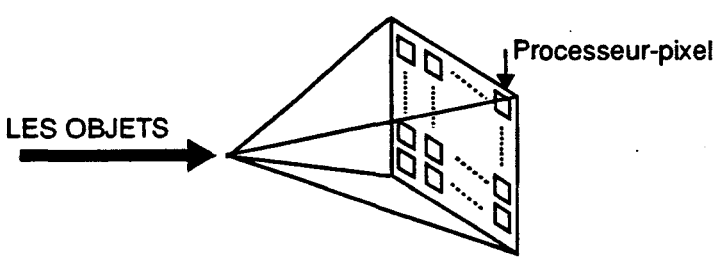


Figure 2.7 : diffusion

- le multi-pipeline : un objet est envoyé à tous les processeurs du bord gauche (simultanément ou en pipeline), puis chaque ligne est traitée en pipeline. Chaque pipeline fonctionne en mode SIMD, mais l'ensemble des pipelines peut fonctionner soit en mode SIMD (dans ce cas un objet est traité même sur les lignes où il n'apparait pas), soit en mode MIMD (ce qui permet à plusieurs objets disjoints en *y* d'être traités simultanément).

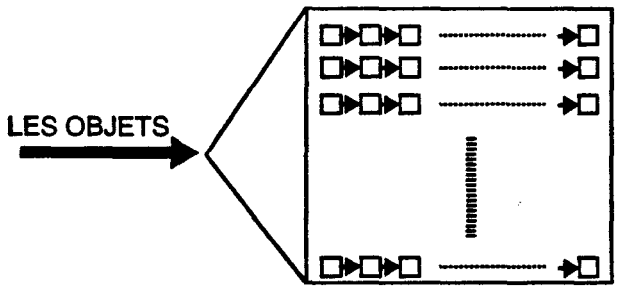


Figure 2.8 : multi-pipeline

Notons que les deux solutions présentées ne possèdent qu'un unique point d'entrée des objets. Les performances de ces systèmes peuvent donc être limitées par le débit de la liaison alimentant le module graphique, quelles que soient les performances de celui-ci.

Nous présentons maintenant deux exemples de systèmes représentatifs des deux approches que nous venons de décrire. Le premier est Pixel-Planes 4, développé dans les années 80 à l'Université de Caroline du Nord. Un prototype ayant effectivement été développé, nous pourrons en tirer des conclusions intéressantes sur les performances et la complexité réelles d'un tel système. Le second système que nous présentons est le projet RC, étudié à l'Université de Lille. Ce projet n'a pas abouti à un prototype opérationnel, mais l'étude a été menée suffisament loin pour pouvoir effectuer des comparaisons et tirer des conclusions.

2.2.1.1 Pixel-Planes 4

Pixel-Planes 4 [EYLE88][FUCH85] est à ce jour la seule machine massivement parallèle (à parallélisme total) orientée pixel effectivement construite. Les objets sont transmis aux processeurs pixels par une méthode à diffusion. En fait, le mécanisme de diffusion est un arbre d'additionneurs permettant à tous les processeurs pixel d'évaluer simultanément une expression linéaire de la forme $ax+by+c$, (x, y) étant les coordonnées du pixel.

Pixel-Planes 4 (Figure 2.9) est composé des élements suivants :

- un ordinateur hôte.
- un processeur graphique effectuant toutes les transformations géométriques et générant pour chaque objet la liste des coefficients *a*, *b* et *c* de toutes les expressions linéaires nécessaires à son affichage.
- un réseau 2D de 512*512 processeurs pixel possédant chacun une ALU 1 bit et 72 bits de mémoire. Un arbre d'additionneurs effectue l'évaluation des expressions linéaires.

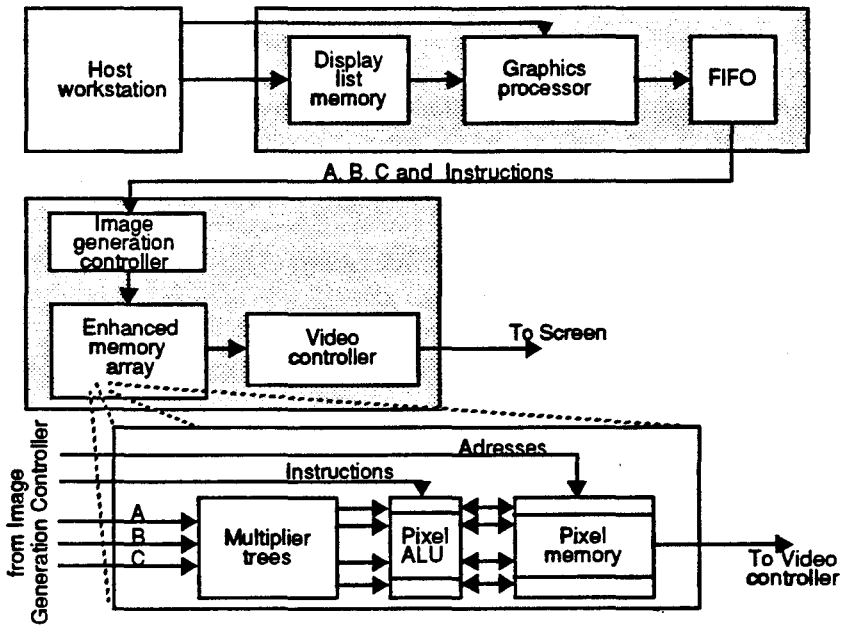


Figure 2.9 : schéma général de Pixel-Planes 4

Le circuit intégré de base développé pour Pixel-Planes 4 contient 128 processeurs pixel. Les différentes unités le composant sont les suivantes :

- la mémoire d'image pour les 128 pixels. 72 bits de mémoire sont disponibles en chaque pixel. Cette mémoire est utilisée pour stocker les valeurs RVB ainsi que la profondeur de l'objet visible en chaque pixel. La mémoire restante peut être utilisée pour effectuer des opérations complexes au niveau pixel (par exemple des opérations CSG).
- les 128 processeurs pixels. Chaque processeur est composé d'une ALU 1 bit effectuant au niveau pixel toutes les opérations arithmétiques et logiques.
- l'arbre d'additionneurs permettant de fournir aux 128 processeurs pixel la valeur d'une expression linéaire. Cette arbre est composé d'additionneurs 1 bit.

Le circuit intégré est réalisé en technologie nMOS 3 microns, et contient environ 63000 transistors (70 % pour la mémoire, 20 % pour l'arbre et 10 % pour les ALU). Un processeur pixel nécessite donc environ 500 transistors. Si l'on veut pouvoir comparer Pixel-Planes 4 avec d'autres systèmes massivement parallèle utilisant une mémoire de trame et un Z buffer externe (ou n'utilisant pas de mémoire de trame), il est nécessaire d'évaluer la complexité d'un processeur pixel privé de sa mémoire locale (72 bits). Si l'on considère que 70% du circuit intégré sont consacrés à la mémoire, on obtient une complexité d'environ 150 transistors par processeur pixel.

On retiendra donc les chiffres suivants :

- complexité d'un processeur pixel : 500 transistors.
- complexité d'un processeur pixel sans mémoire : 150 transistors.
- complexité totale des 512*512 processeurs : 130 Mtransistors.
- complexité totale de la partie active : 40 Mtransistors.

- nombre de circuits intégrés utilisés : 2048.

Pixel-Planes 4 est une machine programmable sur laquelle un grand nombre d'algorithmes de synthèse d'images peuvent être inplémentés. Nous nous intéresserons ici uniquement au rendu de facettes triangulaires par la méthode de Gouraud.

L'affichage d'une facette triangulaire nécessite le calcul de sept expressions linéaires (trois pour le contour, une pour la profondeur et trois pour la couleur). Environ 300 bits sont ainsi nécessaires pour définir une facette. Pixel-Planes 4 opérant en mode bit-série à 10 MHz, il faut environ 30 microsecondes pour afficher une facette, ce qui correspond à une performance globale d'environ 35000 facettes par seconde, quelque soit la taille des facettes.

Rendement

Le rendement de Pixel-Planes 4 (pour les algorithmes de rendu de facettes) est égal à la surface moyenne des facettes (en nombre de pixel) divisé par le nombre total de processeurs pixel. Pour des facettes de surface moyenne 100 pixels, le rendement est d'environ 1/2500.

Augmentation de la résolution d'écran

Le parallélisme massif pixel total impose d'associer un processeur à chaque pixel de l'écran. Pour passer d'une résolution 512×512 à une résolution 1024×1024, il est nécessaire de multiplier par quatre la complexité matérielle du système. Le rendement moyen du système est alors encore diminué (d'un facteur quatre), et les performances restent constantes. En effet, le temps d'affichage d'une facette sur Pixel-Planes dépend uniquement du nombre de bits à transférer dans les arbres d'additionneurs (l'augmentation de résolution accroit légèrement la hauteur de l'arbre d'additionneur, et donc le temps de transfert des coefficients, mais cet effet est négligeable car l'arbre est pipeliné).

Augmentation de la puissance du système en rendu de facettes

Il existe a priori deux moyens pour augmenter la puissance de Pixel-Planes 4 (en nombre de facettes par seconde). La première consiste à augmenter la fréquence d'horloge du système. Les performances sont alors multipliées par le même facteur (ceci ne concerne que les performances de l'unité de conversion). Cette solution ne modifie pas la complexité matérielle du système, mais dépend étroitement de la technologie VLSI utilisée et de la qualité du routage des circuits imprimés. Les technologies courantes utilisées aujourd'hui permettraient de construire un système fonctionnant à plus de 50 MHz.

La seconde solution consiste à paralléliser l'ensemble des opérations (calcul et transfert de données) effectuées dans l'arbre d'additionneur et dans les ALU des processeurs pixel. Par exemple, le passage à un système bit-série 8 bits (et non plus 1 bit) induirait une augmentation des performances d'un facteur 8 pour la plupart des opérations de rendu (notamment l'affichage de facettes). Cette solution entraînerait bien évidemment une augmentation de la complexité matérielle du système (en nombre de transistors) et des boîtiers utilisés pour les circuits intégrés (en nombre de broches).

Les deux solutions que nous venons de présenter peuvent naturellement être combinées.

Ainsi, un système Pixel-Planes 4 fonctionnant en mode bit-série 8 bits à 40 MHz aurait des performances théoriques de l'ordre du million de facettes par seconde (indépendamment de la taille des facettes).

2.2.1.2 le projet RC

Le projet RC [ATAM89], mené dans l'équipe graphique du LIFL, avait pour but d'étudier et de réaliser un réseau cellulaire complet pour l'affichage rapide d'images par la méthode de Gouraud. L'étude a porté notamment sur le type de processeur pixel, la méthode de transmission des objets, et les algorithmes de rendu utilisables dans une telle structure.

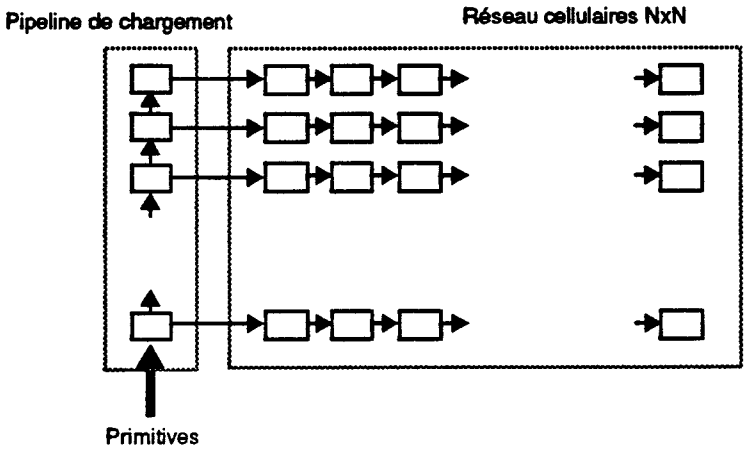


Figure 2.10 : architecture multipipeline de RC

L'architecture retenue est un réseau multi-pipeline couplé à un pipeline de découpage vertical (Figure 2.10). Chaque pipeline est composé de N processeurs pixel, N étant la résolution horizontale de l'écran. Un objet est décrit dans le pipeline horizontal par les valeurs suivantes :

- Xg, Xd : limites gauche et droite de l'objet sur la ligne courante.
- Z, dZx : profondeur de l'objet et incrément en x.
- R, dRx, V, dVx, B, dBx : couleur de l'objet et incréments en x.

Un processeur reçoit un objet de son voisin de gauche, calcule par interpolation la profondeur et la couleur de cet objet en son pixel, effectue l'élimination des parties cachées en son pixel (uniquement si l'objet est effectivement présent , i.e si le numéro du pixel est compris entre Xg et Xd), et transmet l'objet (dont le Z et la couleur ont été modifiés par l'interpolation) à son voisin de droite. Quand tous les objets de la ligne ont été traités, chaque processeur pixel contient l'objet visible et sa couleur.

Les différents pipelines horizontaux sont alimentés par un unique pipeline vertical effectuant les interpolations en y et le calcul des valeurs Xg et Xd. Un objet est décrit dans le pipeline vertical par les valeurs suivantes :

- Yb, Yh : limites basse et haute de l'objet.
- Xg, dXg, Xd, dXd : limites droite et gauche et incréments.
- Z, dZx, dZy : profondeur, incrément en x, incrément en y.

- R, dRx, dRy, V, dVx, dVy, B, dBx, dBy : couleur. incréments en x et en y.

Les objets sont transmis un par un dans le premier processeur du pipeline vertical.

Rendement, augmentation de la résolution et augmentation des performances

Les conclusions énoncées pour Pixel-Planes 4 restent vraies pour RC.

Le système proposé dans [ATAM89] effectue les transmissions et les interpolations en mode bit-série afin de réduire la complexité (en nombre de transistors et en nombre de liaisons).

Toutefois, si toutes les valeurs que nous venons de décrire sont effectivement transmises en parallèle dans le réseau, et si toutes les interpolations sont également effectuées en parallèle, un tel système est alors capable de génerer un objet (une facette) par cycle élementaire quelle que soit la taille de l'objet[1]. Ce système n'a pas été construit, mais une réalisation partielle, étudiée dans un autre cadre chez IBM, implémente un pipeline horizontal de 1024 processeurs pixel (ce système est décrit au paragraphe) en quatre circuits intégrés (un million de transistors par circuits). On peut donc estimer la complexité d'un réseau complet 512*512 à environ un milliard de transistors (ou encore 1000 circuits de un million de transistors chacun). En suposant une fréquence de fonctionnement de 40 Mhz, un tel système aurait des performances proches de 40 millions de facettes par seconde (quelque soit la taille des facettes). Bien évidemment, ces chiffres ne concernent que le module graphique.

Par ailleurs, il faut noter que ces chiffres représentent les performances maximales d'un tel réseau cellulaire. Toute augmentation de puissance (autre que celle obtenue par un changement de fréquence horloge) ne peut se concevoir que par l'introduction d'un second niveau de parallélisme de type MIMD (objet ou image).

2.2.2 Approche objet

De nombreux systèmes à parallélisme massif objet ont été proposés dans les années 80, mais à notre connaissance aucun n'a abouti à un prototype opérationnel. Tous ces systèmes reposent sur le principe suivant : à chaque objet de la scène à visualiser (le plus souvent des facettes) est associé un processeur appelé processeur objet. Celui-ci détermine en chaque pixel si l'objet dont il a la charge est présent, et si oui calcule au minimum la profondeur et la couleur de son objet. Un mécanisme de composition permet de combiner en chaque pixel les sorties des différents processeurs afin de déterminer l'objet visible (Figure 2.11). Si tous les processeurs objets travaillent dans l'ordre et à la vitesse du balayage écran, le décideur peut directement rafraîchir l'écran. Dans le cas contraire, il est indispensable d'utiliser une mémoire de trame.

1. Le temps d'amorçage du réseau (512+512 cycles) est négligeable devant le temps de génération d'une trame complète..

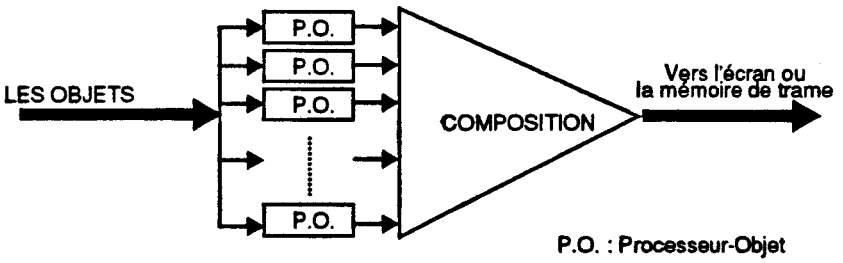


Figure 2.11 : principe du parallélisme massif objet

Les systèmes proposés diffèrent principalement par la structure du décideur. En effet, deux organisations architecturales sont envisageables pour combiner les sorties des N processeurs objet :

- les processeurs objets forment un pipeline (Figure 2.12). Dans ce cas, chaque processeur possède un opérateur de combinaison. Pour chaque pixel, il reçoit les caractéristiques (au minimum la profondeur et la couleur) de son prédécesseur, effectue la combinaison avec son propre objet, et transmet le résultat à son successeur. Notons que les processeurs ne traitent pas simultanément le même pixel. Le temps d'amorçage d'un tel pipeline est de N cycles (en considérant qu'un cycle représente le temps mis pour effectuer la composition).

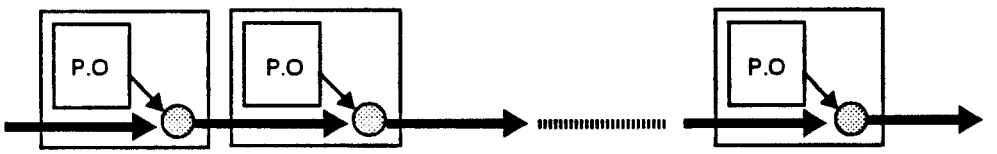


Figure 2.12 : organisation en pipeline

- le décideur est un arbre binaire pipeliné (Figure 2.13) composé de N-1 opérateurs (N étant le nombre de processeurs objet). Chaque noeud de l'arbre reçoit les caractéristiques de deux objets, effectue la combinaison et transmet le résultat au noeud suivant. Tous les processeurs traitent simultanément le même pixel. Le temps d'amorçage d'un arbre est de log(N) cycles.

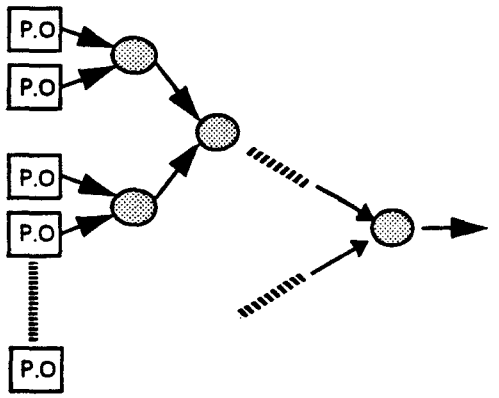


Figure 2.13 : organisation en arbre binaire

comparaison des deux approches

Les deux approches sont comparables sur le plan de la complexité "transistors", chacune nécessitant N-1 opérateurs de combinaison. Cependant, ces opérateurs peuvent être incorporés dans les processeurs objets dans le cas du pipeline, alors que l'arbre impose de définir des circuits spécifiques. De ce fait, l'implémentation matérielle (circuits intégrés et circuits imprimés) d'un arbre de processeurs objet entraîne une plus grande complexité (en terme de nombre de boîtiers et donc de surface, ainsi qu'en termes de nombre de pistes de routage sur les cartes) que dans le cas du pipeline.

Par ailleurs, l'opérateur de combinaison pose des problèmes de conception dans le cas de l'arbre. En effet, dans le cas le plus simple (Zbuffer avec valeurs RVB), le circuit doit posséder deux bus d'entrées et un de sortie comprenant chacun au mininum 40 signaux (Z sur 16 bits et RVB sur 24 bits), soit un total de 120 broches. Parallèlement, la complexité en transistors du circuit est minime (un Zbuffer nécessite uniquement un comparateur 16 bits, un multiplexeur 16 bits et un multiplexeur 24 bits). Il est bien evidemment possible de sérialiser les entrées/sorties, mais au prix d'une diminution importante des performances du système.

Ainsi, la réalisation d'une machine objet réalisant uniquement l'élimination des parties cachées dans un arbre binaire ne se justifie guère. Les seuls cas viables sont ceux nécessitant des opérateurs de combinaison plus complexes que le Zbuffer (par exemple l'élimination des parties cachées avec prise en compte de l'anti-aliassage [SHAW88]), ou des traitements incompatibles avec une structure pipeline (par exemple le traitement d'arbres CSG [KEDE89]). Dans tous les autres cas, l'approche pipeline est préférable à l'arbre binaire.

2.2.2.1 Architecture de Cohen et Demetrescu

Le premier pipeline objet avec élimination des parties cachées par Zbuffer a été proposé par Cohen et Demetrescu en 1980 [COHE80]. Chaque processeur objet calcule dans l'ordre du balayage écran la profondeur et la couleur (interpolation de Gouraud) de son objet. L'opérateur de combinaison effectue uniquement l'élimination des parties cachées entre l'objet de son processeur et celui du prédécesseur. Pour cela, les deux profondeurs sont comparées, la plus petite déterminant l'objet visible. L'ensemble du système est destiné à fonctionner au rythme du balayage, et peut ainsi rafraîchir directement l'écran sans utiliser de mémoire de trame.

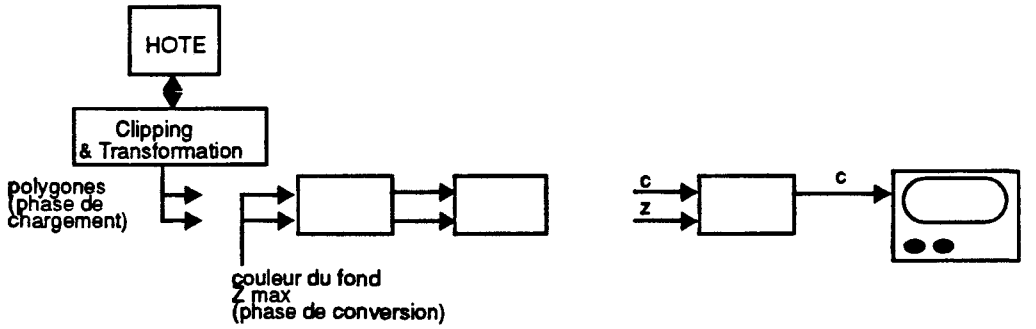


Figure 2.14 : pipeline de Cohen et Demetrescu

2.2.2.2 Architecture de Fussell et Rathi

Fussell et Rathi [FUSS82] ont proposé en 1982 de remplacer le pipeline proposé par Cohen puis Weinberg [WEIN81] par un arbre binaire de comparateurs. Par ailleurs, l'ensemble de leur système est décomposé en sous-unités, chacune gérant un nombre fixé d'objets. Une unité est composée d'une mémoire stockant les triangles à afficher, d'une unité de transformation et de clipping, d'une unité d'initialisation, d'un certain nombre de processeurs objets et d'un arbre de comparateurs (Figure 2.15). Le nombre de processeurs objets est déterminé par la puissance de l'unité de transformation géométrique (i.e. par le nombre de triangles qu'elle peut traiter en temps réel). Ce nombre est évalué à 1000 dans la description du système.

N unités fonctionnant en parallèle peuvent afficher 1000*N triangles en temps réel (la figure présente un exemple de système composé de deux unités). Les auteurs pensaient utiliser 25 unités pour obtenir une puissance de 25000 triangles affichés en temps réel. Il faut noter qu'à l'époque ceci représentait une augmentation de puissance d'un ordre de grandeur par rapport aux simulateurs de vol les plus performants.

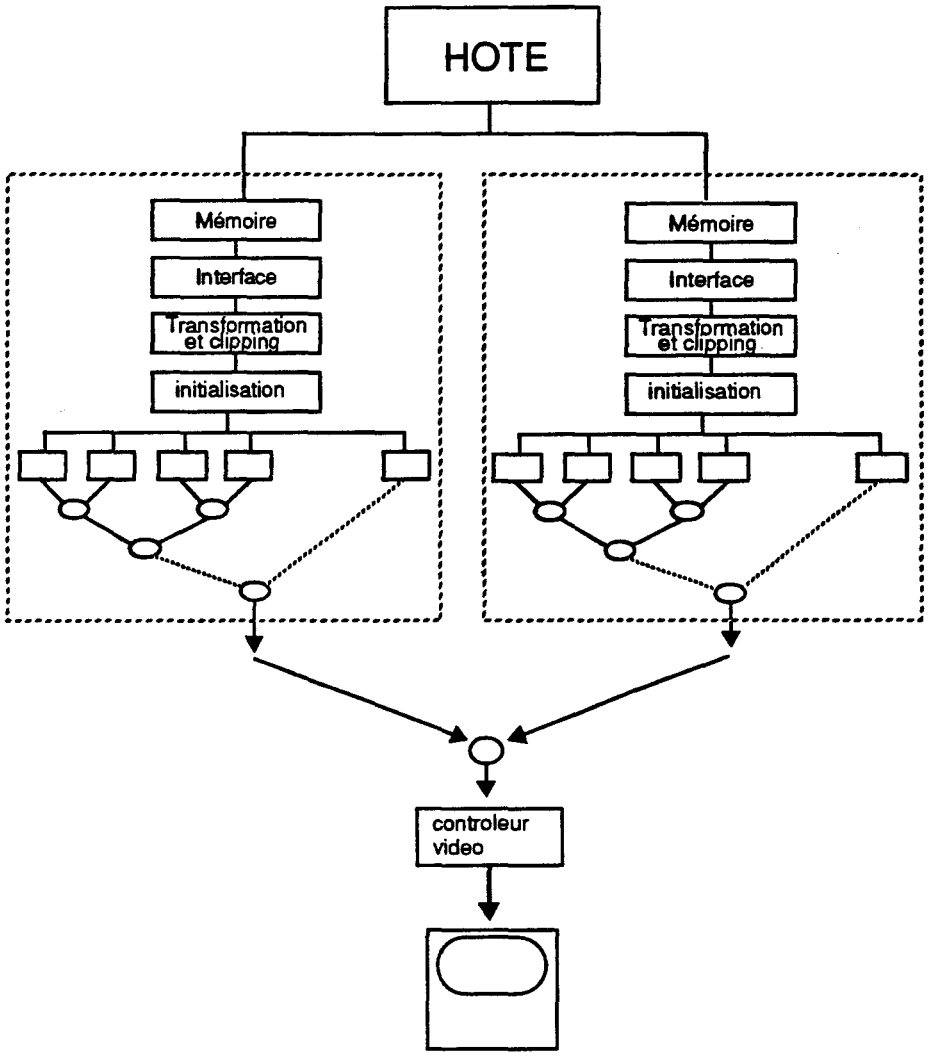


Figure 2.15 : un système composé de deux unités

Du point de vue théorique, l'approche proposée par Fussell et Rathi est très séduisante, car elle possède les avantages suivants :

- extensibilité aisée : pour augmenter la puissance du système, il suffit d'ajouter des unités de rendu.
- faible latence : la composition étant effectuée par un arbre binaire, la latence du système n'augmente que de façon logarithmique en fonction du nombre de processeurs objets utilisés.
- pas de mémoire de trame car l'ensemble du système fonctionne au rythme du balayage écran.
- débit uniforme : la parallélisation du processeur hôte effectuant les transformations géométriques est implicite puisque chaque unité contient une unité de transformation géométrique alimentant (en temps réel) ses processeurs objets.

A l'époque (début des années 80), ces arguments étaient recevables car les plus puissants systèmes graphiques (les simulateurs de vol professionnels) ne pouvaient traiter en temps réel que quelques milliers de facettes.

Si l'on considère aujourd'hui la réalisation effective d'un tel système, certains des arguments que nous venons de présenter peuvent se présenter sous un tout autre jour. Les principaux défauts que l'on peut attribuer à cette architecture sont les suivants :

- très faible rendement et donc très grande complexité matérielle.
- difficulté d'implémenter l'arbre de comparateurs (nécessité de développer des circuits possédant peu de logique interne mais un grand nombre d'entrées/sorties).
- difficulté d'adaptation aux résolutions des écrans actuels. En effet, pour pouvoir rafraîchir directement un écran $1280 \times 1024/70Hz$, l'ensemble des unités de conversion (processeurs objet et arbre de comparateurs) devraient fonctionner à une fréquence supérieure à $100MHz$.

2.2.2.3 L'architecture de Weinberg et la machine PROOF

Weinberg [WEIN81] a proposé en 1981 une modification du pipeline original de Cohen et Demetrescu pour effectuer un traitement d'anti-aliassage sur les images. Pour cela, le pipeline détermine en chaque pixel non plus uniquement l'objet visible, mais la liste de tous les objets potentiellement visibles, triés du plus proche au plus lointain. Pour cela, chaque processeur objet calcule, en plus de la profondeur de son objet, des informations relatives au pourcentage du pixel effectivement recouvert par l'objet. L'opérateur de combinaison, quant à lui, construit la liste des objets potentiellement visibles en tenant compte des profondeurs et des pourcentages de couverture. En sortie du pipeline, une série de filtres détermine la couleur finale de chaque pixel.

Ce schéma a été repris par une équipe de l'Université de Tübingen pour le développement de la machine PROOF [SCHN88a][SCHN88b]. Afin d'améliorer la qualité des images produites, les processeurs objets calculent non pas l'éclairement (défini par ses trois composantes R, V et B), mais le vecteur normal à l'objet (défini par ses trois composantes Nx, Ny et Nz). Un post-processeur d'éclairement calcule en sortie du pipeline la couleur des objets potentiellement visibles en chaque pixel (éclairement de Phong). Une série de filtres détermine ensuite la couleur effective de chaque pixel.

Dans ces deux systèmes, les informations circulant entre les processeurs objet (la liste des objets potentiellement visibles) n'ont pas une taille fixe. En effet, en chaque pixel, la liste des objets dépend de la scène à visualiser. Par ailleurs, la taille de la liste varie au cours de sa progression dans le pipeline. Le système doit donc fonctionner en mode MIMD, les processeurs se synchronisant par envoi de messages. Il est alors indispensable d'utiliser une mémoire de trame pour mémoriser l'image à afficher.

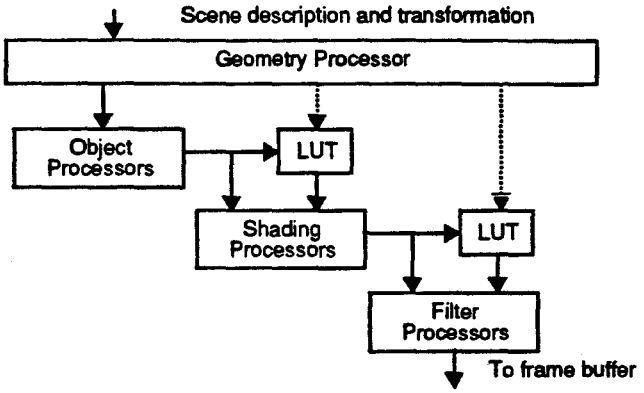


Figure 2.16 : la machine PROOF

A notre connaisance, le projet PROOF a abouti à la conception VLSI d'un processeur triangle et à l'étude du processeur de filtrage. Le processeur, réalisé à l'aide du compilateur de silicium GENESIL, comporte environ 85000 transistors. Par ailleurs, la relative faiblesse du logiciel utilisé ne permet pas d'utiliser une fréquence horloge supérieure à 10 MHz.

Une étude de faisabilité du post-processeur d'éclairement a également été menée [CLAU91].

2.2.2.4 La machine IMOGENE

Le projet IMOGENE [CHAI91][CHAI92a] s'inscrit dans le cadre des machines à parallélisme objet total. L'étude menée par notre équipe depuis quatre ans s'est principalement déroulée suivant trois axes de recherche : choix de l'architecture globale, étude de différents processeurs objet et étude d'un post-processeur d'éclairement.

L'architecture originalement proposée pour IMOGENE est présentée Figure 2.17 :

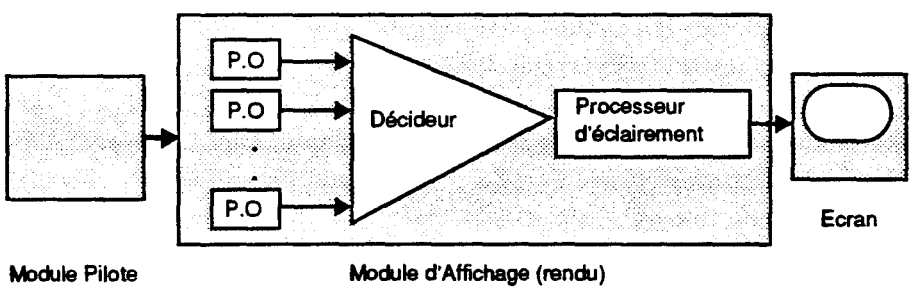


Figure 2.17 : architecture d'IMOGENE

Le module pilote est un ordinateur multiprocesseur d'usage général effectuant toutes les transformations géométriques sur les objets de la base de données. Chaque processeur s'occupe d'une partie des objets, et est connecté aux processeurs objets correspondants.

Chaque processeur objet calcule pour son objet dans l'ordre du balayage écran la profondeur, la normale et la couleur de base. Tous les processeurs travaillent de façon synchrone, traitant

en même temps le pixel courant. Le décideur est un arbre binaire pipeliné effectuant l'élimination des parties cachées. Les caractéristiques de l'objet visible (couleur et normale), disponibles en sortie du décideur, sont utilisés par le processeur d'éclairement pour calculer la couleur à afficher en chaque pixel. L'ensemble du système (processeurs objet, décideur et processeur d'éclairement) travaillant dans l'ordre et au rythme du balayage écran, le processeur d'éclairement peut directement rafraîchir l'écran.

Deux types de processeurs objet ont été étudiés : le processeur facette et le processeur quadrique . Tous les calculs d'interpolations sont effectués au moyen d'unités calculant dans l'ordre du balayage écran des expressions bilinéaires et quadratiques.

Excepté le processeur d'éclairement et les processeurs quadriques, l'architecture d'IMOGENE est très proche du système proposé par Fussell et Rathi. Les conclusions que nous avons énoncées lors de la description de ce système peuvent donc également s'appliquer à IMOGENE (bien évidemment si l'on ne considère que le rendu de facettes par la méthode de Gouraud).

Dans le cadre du projet IMOGENE nous avons réalisé le circuit intégré du Processeur Facette. La descrition complète du circuit étant le sujet du chapitre 4, nous nous bornerons ici à rappeler ses principales caractéristiques.

Le circuit est principalement composé de sept unités d'interpolation calculant chacune une expression linéaire. Chaque unité est composée d'un additionneur 24 bits, d'un regitre 24 bits et de deux mémoires de 3 mots de 24 bits. Le Processeur Facette étant destiné à fonctionner en arbre ou en pipeline, le circuit contient également une unité d'élimination des parties cachées composée d'un comparateur 24 bits et de multiplexeurs.

Le circuit a été réalisé en technologie CMOS 1.5 micron, contient environ 45000 transistors et fonctionne à 16 MHz. Il faut noter que cette complexité pourrait être réduite en utilisant des logiciels de conception VLSI plus performants que celui utilisé (logiciel SOLO 1400 de la société ES2).

La machine IMOGENE étant une machine objet, la complexité totale du système dépend du nombre total de processeurs objets. Par ailleurs, la résolution écran dépend de la fréquence de fonctionnement du système. Nous supposerons dans la suite que le système utilise un écran 512×512 directement rafraîchi par le pipeline de processeurs objet. La fréquence de fonctionnement d'un tel système est alors de 16MHz (écran 512×512, 50Hz, non entrelacé).

Si l'on suppose que le pipeline est composé de 1000 processeurs facettes, on obtient les résultats suivants :

- complexité totale du système (sans le processeur d'éclairement) : 45 Mtransistors
- performance : 50000 facettes/s (1000 facettes en temps réel à 50 Hz) quelle que soit la taille des facettes.

2.2.2.5 La Ray-Casting Machine

La Ray-Casting Machine [KEDE89][ELLI91] est un système à parallélisme objet spécifiquement conçu pour l'affichage d'images construites par combinaison CSG de quadriques. Contrairement aux machines objets que nous venons de présenter, il n'a pas pour but d'afficher en temps réel des images construites à partir de facettes, mais d'afficher en temps raisonnable (quelque dizaines de secondes) des images CSG complexes.

Les principaux éléments composant le système sont (Figure 2.18) :

- Les processeurs objets appelés PC (Primitive Classifier). Chaque PC permet de calculer incrémentalement en tout point de l'écran les profondeurs avant et arrière de l'objet dont il a la charge. Les objets utilisés par le Ray-Casting Engine sont les quadriques.
- Les processeurs de combinaison appelés CC (Combine Processor) permettant de réaliser les opérations logiques CSG (union, intersection et différence) entre les primitives.

Le système est composé de N PC et d'une matrice de NlogN CC. Les auteurs ont montré que tout arbre CSG peut être traité dans une telle matrice après avoir été réécrit de telle sorte que tout fils gauche d'un noeud de l'arbre soit une primitive. Par ailleurs, un mécanisme permet de construire en plusieurs passes un arbre possédant plus de N objets.

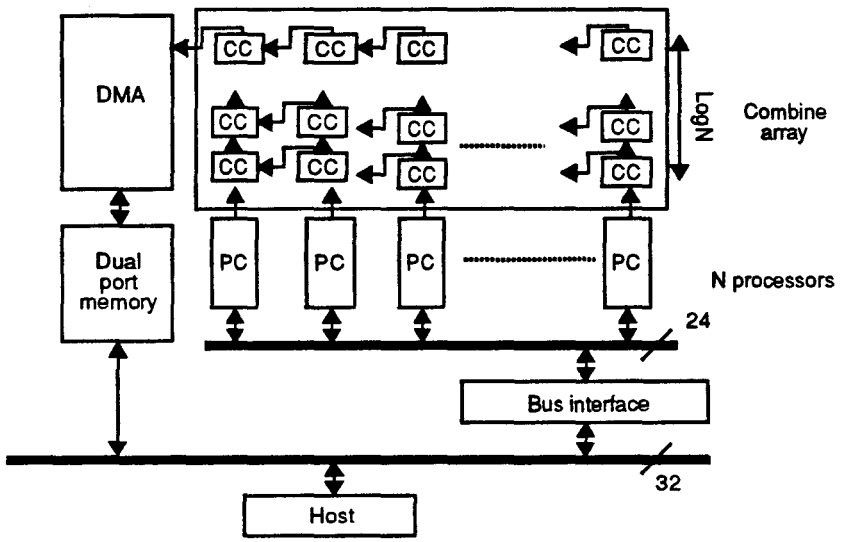


Figure 2.18 : schéma général de la Ray-Casting Machine

Un PC fournit un segment représentant la primitive qu'il traite au pixel courant (segment délimité par les profondeurs avant et arrière de la quadrique). Un CC reçoit de chacun de ses fils une liste de segments, et construit une nouvelle liste en fonction de l'opération CSG à réaliser. La Figure 2.18 présente un exemple simple de construction d'arbre CSG composé de deux objets. Dans le pire des cas (union d'objets disjoints en profondeur), la liste finale peut posséder autant de segments qu'il y a d'objets.

Le processeur CC situé en haut à gauche de la grille fournit pour chaque pixel de l'écran la liste de segments correspondante (Figure 2.19). Cette liste est transmise à l'ordinateur hôte qui peut alors effectuer différents traitements, dont les calculs d'éclairement. Pour afficher une image, seul le premier point du premier segment de chaque liste est pris en compte (ce qui correspond à une opération de Zbuffer).

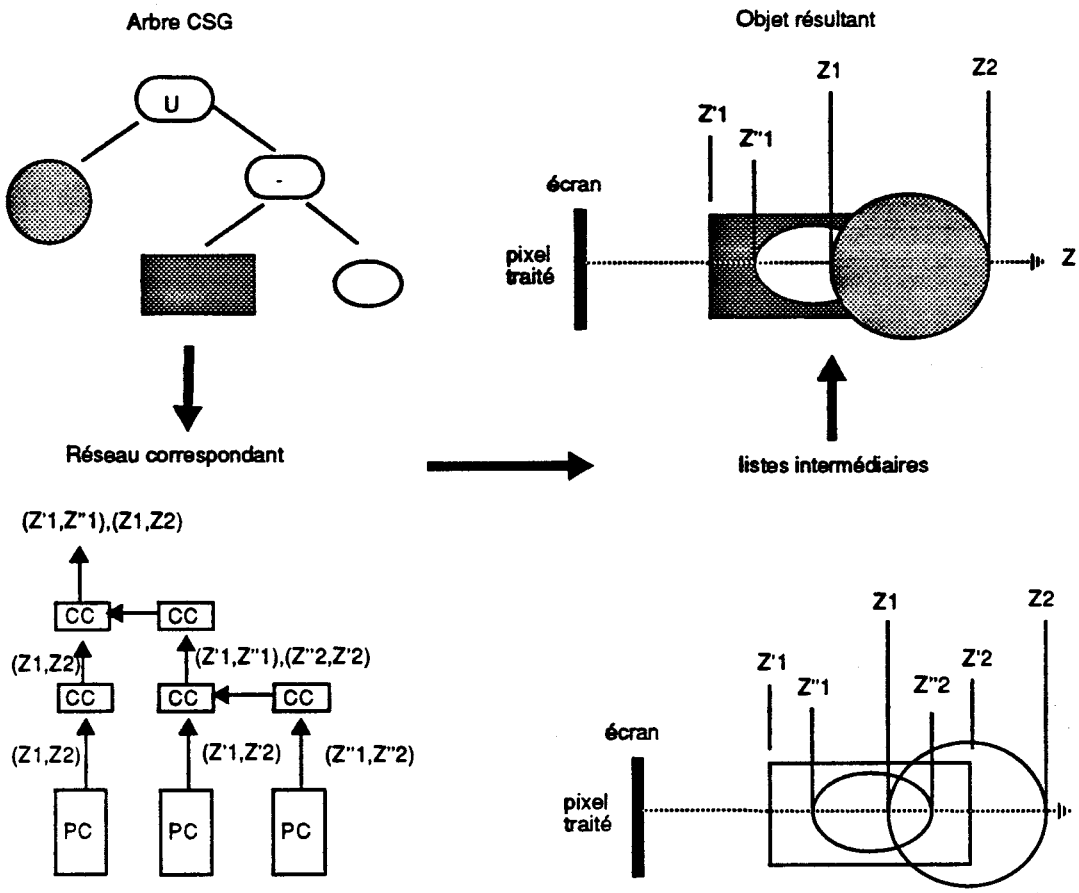


Figure 2.19 : exemple de construction d'un objet

L'ensemble du système fonctionne en mode bit-série. Les valeurs de profondeur sont calculées 2 bits à la fois dans les PC. La largeur des liaisons PC-CC et CC-CC est de 3 bits (un bit pour le premier point du segment, un bit pour le deuxième et un bit pour transmettre une valeur additionnelle comme la couleur du segment). Le dernier prototype réalisé comprend 256 PC et 2048 CC. Suivant la complexité de l'arbre CSG, le temps de calcul d'une image varie de quelques secondes à plusieurs minutes. Il est important de noter que, dans la version actuelle, les calculs d'éclairement sont effectués sur l'ordinateur hôte, ce qui pénalise grandement les performances globales du système. Celui-ci tirerait pleinement avantage d'un post-processeur d'éclairement

2.2.3 Comparaison des approches pixel et objet

Nous allons ici résumer les principales caractéristiques des approches architecturales que nous venons de présenter. En particulier nous insisterons sur le rapport complexité/ performances lors du rendu de facettes par la méthode de Gouraud.

Comme nous venons de le constater, de nombreuses études ont été menées sur les architectures massivement parallèles orientées objets. Parmi celles-ci, la Ray-Casting Machine se distingue par son champ d'application (affichage rapide d'objets CSG modélisés à l'aide de quadriques). Pour pouvoir comparer l'approche objet avec l'approche pixel (type Pixel-Planes 4) en rendu de facettes, nous nous limiterons à l'architecture objet minimale et dont les caractéristiques sont les suivantes :

- L'écran possède une résolution de 512 x 512 (fréquence trame 50 Hz, fréquence pixel 16 MHz).
- les processeurs objet fonctionnent dans l'ordre et au rythme du balayage écran, et traitent des triangles. Ils interpolent la profondeur et la couleur RVB (ombrage de Gouraud). Le système ne possède pas de mémoire de trame.
- l'élimination des parties cachées est effectuée en pipeline, chaque processeur étant muni d'un opérateur de Zbuffer.

La complexité d'un processeur objet peut être estimée à 30000-40000 transistors.

La Figure 2.20 présente une comparaison entre un système Pixel-Planes 4 (performances extrapolées pour une fréquence de 16 MHz) et un système IMOGENE composé de 1000 processeurs objet.

	PP4 16 MHz	IMOGENE 16 MHz
rendement	$\frac{S}{512^2}$	$\frac{S}{512^2}$
complexité	~40 Mtransistors (partie calcul)	~40 Mtransistors (1000 processeurs)
performance	55,000 triangles/s (quelle que soit la taille)	50,000 triangles/s (quelle que soit la taille)

S : surface moyenne des facettes

Figure 2.20 : comparaison Pixel-Planes 4 / IMOGENE

On peut ainsi constater que, *à complexité matérielle égale et à fréquence horloge égale*, les approches pixel (type Pixel-Planes 4) et objet (type IMOGENE) possèdent des performances équivalentes en rendu de facettes par la méthode de Gouraud.

2.2.4 Conclusion

Comme nous venons de le constater, les approches massivement parallèles pixel et objet, bien que diamétralement opposées, conduisent à des machines comparables au niveau des performances (en rendu de facettes) et de la complexité matérielle. Il est aujourd'hui admis que ces solutions, bien que *théoriquement* très séduisantes, conduisent à un rapport performances/coût beaucoup trop faible pour être commercialement viables.

Pour tenter d'améliorer le rendement de ces machines (et donc d'augmenter le rapport coût/ performance), des études ont été menées afin de réaliser des machines à parallélisme massif partiel avec réallocation dynamique des ressources (processeurs pixel ou processeurs objets)

pendant la création et l'affichage d'une image. Les primitives ne sont alors plus générées sur tout l'écran, mais seulement sur les zones où elles sont effectivement présentes.

2.3 Parallèlisme massif partiel séquentiel

Les systèmes à réalisation partielle se distinguent par la forme des zones. On distingue pricipalement les découpages en lignes et ceux en zones rectangulaires. Aucune machine à réalisation partielle séquentielle (une seule unité de conversion utilisée pour l'ensemble des zones de l'écran) et à découpage en zones rectangulaires n'a jamais été proposée. Les machines Pixel-Planes 5 et PixelFlow utilisent un tel découpage, mais ont été spécifiquement conçues pour utiliser plusieurs unités en parallèle (parallélisme multi-niveau). Aussi, dans cette partie, nous nous limiterons aux systèmes utilisant un découpage en ligne.

2.3.1 Introduction

Les machines de cette catégorie utilisent une version massivement parallèle de l'algorithme du Zbuffer par ligne pour afficher des facettes par la méthode de Gouraud ou de Phong. Pour cela, les facettes sont décomposées en segments horizontaux[1]. Pour chaque ligne de l'écran, tous les segments sont traités (élimination des parties cachées par Zbuffer).

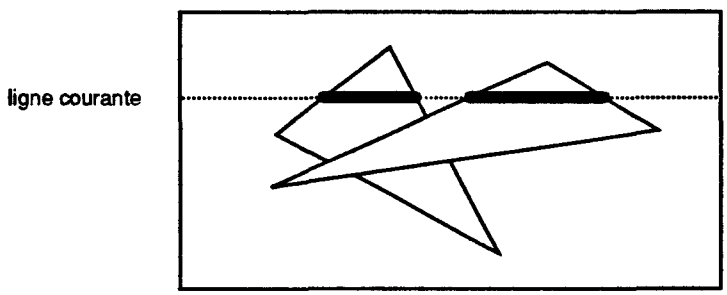


Figure 2.21 : découpage de triangles en segments horizontaux

2.3.1.1 Machines systoliques (ou pipelines)

Ces systèmes utilisent un pipeline de processeurs, dont la nature diffère suivant le type de parallélisme utilisé :

- approche pixel : le système utilise un pipeline de 1024 processeurs pixels pour traiter une ligne écran (résolution de 1024 x 1024). Les segments à traiter sont fournis au premier processeur du pipeline et circulent de la gauche vers la droite. Chaque processeur pixel recevant un objet effectue l'élimination des parties cachées en comparant la profondeur du nouvel objet avec celle de l'objet visible précedement mémorisé. Quand tous les segments de la ligne ont été traités, chaque processeur pixel contient l'objet visible. Un mécanisme permet alors de transférer les valeurs RVB correspondantes soit directement vers l'écran, soit vers une mémoire de trame.

1. horizontal spans en anglais

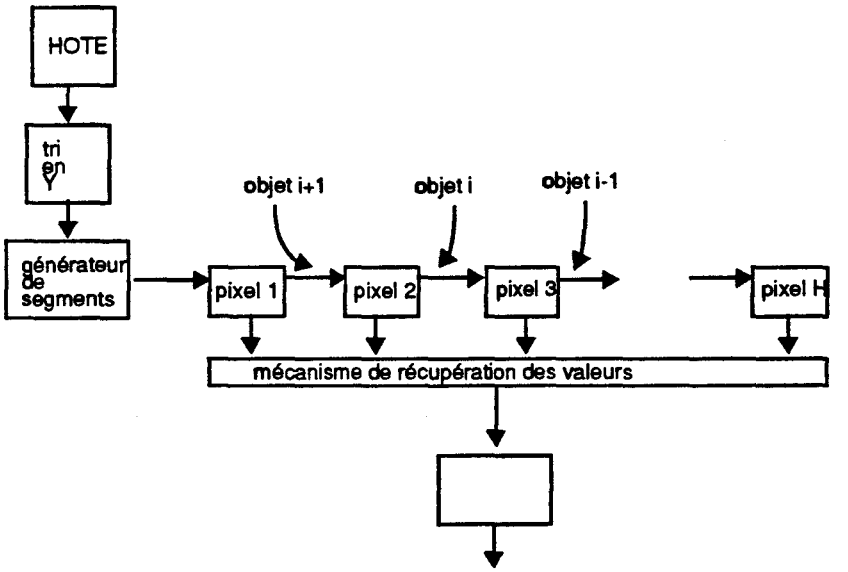


Figure 2.22 : pipeline de processeurs pixels

- approche objet : le pipeline de processeurs objet est identique à celui des machines objets massivement parallèles, mais les facettes ne sont générées que sur les lignes où elles sont effectivement présentes. Le dernier processeur du pipeline fournit dans l'ordre du balayage de la ligne les valeurs RVB à afficher. En fonctionnement temps réel sans mémoire de trame, le nombre de segments par ligne est limité par le nombre de processeurs.

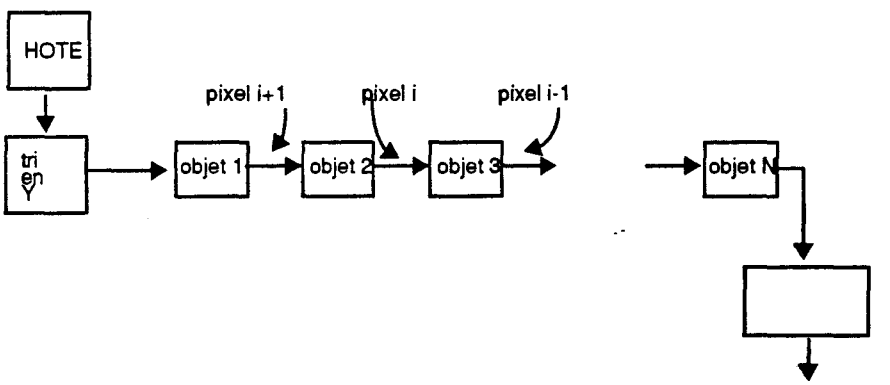


Figure 2.23 : pipeline de processeurs objet

2.3.1.2 Alimentation par arbre binaire

Il est également envisageable, dans le cas d'une machine pixel, d'alimenter les processeurs non pas via un pipeline mais via un arbre de diffusion (version linéaire de l'arbre de diffusion bilinéaire de Pixel-Planes 4). Toutes les conclusions que nous énoncerons à propos du pipeline de processeurs pixels s'appliquent également à un tel système.

Nous présentons maintenant deux exemples concrets de machines systoliques (ou pipeline) par ligne proposées en 1988, et ayant abouti à la conception de circuits intégrés spécifiques[1] : SAGE (machine pixel) et GSP-NVS (machine objet)

2.3.2 Approche pixel : SAGE

SAGE (Systolic Array Graphics Engine) [GHAR88] est une machine pixel utilisant un processeur par pixel pour une ligne écran, et reprenant les concepts développées dans le projet SUPER BUFFER [GHAR85]. Les objets traités sont des segments horizontaux repérés par leurs abscisses droites et gauches (Xr et Xl). Les processeurs pixel sont connectés en pipeline, le chargement des segments s'effectuant par le premier processeur de la ligne. Chaque processeur calcule incrémentalement pour son pixel les valeurs Z et RGB du segment en cours de traitement, effectue l'élimination des parties cachées en comparant la valeur Z ainsi calculée avec celle du segment visible, et transmet à son voisin les valeurs Z et RGB qu'il vient de calculer, ainsi que les incréments DZ et DRGB. Quand tous les segments ont été traités, chaque processeur pixel de la ligne contient la valeur RGB à afficher. La Figure 2.24 présente le principe d'un processeur pixel.

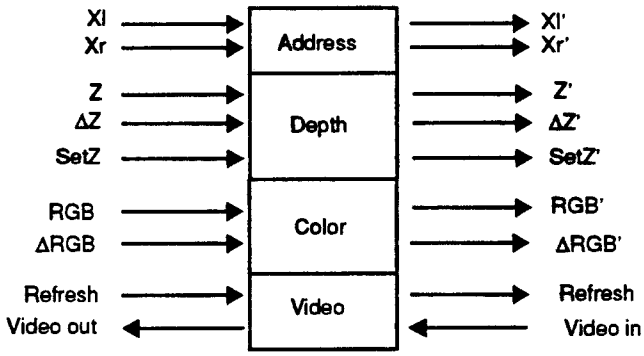


Figure 2.24 : le processeur pixel de SAGE

L'unité "Address" est composée de deux comparateurs et permet de déterminer si le pixel traité par le processeur appartient au segment en cours de traitement (comparaison du numéro du pixel avec les valeurs Xl et Xr). L'unité "Depth" est composée d'un registre stockant le profondeur du segment visible, d'un comparateur pour comparer cette profondeur avec celle du segment en cours de traitement, et d'un additionneur pour calculer incrémentalement la profondeur du segment (Z+dZ). L'unité Color est composée de trois registres stockant les valeurs RGB du segment visible, et de trois additionneurs pour calculer incrémentalent les valeurs RGB du segment en cours de traitement.

Pour un écran 1024×1024, le système SAGE utilise un pipeline de 1024 processeurs. Toutefois, ces processeurs n'effectuent que des interpolations en x des valeurs Z et RGB. Les interpolations en y sont effectuées par un circuit spécifique (appelé générateur de commande ou générateur de segments) calculant pour tous les segments actifs de la ligne en cours de traitement les valeurs initiales Xl , Xr, Z et RGB, ainsi que les incréments correspondants. Les valeurs ainsi calculées sont envoyées au premier processeur pixel du pipeline (un segment toutes les 40 ns).

1. les systèmes complets n'ont cependant jamais été construits.

Le circuit intégré développé pour le projet SAGE contient 256 processeurs pixel. Il intègre 1 million de transistors, et est capable de générer un pixel toutes les 40 ns. La complexité d'un processeur pixel est donc d'environ 4000 transistors. Un système complet pour un écran 1024 x 1024 nécessite quatre circuits intégrés. Deux configurations peuvent être utilisées pour connecter les circuits :

- en configuration série, le premier circuit génère les pixels de 0 à 255, le deuxième les pixels de 256 à 511, et ainsi de suite. Le rythme de sortie des valeurs Video RGB est alors d'une toute les 40 ns, ce qui est insuffisant pour rafraîchir directement un écran 1024 x 1024 (11 ns). Une mémoire de trame externe est donc indispensable.

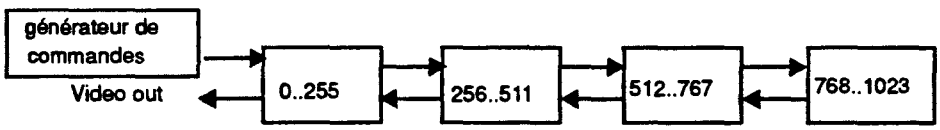


Figure 2.25 : 4 circuits SAGE en série

- en configuration parallèle, le premier circuit génère les pixels 0, 4, 8, 12... le second génère les pixels 1, 5, 9, 13 et ansi de suite. Un sérialiseur permet de regrouper les sorties Vidéo des quatre circuits. Le rythme de sortie des valeurs Video RGB est alors d'une toute les 10 ns, ce qui permet de rafraîchir directement l'écran. Toutefois, seuls 256 segments peuvent être traités sur une ligne donnée.

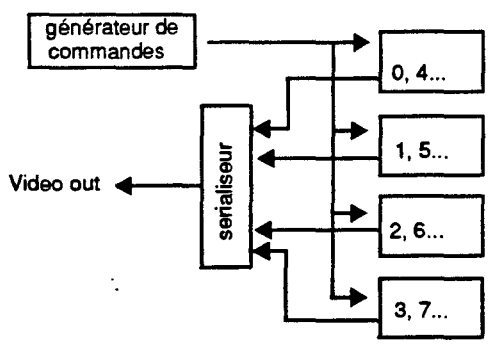


Figure 2.26 : 4 circuits SAGE en parallèle

En configuration parallèle sans mémoire de trame, le système SAGE est capable de traiter 256 segments par ligne, quelle que soit la longueur des segments. Les performances totales du système dépendent uniquement de la hauteur moyenne des facettes. Pour des facettes de hauteur moyenne 32 pixels, les concepteurs du système annoncent des performances proches de 1 million de facettes par seconde.

Une telle configuration représente un total d'environ 4 millions de transistors (il faut toutefois ajouter la complexité du générateur de commandes qui réalise les interpolations en y). D'après les articles dont nous disposons, un tel système ne gérerait que 8 bits par pixel pour calculer la couleur. Un système vraies couleurs (24 bits par pixel) nécessiterait alors 12

circuits, soit une complexité totale de 12 millions de transistors.

Rendement

Le rendement moyen du système SAGE est égal à la largeur moyenne des facettes (sur une ligne) divisé par 1024. Il ne dépend pas de la hauteur moyenne des facettes.

Augmentation de la résolution d'écran

Etant donné le principe même du parallélisme pixel par ligne (un processeur par pixel de la ligne courante), toute augmentation de la résolution horizontale entraîne une augmention similaire du nombre de processeurs pixels utilisés. Par contre, une augmentation de la résolution verticale n'entraîne qu'une augmentation de la taille de la mémoire de trame.

Augmentation de la puissance du système

La seule solution possible pour augmenter la puissance du système est de réduire le temps de cycle du pipeline. Pour cela, on peut soit augmenter la fréquence de l'horloge (les circuits doivent alors être redessinés), soit augmenter le nombre de processeurs pixels utilisés en configuration parallèle. La première solution pose des problèmes technologiques, la seconde augmente la complexité matérielle du système. Il est bien évident que, dans les deux cas, le générateur de segments alimentant le pipeline doit pouvoir bénéficier du même accroissement de puissance.

2.3.3 Approche objet : GSP-NVS

GSP-NVS [DEER88] est l'approche duale de SAGE en parallélisme objet. L'approche retenue est similaire à celle des systèmes à parallélisme objet présentées au paragraphe 2.2.2, mais les performances sont grandement améliorées car les objets (des triangles) ne sont plus générés sur tout l'écran, mais uniquement sur les lignes où ils sont effectivement présents.

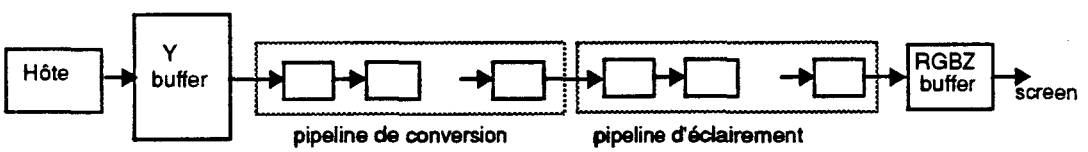


Figure 2.27 : principe de GSP-NVS

Le principe de fonctionnement de GSP-NVS est le suivant : après que le hôte ait effectué les transformations géométriques, l'unité Y-buffer constitue 1024 listes (pour un écran composé de 1024 lignes) de triangles. Chaque liste contient l'ensemble des triangles dont le sommet supérieur appartient à la ligne considérée. Une fois ce classement effectué, les triangles sont chargés dans le pipeline effectuant la conversion. A chaque début de ligne, tous les triangles de la liste correspondante sont chargés dans le pipeline, qui effectue alors la conversion des objets et l'élimination des parties cachées. L'objet visible en chaque pixel de la ligne, caractérisé par sa profondeur, son vecteur normal et sa couleur de base, est ensuite récupéré par un second pipeline effecuant tous les calculs d'éclairement (formule de Phong avec cinq

sources lumineuses) et de texture.

Si sur la ligne courante il y a plus de segments que de processeurs disponibles, les triangles qui ne peuvent être traités sont conservés dans le Y-buffer et traités lors d'une seconde passe. Lors de cette deuxième passe, seules les lignes surchargées sont prises en compte, ce qui permet de limiter considérablement le surcoût temporel (les concepteurs de GSP-NVS estiment qu'il ne dépasse pas 20% pour de scènes moyennement surchargées).

Il est important de noter que le chargement d'un triangle n'est effectué qu'à la première ligne où il apparaît. Une fois attribué à un processeur, le triangle y reste jusqu'à ce qu'il ait été traité sur toute sa hauteur[1]. Ainsi, contrairement au système SAGE, le découpage des triangles en segments horizontaux n'est pas effectué par une unité spécialisée gérant l'ensemble des objets de la base de données, mais par chaque processeur qui détermine à chaque début de ligne les coordonnées gauche et droite de son segment (intersection de son triangle avec la ligne courante).

Les principales composantes du processeur objet de GSP-NVS sont les suivantes (Figure 2.28) :

- L'unité X-Unit interpole à chaque début de ligne les valeurs Xl et Xr du triangle. Par ailleurs, en comparant ces valeurs avec l'abscisse du pixel courant (calculée par l'unité de contrôle), elle indique aux autres unités si le triangle est présent ou non au pixel considéré.
- L'unité Z-Unit calcule par interpolation la profondeur du triangle en chaque pixel de la ligne, et compare cette valeur avec celle provenant du processeur précedent afin de déterminer le triangle visible au pixel courant.
- Les unités N-Unit interpolent en chaque pixel les trois composantes de la normale.

Notons que les unités Z-Unit et N-Unit effectuent des interpolations bilinéaires (en x et en y). Les interpolations en y sont effectuées à la fin de chaque ligne écran, alors que les interpolations en x sont effectuées à chaque pixel.

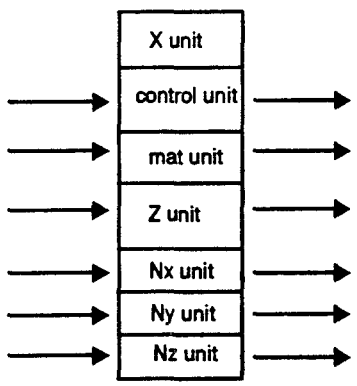


Figure 2.28 : schéma (très) simplifié d'un processeur objet de GSP-NVS

1. Pour cette raison, GSP-NVS est souvent classé comme machine du type "virtual processors" et non pas "virtual scan-line buffer".

Un circuit intégré comprenant un processeur triangle a été développé par l'équipe de GSP-NVS. Il contient environ 25000 transistors et génère un pixel toutes les 50 ns. Aucun prototype n'a cependant été construit à partir de ces circuits.

bénéfices apportées par la réalisation partielle

Pour un nombre donné de processeurs objets, la réalisation partielle par ligne permet d'obtenir un accroissement considérable des performances du système. Prenons comme exemple un pipeline composé de 100 processeurs objets fonctionnant à 16 MHz.

Si chaque processeur génère un objet sur tout l'écran (parallélisme massif total), le système peut afficher 100 objets en temps réel sur un écran 512 x 512 à 50 Hz (quelque soit la taille des objets), soit des performances de 5000 facettes par seconde.

Si au contraire une organisation partielle par ligne est adoptée, le système peut alors générer 100 objets par ligne quelque soit la largeur des objets. Les performances du système dépendent donc à la fois de la taille (en fait de la hauteur) des objets et de leur répartition spatiale.

Le surcoût induit par la réalisation partielle réside principalement dans le classement des objets et possède à la fois un caractère temporel (le classement des objets prend un certain temps) et matériel (il faut disposer d'une zone mémoire importante pour mémoriser les 1024 listes d'objets).

Rendement

Le rendement de GSP-NVS est le même que celui de SAGE (largeur moyenne des facettes sur une ligne divisé par la longueur de la ligne).

Augmentation de la résolution d'écran

L'augmentation de la résolution de l'écran n'entraîne qu'une augmentation de la taille de la mémoire de trame globale. Une modification de la largeur des interpolateurs doit également être envisagée pour prévenir tout rique de dépassement de capacité.

Augmentation de la puissance du système

Deux solutions sont envisageables pour augmenter la puissance du système. La première consiste à augmenter le nombre de processeurs objet dans le pipeline, permettant d'augmenter d'autant le nombre d'objets que le système peut traiter en une passe sur chaque ligne. Il faut cependant noter que dans certains cas, augmenter le nombre de processeurs ne modifie en rien les performances, car il est inutile de disposer de plus de processeurs qu'il n'y a d'objets sur la plus chargée des lignes de l'écran. Cette remarque illustre une nouvelle source d'inefficacité propre aux machines objets : certains processeurs peuvent être passifs si aucun objet ne leur sont affectés (ce point sera analysé plus en détail dans le chapitre suivant).

La seconde solution consiste à réduire le temps de cycle du pipeline. Les performances sont alors augmentées quelle que soit la répartition des facettes dans la scène à afficher.

2.3.4 Comparaison des deux approches

Nous allons tenter ici de comparer les machines SAGE et GSP-NVS (1000 processeurs), notamment sur le plan de la complexité et des performances. Cette comparaison s'appuie sur les chiffres cités dans les articles originaux décrivant ces systèmes.

	SAGE	GSP-NVS
rendement	$\frac{L}{1024}$	$\frac{L}{1024}$
complexité	~25 Mtransistors	~12 Mtransistors
performance	~1 million triangles/s	~1 million triangles/s

Figure 2.29 : comparaison SAGE/GSP-NVS

On constate encore que, malgré l'utilisation de types de parallélisme diamétralement opposés, ces deux systèmes ont des performances similaires. La différence de complexité s'explique facilement. En effet, les processeurs de GSP-NVS effectuent des interpolations bilinéaires (en x et en y), alors que ceux de SAGE n'effectuent que des interpolations linéaires (en x).

2.3.5 Equivalent en monoprocesseur

Les systèmes que nous venons de présenter utilisent une unité de conversion massivement parallèle et un découpage en lignes. Il faut signaler ici que de nombreux systèmes non massivement parallèles utilisant un tel découpage ont été proposés. L'algorithme utilisé est alors un Z-buffer en ligne[1]. Un tel système est composé des éléments suivants (nous supposerons que les primitives graphiques sont des triangles) :

- un générateur de segments horizontaux calculant pour chaque ligne écran les coordonnées droites et gauches (Xg et Xd) de chaque triangle présent sur la ligne. Il effectue également les interpolations verticales pour le calcul de la profondeur et des valeurs RVB (ou des valeurs Nx, Ny et Nz dans le cas d'un ombrage de Phong).
- un générateur de pixel effectuant pour chaque segment (i.e pour les pixels compris entre Xg et Xd) les interpolations horizontales sur la profondeur et la couleur.
- une mémoire d'une ligne mémorisant pour chaque pixel de la ligne la profondeur et la couleur, et un opérateur de Z-buffer permettant d'effectuer l'élimination des parties cachées.
- une mémoire de trame.

1. Scan-line Z-buffer

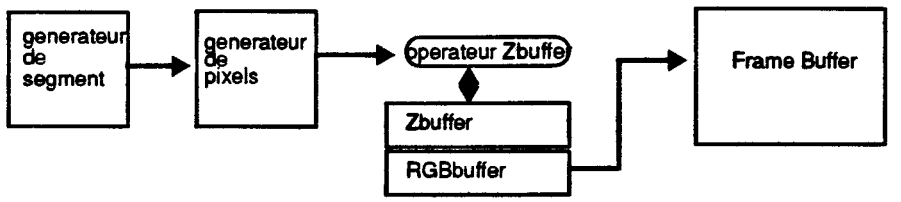


Figure 2.30 : Zbuffer par ligne

Le générateur de segments est quasiment identique au générateur de commandes alimentant le pipeline SAGE. Le générateur de pixel est construit à partir d'interpolateurs linéaires (un pour la profondeur et trois pour la couleur) semblables à ceux utilisés sur GSP-NVS (plus simples car ils n'effectuent que des interpolations en x).

Quand tous les segments de la ligne courante ont été traités, la mémoire ligne RGB est transférée dans la ligne correspondante de la mémoire de trame globale. Un tel système pourrait également être utilisé pour fonctionner au rythme du balayage écran sans utiliser de mémoire de trame si chaque ligne est générée en moins de 1024*10 ns. Le nombre de segments par ligne est alors limité.

Un système semblable à celui que nous venons de décrire est actuellement en cours de réalisation chez Apple [KELL92]. La principale différence réside dans le fait que le générateur de segments n'effectue pas d'interpolations verticales, mais calcule directement les valeurs Xg et Xd à l'aide de multiplieurs de de diviseurs. Deux circuits spécifiques ont été développés. Le premier est le générateur de segments, le deuxième intègre le générateur de pixels et la mémoire de ligne. Un tel système cadencé à 40 MHz est annoncé à 220.000 facettes/s (facettes de 100 pixels).

2.4 Parallélisme image haut niveau

2.4.1 Rappel

Les architectures de ce type utilisent un parallélisme image haut niveau (plusieurs unités de conversion travaillant en parallèle sur des zones disjointes de l'écran), et un parallélisme massif bas niveau (chaque unité de conversion est une machine massivement parallèle).

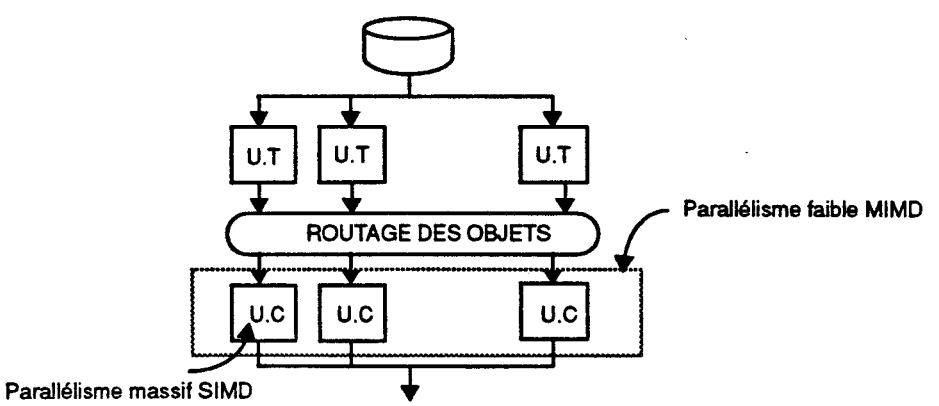


Figure 2.31 : parallélisme image haut niveau

Les architectures se distinguent principalement par le type de parallélisme utilisé dans les unités de conversion. Nous présentons dans la suite le système Pixel-Planes 5, dont les unités de conversion sont des machines massivement parallèles pixel.

2.4.2 Parallélisme image bas niveau : Pixel-Planes 5

Comme nous l'avons constaté lors de la présentation de Pixel-Planes 4, cette machine possède un très faible rendement, et en conséquence une grande complexité matérielle (pour des performances somme toute limitées). Afin de pallier ces défauts, l'équipe de Henri Fuchs a proposé une nouvelle architecture, appellée Pixel-Planes 5 [FUCH89], basée sur la technique du partitionnement de l'écran.

L'unité de base de Pixel-Planes 5, appelé Renderer (Figure 2.32), est une machine pixel réduite de 128 x 128 processeurs pixel (s'occupant donc d'une zone écran de 128 x 128 pixels). Les principales modifications architecturales apportées sur l'unité de conversion par rapport à Pixel-Planes 4 sont les suivantes :

- l'arbre d'additionneurs a été modifié afin de pouvoir évaluer des expressions quadratiques et non plus linéaires.
- chaque processeur pixel dispose en interne de 208 bits de mémoire.
- le Renderer dispose d'une mémoire auxiliaire externe de 4K bits par pixel (réalisée à l'aide de composants VRAM).
- la fréquence horloge a été augmentée (40 MHz).

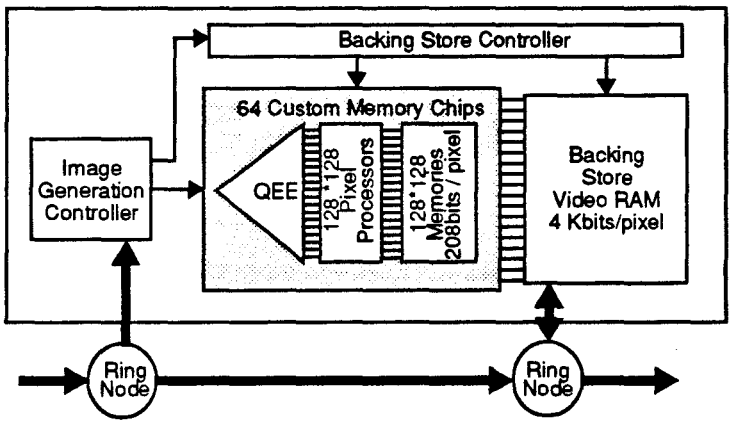


Figure 2.32 : Le Renderer de Pixel-Planes 5

Plusiers Renderers sont utilisés en parallèle pour traiter simultanément des zones disjointes de l'écran. Par ailleurs, tous les calculs géométriques sont effectués par plusieurs processeurs graphiques i860. Les Renderers et les processeurs graphiques sont connectés à un anneau à haut débit (8 voies à 80 Moctets/s chacune) assurant toutes les communications entre les différentes unités. La Figure 2.33 présente un schéma global de l'architecture de Pixel-Planes 5.

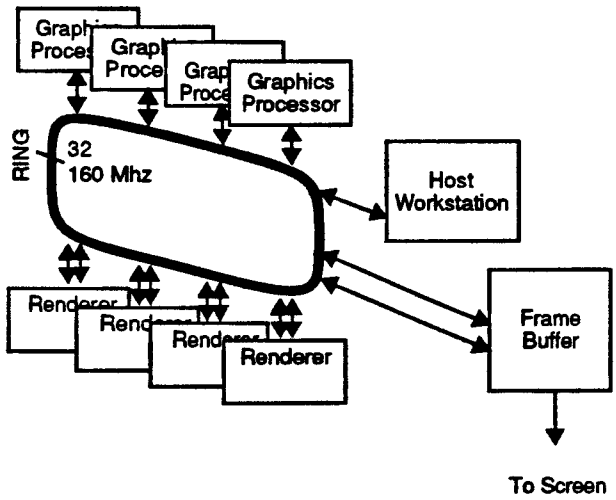


Figure 2.33 : schéma général de Pixel-Planes 5

Le principe de fonctionnement de Pixel-Planes 5 est le suivant :

L'écran (1280*1024) est partitionné en 80 zones disjointes de 128*128 pixels. Les objets à afficher, répartis entre les différents processeurs graphiques, sont classés en fonction des zones auquelles ils appartiennent. Les Renderers sont ensuite alloués aux premières zones de l'écran, et recoivent des processeurs graphiques les primitives à afficher. Dès qu'un Renderer a traité tous les objets de la zone dont il a la charge, le contenu de sa mémoire (valeurs RVB) est transféré via l'anneau dans la partie correspondante de la mémoire de trame globale, et le Renderer peut alors s'occuper d'une nouvelle zone écran[1]. Quand toutes les zones ont été traitées, le contenu de la mémoire de trame est affiché à l'écran.

La complexité totale de Pixel-Planes 5 est délicate à évaluer, en partie parce que le système utilise une quantité considérable de mémoire externe (8 Moctets de VRAM externe par Renderer).

Le Renderer est réalisé à partir de 64 circuits intégrés implémentant chacun 256 processeurs pixel avec leur mémoire et l'arbre de génération correspondant. Ces circuits sont réalisés en technologie CMOS 1.6 micron, intègrent chacun 417,000 transistors, et fonctionnent à 40 MHz (tout comme Pixel-Planes 4, l'ensemble fonctionne en mode bit-série). La complexité d'un Renderer (sans le mémoire externe) est donc d'environ 25 Mtransistors.

L'augmentation de la fréquence d'horloge permet à un Renderer d'atteindre une performance de 150000 triangles/s (4,4 fois plus rapide que Pixel-Planes 4). Toutefois, un système Pixel-Planes 5 composé d'un unique Renderer n'atteindrait pas ce seuil. En effet, toute primitive appartenant à plusieurs zones écran doit être traitée plusieurs fois, ce qui diminue les performances du système. Des simulations effectuées sur plusieurs scènes ont indiqué des performances effectives de l'ordre de 100.000 facettes/s (pour des facettes de petite taille).

L'utilisation de plusieurs Renderers en parallèle permet dans de nombreux cas de multiplier les performances du système par le nombre de Renderers. Ainsi, un système composé de dix

1. Pixel-Planes 5 utilise donc une réalisation partielle parallèle.

Renderers atteindrait des performances proches du million de facettes par seconde (pour des facettes de 100 pixels). Les cas défavorables sont ceux possédant une très grande disparité de charge entre les zones écran. Par exemple, dans le cas extrême où toutes les primitives appartiennent à une seule zone écran, seul un Renderer est effectivement utilisé.

La plus puissante configuration de Pixel-Planes 5 effectivement construite comprend 56 processeurs i860 pour les transformations géométriques, et 24 Renderers pour la conversion. La puissance de ce système est d'environ 2,4 millions de facettes par seconde (méthode de Phong).

Bénéfices apportés par rapport à Pixel-Planes 4

Les principales améliorations architecturales apportées par rapport à Pixel-Planes 4 peuvent être regroupées en trois catégories :

(1) évaluation d'expressions quadratiques et non plus linéaires.

(2) réalisation partielle 128×128.

(3) parallélisme MIMD entre les différents Renderers.

La première amélioration permet principalement d'afficher directement des primitives graphiques de plus haut niveau que les facettes (sphères et cylindres). Ce sujet sera traité au chapitre 5. En ce qui concerne l'affichage de facettes, cette modification architecturale n'a aucune conséquence sur les performances du système.

La deuxième amélioration permet de diminuer la complexité du système tout en conservant des performances quasi-identiques lors de la création d'images composées de petites primitives (100 pixels par exemple).

Considérons le cas d'un système Pixel-Planes 5 composé d'un unique Renderer 128×128[1]. Celui-ci traite alors toutes les facettes de la première zone, puis toutes celles de la deuxième zone, et ainsi de suite. Si l'on considère une scène composée de N facettes, le Renderer devra traiter M facettes, avec M>N. En effet, toute facette doit être traitée dans chacune des zones où elle apparaît (voir Figure 2.34). Par ailleurs, le temps de génération d'une facette ne dépend pas de la taille de celle-ci (ceci est une caractéristique des unités de conversion massivement parallèles), et, *à fréquence d'horloge égale*, le temps d'affichage d'une facette sur le réseau Pixel-Planes 4 512*512 est le même que sur le Renderer 128×128. Ainsi, le temps d'affichage d'une facette appartenant à quatre zones (par exemple) sera quatre fois plus long sur le Renderer de Pixel-Planes 5.

De plus, le rapport M/N (que l'on peut interpréter comme le pourcentage de facettes appartenant à plusieurs zones) augmente quand la taille moyenne des facettes augmente. Ainsi, un système Pixel-Planes 5 composé d'un unique Renderer aurait, à fréquence égale, des performances inférieures (20% pour des facettes de 100 pixels) à celles de Pixel-Planes 4, mais pour une complexité matérielle diminuée d'un facteur 16 dans le cas d'un écran 512×512 (si l'on considère un système 1280×1024, le gain est alors de 80). En contrepartie, il est nécessaire d'effectuer un classement des facettes en fonction des zones écran, et donc de disposer d'un hôte plus puissant.

1. i.e un système utilisant une réalisation partielle séquentielle

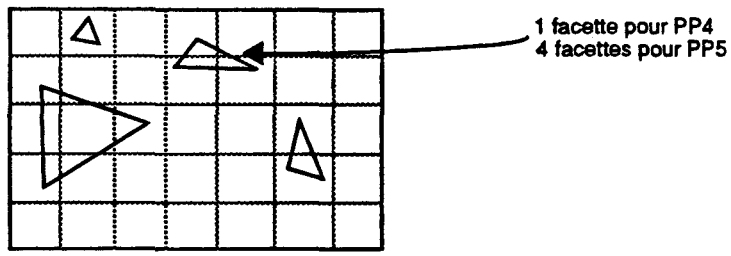


Figure 2.34 : duplication de primitives sur Pixel-Planes 5

Il est intéressant de noter que la taille des facettes peut grandement influencer les performances d'un tel système. Toutes les applications nécessitant l'affichage de grandes primitives sont fortement pénalisées par le taux de duplication important. Par exemple, lors du calcul d'ombres portées par la méthode de Crow, les primitives utilisées sont les polyèdres d'ombres générés par les facettes de la base de données. Ces polyèdres ont en général une taille très importante, et par conséquent induisent un taux de duplication élevé. Ainsi, l'un des points forts de Pixel-Planes 4 [FUCH85] disparaît sur Pixel-Planes 5 du fait de la réalisation partielle utilisée.

La troisième amélioration apportée par rapport à Pixel-Planes 4 permet, dans le meilleur des cas, une augmentation linéaire des performances en fonction du nombre de Renderers. Elle utilise un parallélisme de zones écran (plusieurs zones sont traitées en parallèle par des unités différentes). L'augmentation de performances dépend principalement de la répartition des primitives sur l'écran, et donc de l'équilibrage de charge entre les différents Renderers.

Rendement

Le rendement d'un Renderer est égal à la surface moyenne des facettes (dans la zone 128×128) divisé par la taille de la zone. L'ensemble des Renderers fonctionnant en parallélisme MIMD, le rendement d'un système complet est égal à celui d'un unique Renderer.

Augmentation de la résolution d'écran

L'augmentation de la résolution écran n'entraîne au niveau matériel qu'un accroissement de la taille de la mémoire de trame globale. Parallélement, le nombre de zones à transférer des Renderers vers la mémoire de trame augmente, ce qui, à puissance égale, diminue la fréquence d'animation.

Augmentation de la puissance du système

Pour augmenter la puissance du système, on peut soit augmenter la puissance du Renderer (augmentation de la fréquence d'horloge ou parallélisation des calculs), soit augmenter le nombre de Renderers. Notons que la première solution impose de redessiner tous les circuits

intégrés utilisés dans le Renderer. Cependant elle permet d'augmenter la puissance du système sans tenir compte des éventuels problèmes d'équilibrage de charge entre les zones écran. Dans la nouvelle version du système (appelée Pixel-Flow), la fréquence horloge est de 66 MHz et tous les calculs et transferts sont effectués sur 8 bits. Une telle technologie utilisée sur Pixel-Planes 5 permettrait au système d'atteindre des performances proches de 1,5 million de facettes/s avec un seul Renderer.

2.5 Parallélisme objet haut niveau

2.5.1 Rappel

Les architectures de ce type utilisent un parallélisme objet haut niveau (plusieurs unités de conversion travaillant en parallèle sur les objets de la base de données), et un parallélisme massif bas niveau (chaque unité de conversion est une machine massivement parallèle).

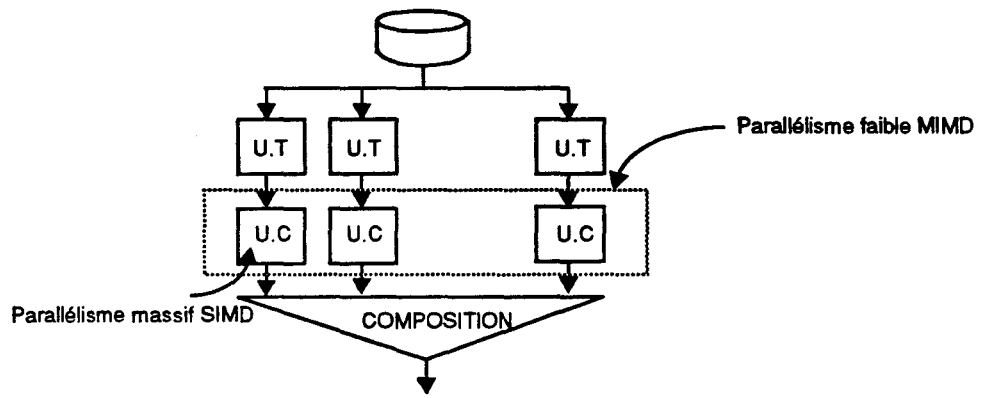


Figure 2.35 : parallélisme objet haut niveau

2.5.2 Parallélisme pixel bas niveau : PixelFlow

PixelFlow [MOLN91][MOLN92] est un projet mené à l'Université de Caroline du Nord pour définir une architecture à parallélisme objet de haut niveau. Ce système fait suite aux travaux de Steve Molnar sur les architectures à composition [MOLN88].

Plusieurs unités de conversion, appelées Renderer, génèrent chacune une partie des objets de la base de données. Un réseau de composition, implementé sous la forme d'un pipeline, effectue la composition (Z-buffer uniquement) des images générées par les différents Renderer. Par ailleurs, les calculs d'éclairement (formule de Phong) et de texture sont réalisés en post-traitement par plusieurs processeurs d'éclairement appelés Shaders (Figure 2.36).

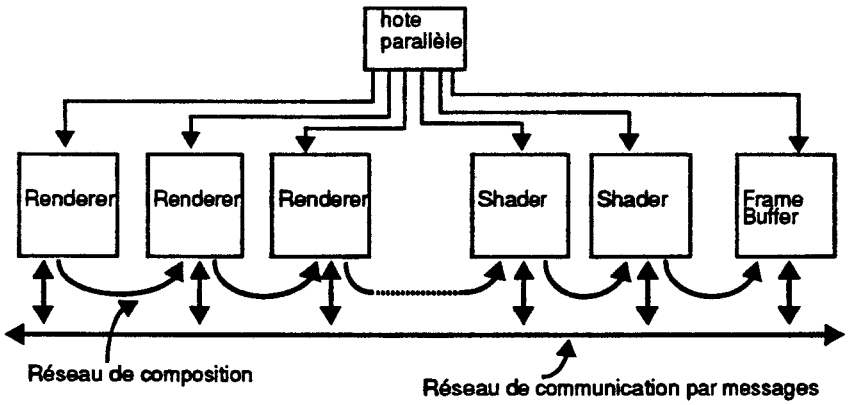


Figure 2.36 : schéma général de PixelFlow

Les performances d'une telle architecture sont déterminées d'une part par le débit du réseau de composition, et d'autre part par les performances des Renderers. Plusieurs possibilités ont été envisagées pour le Renderer de PixelFlow (dont notamment une solution à base de processeurs i860). La solution finalement retenue reprend le principe du Renderer de Pixel-Planes 5, ce qui permet de réutiliser les investissements et connaissances acquises sur ce projet, tant du point de vue matériel (développement des ASICs) que logiciel (développement des algorithmes de rendu). Cependant, les évaluateurs quadratiques ont été abandonné au profit d'évaluateurs linéaires (comme sur Pixel-Planes 4).

L'unité de conversion de PixelFlow est donc une machine massivement parallèle pixel utilisant 128×128 processeurs pixel. Elle est réalisée à l'aide de 64 circuits intégrant chacun 256 processeurs.

Les principales améliorations apportées par rapport au Renderer de Pixel-Planes 5 sont les suivantes :

- toutes les opérations (arbre de diffusion des expressions linéaires et ALU) sont effectuées sur des chemins de données 8 bits.
- la fréquence horloge est de 66 MHz.
- un multiplieur (partiel) est incorporé dans les processeurs pixel.
- chaque processeur pixel dispose en interne de 2048 bits de mémoire.
- un processeur i860 est directement associé à l'unité de conversion.

Le Renderer utilise toujours une réalisation partielle, mais de type séquentielle et non plus parallèle comme dans Pixel-Planes 5. Il reçoit tous les objets dont il a la charge, les classe suivant le découpage écran (zones 128×128), puis traite séquentiellement toutes les zones. La mémoire interne disponible dans les circuits du Renderer lui permet de mémoriser plusieurs zones. Lorsque tous les Renderers ont traité une zone déterminée (chacun ayant converti une partie des objets), leurs buffers sont composés via le pipeline de composition. Cette opération est la seule nécessitant une synchronisation entre les différents Renderers. Celle-ci est assurée par un réseau de communication à jetons (qui n'est autre qu'une version réduite du Ring de Pixel-Planes 5). Le réseau de composition est quant à lui implementé d'une part sur un fond de panier (256 bits à 132 MHz, soit un débit d'environ 30 Gbits/s), et d'autre part sur chaque circuit du Renderer sous la forme d'un opérateur de composition (i.e un opérateur de Z-buffer) 4 bits (les 64 circuits forment ainsi un opérateur 256 bits).

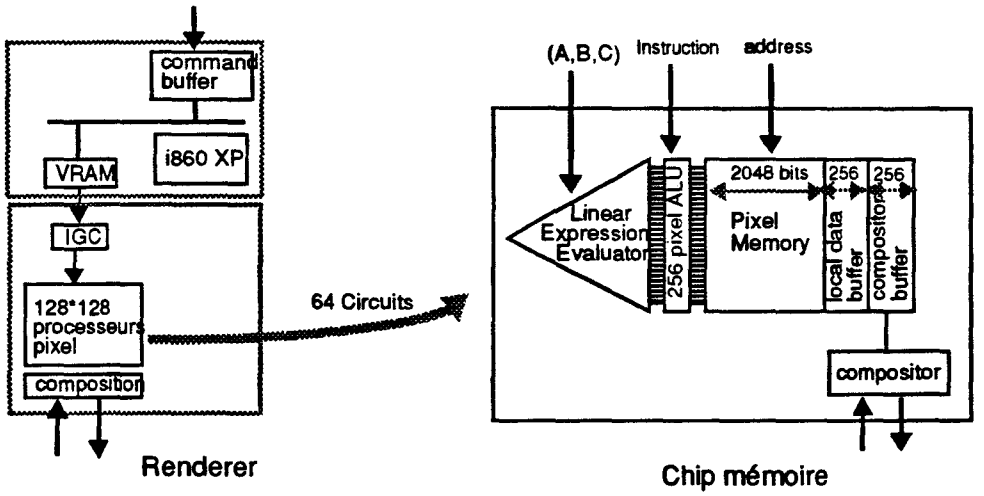


Figure 2.37 : le Renderer de PixelFlow

Lorsqu'une zone est entièrement traitée (i.e en sortie du réseau de composition), les calculs d'éclairement sont effectués pour tous les pixels de la zone par les processeurs d'éclairement (chaque pixel étant défini par sa couleur de base, son vecteur normal et éventuellement des informations de texture).

Le Shader est une machine massivement parallèle 128 x 128 basée sur les mêmes circuits que le Renderer (la principale différence entre la carte Renderer et la carte Shader réside dans l'ajout d'une mémoire de texture connectée à la mémoire des processeurs pixel). Les calculs d'éclairement et de texture sont ainsi effectués simultanément sur les 128*128 pixels de la zone en cours de traitement, les algorithmes utilisés étant les mêmes que ceux développés pour Pixel-Planes 5 [ELLS91]. Un Shader est capable de traiter 10000 zones par seconde (éclairement de Phong). Par ailleurs, plusieurs Shaders peuvent être utilisés en parallèle suivant la puissance désirée. Notons que le nombre de Shaders nécessaire dépend uniquement de l'algorithme d'éclairement utilisé, et non du nombre d'objets dans la scène.

Les performances du système dépendent principalement du nombre de Renderers et du nombre de Shaders. Un unique Renderer est capable de traiter environ 1,4 millions de triangles par seconde en utilisant la méthode de Gouraud, et 0.8 millions de triangles par seconde en utilisant la méthode de Phong et les textures. Les performances d'un système composé de plusieurs Renderers augmentent théoriquement linéairement avec le nombre de Renderers. En pratique, deux facteurs peuvent limiter cette augmentation de performances :

- un Renderer utilisant une réalisation partielle séquentielle, ses performances dépendent du facteur de duplication des primitives (voir le paragraphe sur Pixel-Planes 5).
- Un problème d'équilibrage de charge entre les différents Renderers peut apparaître du fait de la synchronisation nécessaire lors de la composition d'une zone. En effet, si tous les Renderers traitent globalement le même nombre d'objets, il se peut que sur une zone donnée, certains aient à traiter beaucoup plus d'objets que d'autres, obligeant ainsi ces derniers à attendre. Ce problème est en partie résolu par la possibilité de mémoriser plusieurs zones dans chaque Renderer.

Il est intéressant de noter que les deux problèmes que nous venons de soulever sont une conséquence directe de l'utilisation d'une réalisation partielle dans un système massivement parallèle.

2.6 Conclusion

Nous venons de présenter dans ce chapitre les principaux systèmes graphiques massivement parallèles proposés ces dix dernières années. La Figure 2.38 présente un tableau de classification des ces architectures.

parallélisme bas niveau (SIMD) / parallélisme haut niveau (MIMD)	Objet	Pixel
Total	Cohen Weinberg PROOF IMOGENE	PP4 RC
Partiel	GSP-NVS	SAGE PP5 1 Renderer PixelFlow 1 Renderer
Objet	-	Pixel-Flow
Pixel	-	PP5

Figure 2.38 : classement des architectures

On constate en consultant ce tableau que les unités de conversion objet n'ont jamais été utilisées dans des architectures multi-niveaux. De telles architectures hybrides sont pourtant envisageables, et il serait par exemple possible de remplacer les Renderers de Pixel-Planes 5 par des unités de conversion objet [KARP92]. On est en droit de se demander pourquoi le parallélisme massif objet, malgré les nombreuses études dont il a fait l'objet, n'a jamais réellement percé dans le domaine des systèmes graphiques hautes performances. Il nous semble que la réponse à cette question passe d'abord par une étude détaillée des unités de conversion objet et pixel. Cette étude fait l'objet du chapitre suivant.

Chapitre 3

Unités de conversion massivement parallèles

3.1 Présentation de l'étude

Nous avons montré dans le chapitre précédant que le parallélisme massif total n'est pas la solution du problème de la synthèse d'images interactives. La seule solution acceptable est d'utiliser un parallélisme massif partiel (réallocation dynamique des processeurs). Par essence, cette solution impose de découper l'image en zones non recouvrantes. Les architectures diffèrent alors par la forme et la taille des zones écran. On distingue principalement deux types de découpage :

- découpage par lignes (GSP-NVS en parallélisme objet, SAGE en parallélisme pixel)
- découpage par zones rectangulaires (Pixel-Planes 5 et PixelFlow en parallélisme pixel)

Dans ce chapitre, nous considérons un système à réalisation partielle séquentielle composé d'une unité de conversion massivement parallèle (objet ou pixel). Le but de chapitre est d'évaluer et de comparer les performances d'un tel système dans les deux cas suivants :

- parallélisme bas niveau objet (l'unité de conversion est une machine massivement parallèle objet).
- parallélisme bas niveau pixel (l'unité de conversion est une machine massivement parallèle pixel).

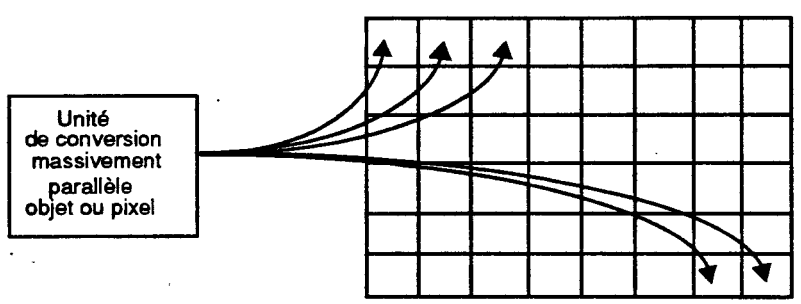


Figure 3.1 : le système étudié

L'étude que nous allons mener dans ce chapitre se déroulera en trois phases :

- Etude des performances intrinsèques de l'unité de conversion, c'est-à-dire du temps mis pour générer N objets contenus dans une zone de taille $L \times H$.

- Etude de l'accélération intrinsèque de l'unité de conversion (comparaison avec un générateur d'objets non massivement parallèle).
- Etude des performances globales du système à réalisation partielle séquentielle, c'est- à-dire du temps mis pour générer sur tout l'écran l'ensemble des objets présents dans la scène à visualiser.

3.2 Caractéristiques des unités de conversion

3.2.1 Unité de conversion objet

3.2.1.1 Introduction

L'unité de conversion objet que nous considérons est un pipeline de processeurs objets (voir chapitre 2) connecté via un opérateur de Zbuffer à une unité mémoire représentant une zone écran. Ce pipeline permet de traiter tous les objets appartenant à la zone en question. L'élimination des parties cachées est effectuée

- dans le pipeline pour tous les objets traités au cours de la même passe;
- par l'opérateur de Zbuffer pour les objets traités au cours de passes différentes.

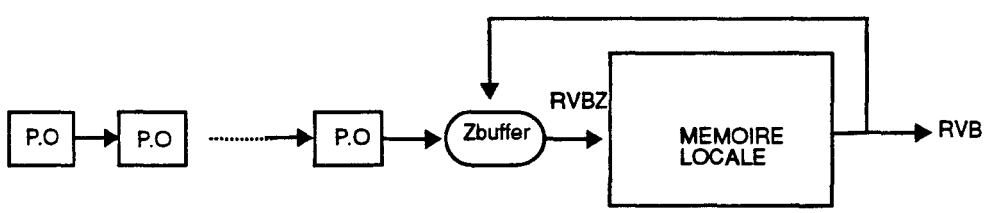


Figure 3.2 : pipeline objet et buffer de zone

La mémoire locale représente une zone rectangulaire de l'écran. La taille de cette zone dépend du nombre de boîtiers mémoires utilisés pour l'implémentation matérielle. Par exemple, si l'on choisit de mémoriser pour chaque pixel une profondeur (24 bits) et une couleur RVB (24 bits), une zone de 16K pixel correspond à une capacité mémoire de 96 Ko.

Par contre, la forme de la mémoire locale (nombre de lignes et nombre de colonnes) peut être choisie librement. Par exemple, une mémoire de 16K pixel peut représenter une région 28×128, une région 512×32, une région 1280×12 (i.e. 12 lignes écran), etc... Cette grande flexibilité dans la configuration de la mémoire est rendue possible par le fait qu'un pipeline objet génére les pixels dans l'ordre du balayage écran. Les unités de conversion orientées pixel ne possèdent pas cette caractéristique, et par conséquent doivent nécessairement implémenter des buffers de taille et de forme fixes (déterminées par le nombre et l'organisation des processeurs pixels). L'approche objet apporte donc une certaine flexibilité que l'on ne retrouve pas dans l'approche pixel.

Par ailleurs, une fois la taille de la zone mémoire fixée (par l'implémentation matérielle), il est possible, avec un pipeline objet, d'effectuer la conversion sur une zone virtuelle incluse dans la zone physique, et donc de taille plus réduite [KARP92]. Ainsi, un buffer de 16K pixel peut représenter quatre régions de 4K pixels (par exemple quatre zones 64*64, ou encore quatre zones 128*32, etc...).

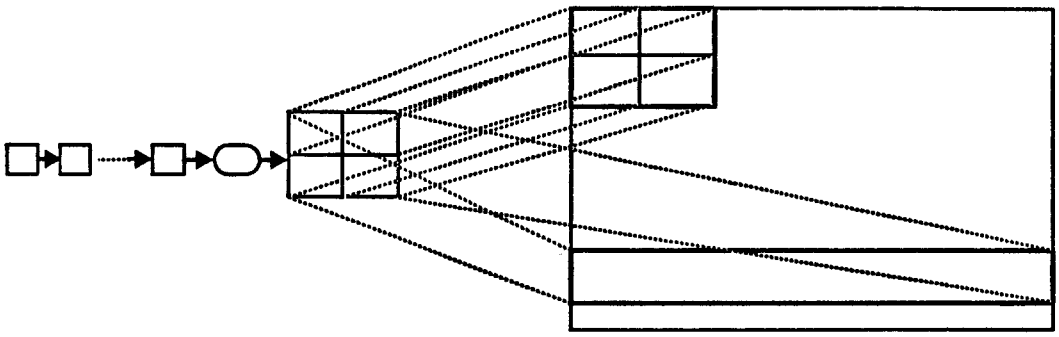


Figure 3.3 : possibilités de configuration de la mémoire locale

En définitive, un système composé d'un pipeline objet connecté à un buffer de zone via un opérateur de Zbuffer permet de construire une machine à découpage écran (parallélisme image) dont seule la taille maximale (en nombre de pixels) des zones est imposée par la réalisation matérielle. Le choix de la configuration (nombre de zones, taille et forme des zones) est entièrement programmable, soit statiquement (en début de session), soit dynamiquement (deux trames successives peuvent être configurées différement).

3.2.1.2 Valeurs caractéristiques

Nous venons de montrer que tout pipeline objet peut être configuré facilement (par programme) pour effectuer la conversion des objets sur une zone rectangulaire de taille et de forme quelconque. Dans la suite, le pipeline objet sera donc caractérisé par les valeurs suivantes :

- *No* : nombre de processeurs objet dans le pipeline.
- *L*, *H* : largeur et hauteur de la zone de conversion.
- *To* : temps de cycle. Le pipeline fournit une valeur tous les *To* secondes.
- *Co* : temps de chargement d'un objet dans le pipeline.
- *So* : temps de transfert des pixels de la mémoire locale vers la mémoire de trame globale.

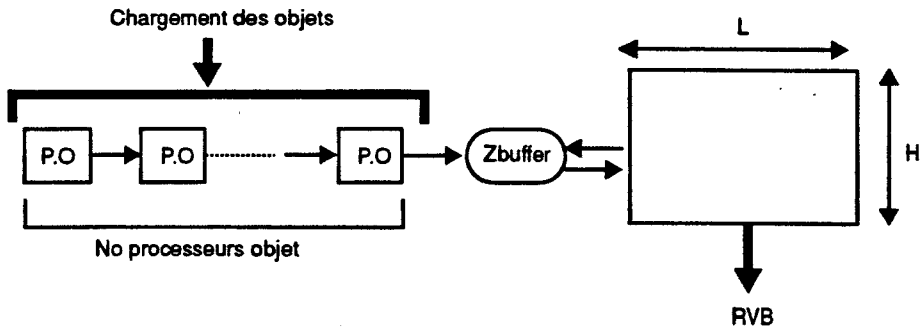


Figure 3.4 : modélisation du pipeline objet

No determine en grande partie la complexité matérielle du système. *To* dépend de la technologie utilisée dans la réalisation des circuits intégrés, ainsi que des composants mémoire utilisés. *L* et *H* n'ont aucune incidence sur la complexité du système, mais nous verrons par la suite qu'elles peuvent influer grandement sur ses performances.

No x *Co* représente le temps total nécessaire au chargement de la description des objets dans les *No* processeurs du pipeline. Deux solutions sont envisageables pour effectuer ce chargement :

- le chargement des objets utilise le même chemin de données que la conversion. Dans ce cas, chargement et conversion sont effectués alternativement.

Figure 3.5 : chargement/conversion séquentiels

- le chargement des objets utilise un chemin de données spécifique. Dans ce cas, chargement et conversion peuvent être effectués en parallèle.

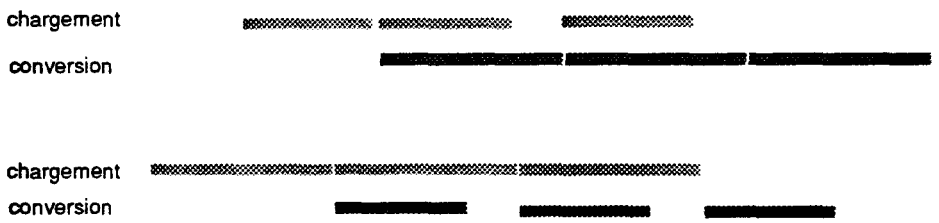


Figure 3.6 : chargement/conversion parallèles

So représente le temps de transfert des pixels de la mémoire locale (définis par leur seule valeur RVB) vers la mémoire de trame globale. Ce temps dépend de la nature de la liaison entre les deux systèmes de mémoire (il est possible de transférer plusieurs pixels à la fois), ainsi que du débit d'entrée de la mémoire de trame. A titre d'exemple, considérons le cas d'une mémoire de trame réalisée à l'aide de composants VRAM. En mode page (accès successif à toutes les lignes d'une même colonne), le temps d'accès en écriture est de l'ordre de 60 ns. Si deux composants sont utilisés en parallèle, le temps d'accès à la mémoire est alors de 30 ns, autorisant un débit d'entrée de 33 Mpixels/s. Pour une mémoire locale de taille 128×128, on a alors $So = 0.5ms$.

Notons ici qu'il est possible de ne pas utiliser de mémoire locale, mais d'effectuer directement la conversion sur la mèmoire de trame globale. Dans ce cas, le temps de transfert des pixels est nul. Cependant, cette solution possède deux inconvénients majeurs :

- l'élimination des parties cachées doit alors être effectuée sur la mémoire globale, qui, pour des raisons de coût, est réalisée à l'aide de composants de mémoire dynamique, plus lents que les mémoires statiques.

- il est nécessaire de disposer d'une mémoire de profondeur de la taille de l'écran, ce qui augmente le coût du système.

Pour les systèmes utilisant une mémoire locale, nous considérerons que celle-ci est dédoublée[1], permettant d'effectuer en parallèle la conversion d'une zone et le transfert des pixels de la zone précédente.

3.2.2 Unité de conversion pixel

L'équivalent pixel de l'unité objet décrite ci-dessus est une unité de conversion orientée pixel implémentant une grille de $L \times H$ processeurs pixels. Comme nous l'avons montré plus haut, les valeurs L et H sont fixées lors de la construction du système. Un tel système est caractérisé par :

- L, H : largeur et hauteur de la zone de conversion.
- Tp : temps de cycle, i.e. temps de génération d'un objet[2] (en général une facette).
- Sp : temps de transfert des pixels de la zone vers la mémoire de trame globale.

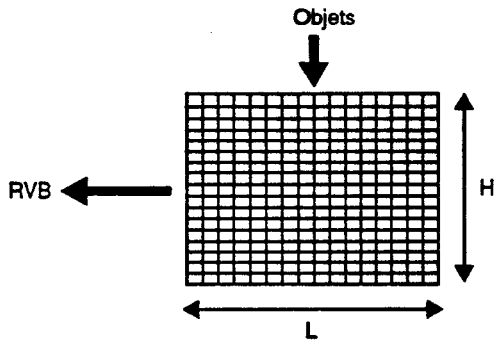


Figure 3.7 : modélisation d'une unité de conversion pixel

L et H déterminent le nombre de processeurs pixel, et donc reflètent la complexité matérielle de l'unité de conversion. Tp dépend de l'implémentation matérielle de l'algorithme de génération (de facettes généralement) choisi, et donc également de la complexité du système. Par exemple, Pixel-Planes 4 fonctionne en mode bit-série 1 bit à 10 Mhz, et génère une expression linéaire à la fois. En considérant qu'il faut 300 bits pour définir une facette, son temps de cycle est donc de 30 µs. Le Renderer de Pixel-Flow, quant à lui, fonctionne en mode bit-série 8 bits à 66 MHz, mais génère toujours une expression à la fois. Son temps de cycle est donc de 0,5 µs. Le système SAGE utilise un découpage par ligne (génération de segments horizontaux), et génère un segment à chaque cycle d'horloge. Son temps de cycle est donc égal au temps de cycle de base, à savoir 40 ns.

Une des caractéristiques essentielles des unités de conversion pixel est que Tp ne dépend pas de L et H dans le cas où le mécanisme de conversion (arbre ou multi-pipeline) est pipeliné. Nous analyserons par la suite l'influence de cette caractéristique sur la complexité matérielle et les performances globales du système.

Sp représente le temps de transfert des valeurs RVB contenues dans les processeurs pixel du

1. double-buffered en anglais
2. Ce temps est en général différent du temps de cycle de base de la machine.

réseau vers la mémoire de trame globale.

Notons que, une fois L et H choisis, les performances de l'unité de conversion ne dépendent que de Tp.

3.2.3 Unité de conversion séquentielle

Dans la suite de ce chapitre, nous tenterons de comparer les performances des unités de conversion objet et pixel. Par ailleurs, afin d'estimer le gain apporté par le parallélisme SIMD (qu'il soit objet ou pixel), nous utiliserons comme référence une unité de conversion constituée d'un unique processeur[1] ne générant pour chaque facette que les pixels utiles à raison d'un pixel par cycle. Une telle unité est caractérisée uniquement par son temps de cycle Ts.

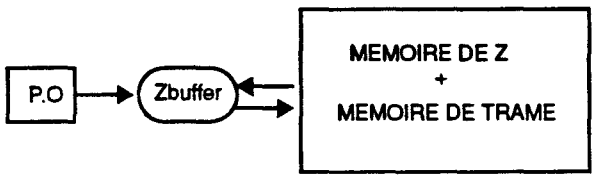


Figure 3.8 : modélisation d'une unité de conversion séquentielle

3.3 Caractérisation de scènes

3.3.1 Définitions des paramètres caractéristiques

Nous allons ici préciser certains paramètres permettant de décrire la complexité de la scène à représenter (des études semblables sont menées dans [GHAR89] et [GROS91]). Ces paramètres nous serviront par la suite pour analyser l'efficacité et le rendement du parallélisme massif objet.

- L : largeur de la zone de conversion
- H : hauteur de la zone de conversion
- N : nombre d'objets[2] dans la zone ce conversion
- S : surface moyenne des objets (en nombre de pixels)
- P : profondeur moyenne de la scène (la profondeur d'un pixel correspond au nombre d'objets présent en ce pixel, cachés ou visibles)

Ces paramètres sont caractéristiques de la scène, mais ne sont pas indépendants. En effet, ils sont liés par la relation suivante :

$$L \times H \times P = N \times S \qquad (1)$$

1. tel que le chip PRC [SWAN86].
2. Nous considérons comme objets les primitives de base (cablées) de la machine. En général, ce sont des triangles, mais il est possible d'envisager des primitives cablées de plus haut niveau (par exemple les quadriques).

Cette relation peut être utilisée pour calculer la profondeur moyenne de la scène, et s'exprime alors par :

$$P = \frac{N \times S}{L \times H}$$

Elle signifie alors que la profondeur moyenne de la scène est égale au nombre de pixels générés lors de la conversion divisé par le nombre de pixels de la zone.

3.3.2 Modification des paramètres

Nous allons présenter ici les différentes possibilités de variation des paramètres, et leur influence sur le processus de transformation géométrique et le processus de conversion. Nous supposerons que les valeurs L et H sont constantes, et considérerons principalement l'affichage d'une scène facettisée.

Soit une scène caractérisée par ses paramètres N, S, et P. Les situations suivantes entraînent une modification de ces paramètres.

- augmentation de la précision de la facettisation. Dans ce cas, N augmente, mais S diminue, et P reste constant.
- ajout d'objets à même niveau de facettisation. Dans ce cas, N augmente, S reste constant et donc P augmente.
- changement de taille des objets. Dans ce cas, S augmente (ou diminue), N est constant et P augmente (ou diminue).
- passage à des primitives de plus haut niveau. Dans ce cas, N diminue, S augmente et P reste constant.

Il est bien évident que ces différentes situations peuvent se combiner (par exemple ajout d'objets plus précisément facettisés).

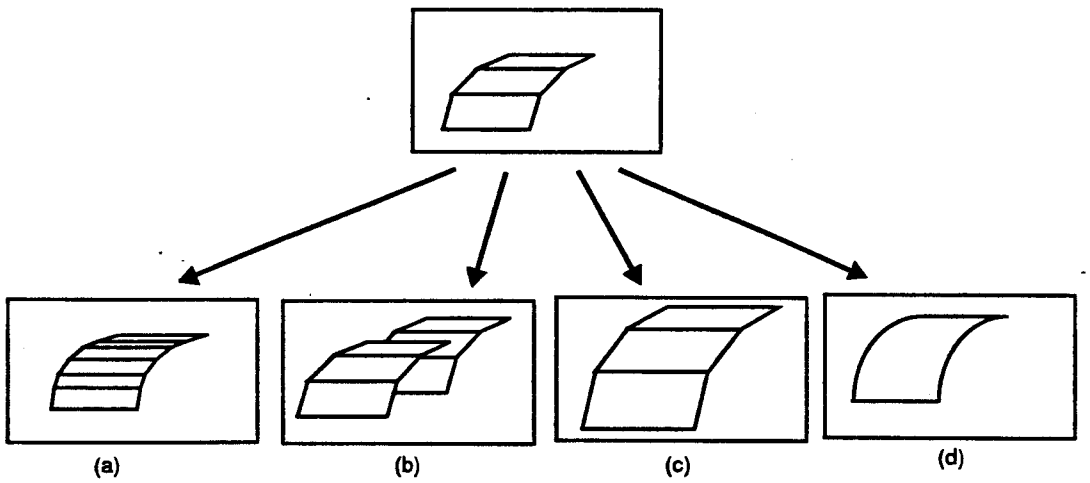


Figure 3.9 : exemples de modification des paramètres

Remarque

Les cas (b) (la taille des objets augmente quand l'observateur se rapproche d'eux) et (c) (de nouveaux objets apparaissent dans la scène) correspondent à une animation courante. Les cas (a) et (d) sont différents, car ils ne dépendent pas de l'animation.

Le cas (a) correspond à une augmentation de la précision de la facettisation. La qualité de l'image est alors meilleure, mais au prix d'une augmentation du nombre d'objets traités par le hôte. A partir d'un niveau trop fin de facettisation, celui-ci ne sera plus capable d'effectuer les transformations géomètriques en temps réel.

Le cas (d) suppose que l'on dispose d'une machine capable d'afficher directement des primitives de haut niveau. Le nombre d'objets à traiter sur le hôte est alors considérablement réduit.

3.4 Implémentation matérielle des unités de conversion

Nous présentons ici rapidement les implémentations matérielles possibles des unités de conversion objet et pixel. Une description détaillée de la réalisation VLSI d'un processeur objet et d'un processeur pixel sera fournie dans le chapitre 4.

3.4.1 Choix de la méthode de génération des facettes

Il existe deux grandes familles de méthodes d'affichage de facettes triangulaires : les méthodes par équation et les méthodes par suivi de contour.

Dans les méthodes par équation, le contour d'une facette est définie par les équations des droites la délimitant. Ces équations sont du type $ax+by+c=0$ (la méthode de calcul incrémental d'une telle équation est présentée dans le chapitre suivant). L'intérieur d'une facette triangulaire est alors déterminé par le système d'inéquations suivant :

$$a1x+b1y+c1 \geq 0$$
$$a2x+b2y+c2 \geq 0$$
$$a3x+b3y+c3 \geq 0$$

Ces équations sont définies dans le repère absolu de l'écran (l'origine du repère est par exemple le point supérieur gauche de l'écran). Si on utilise un découpage en zones rectangulaires, il est nécessaire d'effectuer dans chaque zone où la facette est présente un changement de repère afin d'utiliser le point supérieur gauche (de coordonnées (x_0, y_0)) de la zone comme nouvelle origine. Une équation $ax+by+c$ est alors transformée en $a(x+x_0)+b(y+y_0)+c$, qui peut encore se réécrire sous la forme $ax+by+(ax_0+by_0+c)$. Dans la zone, la facette est donc toujours définie par trois inéquations:

$$a1x+b1y+a1x_0+b1y_0+c1 \geq 0$$
$$a2x+b2y+a2x_0+b2y_0+c2 \geq 0$$
$$a3x+b3y+a3x_0+b3y_0+c3 \geq 0$$

Si les coordonnées (x_0, y_0) sont des multiples de 2, les multiplications sont de simples décalages. En conclusion, l'utilisation d'un découpage en zones rectangulaires n'augmente que peu le travail du hôte (deux décalages et deux additions par équation).

Dans les méthodes à suivi de contour, il faut tout d'abord déterminer un des sommets du triangle (en général celui qui possède la plus petite ordonnée). Un suivi incrémental de contour permet alors de déterminer, pour chaque ligne, les coordonnées gauches et droites du segment intérieur à la facette (Le système SAGE utilise une telle méthode).

Cette méthode est parfaitement adaptée à un découpage en ligne. Cependant, si on utilise une découpage en zone, il est absolument nécessaire de déterminer le polygone résultant du fenêtrage de la facette par la zone rectangulaire (opération très coûteuse car elle nécessite de nombreuses divisions). Par ailleurs, le nombre de cotés de ce polygone varie de trois à sept suivant la position de la facette initiale. Dans le plupart des cas, il est donc indispensable de le redécomposer en triangles.

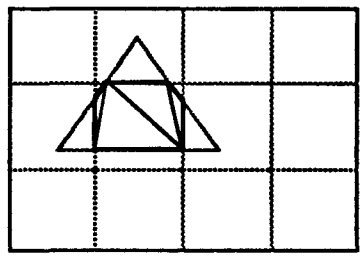


Figure 3.10 : découpage d'une facette dans une zone

En conclusion:

Dans le cas d'un découpage en lignes, on peut utiliser indifféremment la méthode par équations ou la méthode de suivi de contour.

Dans le cas d'un découpage en zones rectangulaires, il est préférable d'utiliser la méthode par équations.

Dans la suite de cet exposé, nous supposons que les unités de conversion étudiées utilisent la méthode par équations.

3.4.2 Unité de conversion objet

L'unité de conversion objet est composée principalement de sept unités calculant chacune une expression bilinéaire du type $ax+by+c$. Les trois premières unités déterminent le contour du triangle, la quatrième détermine la profondeur et les trois dernières les trois composantes RVB de la couleur (dans le cas d'une interpolation de Gouraud).

3.4.3 Unité de conversion pixel

Dans le cas d'un découpage en zones rectangulaires, on distingue deux types de réseaux pixels utilisant la méthode par équations : les réseaux multi-pipeline et les réseaux à arbre de diffusion.

3.4.3.1 Multi-pipeline

La Figure 3.11 présente un réseau multi-pipeline.

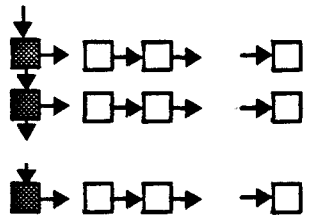


Figure 3.11 : multi-pipeline

Un multi-pipeline utilise deux types de cellule : les cellules du pipeline horizontal effectuent des interpolations linéaires en x, alors que celles du pipeline vertical effectuent des interpolations linéaires en y. Si les sept expressions à évaluer transitent en parallèle dans le réseau, on peut alors considérer que la cellule d'un réseau pixel est environ deux fois moins complexe que le processeur d'un pipeline objet.

3.4.3.2 Diffusion

Il est également possible d'utiliser un arbre de diffusion pour évaluer en parallèle une expression $ax+by+c$ sur tous les processeurs pixels (c.f. Pixel-Planes 4 et Pixel-Flow). La complexité matérielle (nombre de transistors) d'un tel arbre est identique à celle d'un multi-pipeline. Cependant, la surface silicium occupée est beaucoup plus importante, car l'arbre nécessite plus de communications que le pipeline (un pipeline de n opérateurs nécessite $n-1$ liaisons alors qu'un arbre en nécessite $2(n-1)$. Par ailleurs, l'arbre est une structure moins régulière que le pipeline, et se prête donc moins facilement à une intégration VLSI.

Nous pensons donc qu'il est préférable d'utiliser une structure multi-pipeline plutôt qu'une structure en arbre de diffusion.

3.5 Performances intrinsèques des unités de conversion

Nous étudions dans cette partie les performances intrinsèques des unités de conversion objet et pixel. Pour cela, nous considérons une zone écran définie par ses paramètres L, H, N, S et P, et calculons le temps de génération des objets présents dans la zone.

Pour les unités de conversion objet, nous considérerons toujours un découpage en zones rectangulaires non dégénérées (généralement carrées). Le cas du découpage par ligne (type GSP-NVS) est différent, et sera étudié au paragraphe 3.8.2.

3.5.1 Unité de conversion objet

Pour calculer les performances de l'unité de conversion objet, nous nous placerons dans le cas où chargement et conversion sont effectués en parallèle (i.e sur des chemins de données séparés). Deux cas de figure peuvent alors se présenter suivant que le temps de chargement des objets est inférieur ou supérieur au temps de conversion. Le temps de conversion des No objets est $L \times H \times To$. Le temps de chargement des No objets est $No \times Co$. Le temps de chargement est toujours inférieur au temps de conversion si

$$No \times Co \leq L \times H \times To, \text{ soit } No \leq \frac{L \times H \times To}{Co}$$

Dans la suite, nous appellerons No_{max} le seuil que nous venons de calculer. Nous étudions

dans la suite le comportement du pipeline dans les deux cas suivants :

- $No < No_{max}$
- $No \geq No_{max}$

A titre d'exemple (que nous reprendrons dans toute la suite de ce chapitre), considérons un pipeline fonctionnant à 33 MHz alimenté par un processeur i860XP (bus 64 bits pouvant fournir une valeur toute les 50 ns), et utilisant un découpage en zones 32×32. Une facette est codée à l'aide de sept expressions linéaires $Ax + By + C$. Nous supposerons que les trois coefficients A, B, C peuvent être codés sur un unique mot de 64 bits (format du bus d'un processeur 64 bits de type i860). Dans ce cas, Co ne dépend que du débit du bus. Si les processeurs sont connectés directement sur le bus, il est possible de charger une valeur toutes les 50 ns (vitesse maximale du bus du i860XP), soit un objet toutes les 350 ns (on a alors $Co = 350ns$).On a donc

$$L = 32,\ H = 32,\ To = 30ns,\ Co = 350ns.$$

On a alors $No_{max} = 88$

3.5.1.1 $No < No_{max}$

Dans ce cas, le temps de traitement des objets dépend uniquement du nombre de passes de conversion effectuées par le pipeline. Deux cas peuvent se présenter :

- si N est un multiple de No, alors le nombre de passes à effectuer est

$$NdivNo = \frac{N}{No}$$

Le temps de génération des N objets est alors

$$t_o = \frac{N}{No} \times L \times H \times To$$

Notons que chaque passe utilise l'ensemble des processeurs pipeline.

- si N n'est pas un multiple de No, le nombre de passes à effectuer dans le pipeline est

$$(NdivNo) + 1 \ ^{1}$$

Le temps de génération des N objets est donc

$$t_o = ((NdivNo) + 1) \times L \times H \times To$$

Il est important de noter que, lors de la dernière passe, tous les processeurs ne sont pas utilisés. Le nombre de processeurs effectivement actifs lors de cette passe est $NmodNo$ [2]. Par exemple, pour traiter 250 objets dans un pipeline de 100 processeurs, il faut effectuer deux passes avec tous les processeurs actifs, puis une passe avec

1. a div b correspond au quotient de a par b
2. a mod b correspond au reste

seulement 50 processeurs actifs.

Les performances intrinsèques de l'unité de conversion objet peuvent donc être représentées par la courbe suivante :

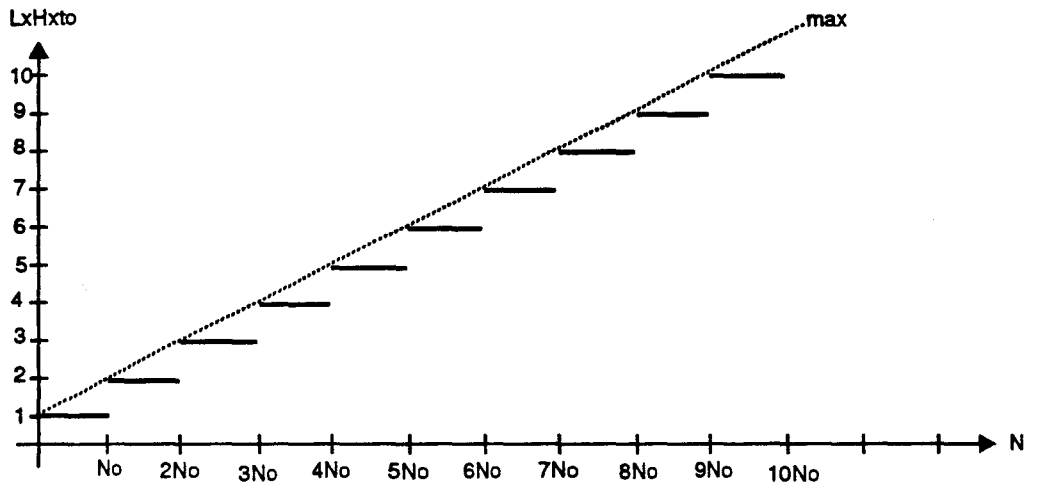


Figure 3.12 : courbe t_o en fonction de N

Nous pouvons constater que t_o n'est pas une fonction linéaire. De ce fait, il est a priori difficile de définir les performances de l'unité de conversion objet en respectant l'unité de mesure classique, à savoir le nombre d'objets générés par seconde. Cette unité ne peut être utilisée que pour définir les performances maximales de l'unité de conversion, qui sont atteintes quand N est un multiple de No. Dans ce cas, l'unité de conversion est capable de générer

$$\frac{No}{L \times H \times To} \text{ objets/s (si } To \text{ est exprimé en seconde).}$$

En particulier, si $No = No_{max}$, la puissance maximale est alors de

$$\frac{1}{Co} \text{ objets/s}$$

La puissance maximale de l'unité de conversion est donc égale à la fréquence de chargement des objets dans le pipeline. Cette valeur est indépendante de la taille des zones. Bien evidemment, le nombre de processeurs à utiliser pour atteindre cette puissance dépend quant à lui de la taille des zones.

A titre d'exemple, notre pipeline de référence (88 processeurs, 33 MHz, zones 32×32) développe une puissance maximale théorique de 2,86 Mfacettes/s. A la même fréquence, un pipeline de 352 processeurs utilisant des zones 64×64 aurait la même puissance maximale.

3.5.1.2 $No \geq No_{max}$

Dans ce cas, le temps de chargement des No objets est plus long que leur temps de conversion.

La performance maximale est atteinte quand les processeurs sont toujours occupés. Dans ce cas, puisque le temps de chargement est supérieur au temps de conversion, le temps de

traitement des N objets est $N \times Co$. La puissance maximale du pipeline est donc

$$\frac{1}{Co} \text{ objets/s}$$

3.5.1.3 Conclusion

La conclusion partielle que l'on peut tirer de cette étude est la suivante :

Dès que le nombre de processeurs objets est plus grand que la valeur limite, la performance maximale de l'unité de conversion objet est égale à la fréquence de chargement des objets dans le pipeline.

Si l'on considère notre exemple de base, cela signifie qu'à partir de 88 processeurs, la puissance maximale de l'unité de conversion est constante et égale à 2,86 Mfacettes/s. Cette conclusion peut paraître surprenante, car il semble a priori évident que plus le nombre de processeurs est élevé, plus le système devrait être puissant. En fait, nous verrons plus tard que le nombre de processeurs ne joue pas sur la puissance du système (en terme de nombre d'objets générés par seconde), mais uniquement sur sa fréquence d'animation (nombre d'images générées par seconde).

Par ailleurs, une autre question se pose : est-il préférable d'utiliser 88 processeurs sur des zones 32×32, ou 352 processeurs sur des zones 64×64 ? Nous répondrons à cette question dans la suite de ce chapitre.

3.5.2 Unité de conversion pixel

Les performances d'une unité de conversion pixel (par exemple le Renderer de Pixel-Planes 5 ou PixelFlow) sont (beaucoup) plus simples à calculer, car il n'y a pas de phase de chargement des objets. Par ailleurs, les objets étant traités séquentiellement, le temps de génération croît linéairement avec le nombre d'objets dans la zone. Le temps de génération des N objets est

$$t_p = N \times Tp$$

La courbe des performances est donc la suivante :

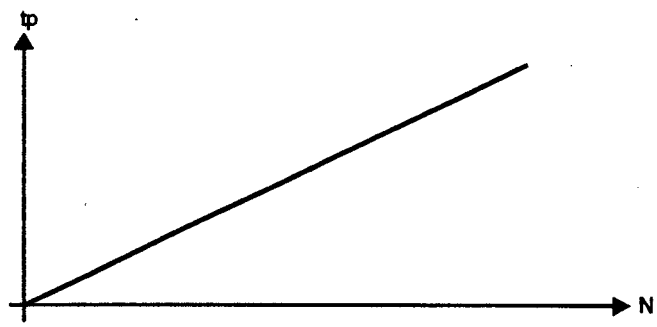


Figure 3.13 : courbe t_s en fonction de N

Dans le cas de l'unité de conversion pixel, il est possible de calculer effectivement les performances en termes de nombre d'objets par seconde. L'unité est capable de génerer

$\frac{1}{Tp}$ objets/s (si Tp est exprimé en seconde).

Notons que contrairement au cas de l'unité de conversion objet, ce chiffre représente la puissance effective (et non pas maximale) de l'unité de conversion pixel.

De la même façon que nous l'avons fait pour le pipeline objet, nous allons définir pour l'unité de conversion pixel un exemple de référence que nous utiliserons tout au long de ce chapitre. Pour cela, nous considérons le réseau multi-pipeline équivalent (en complexité) à notre pipeline objet de référence, et cadencé à la même fréquence (33 MHz). La complexité d'un processeur objet étant le double de celle d'un processeur pixel, le pipeline de 88 processeurs objets est environ équivalent en complexité à un réseau pixel 32×32 dans lequel les sept expressions transitent en série (une cellule contient donc un seul interpolateur et non sept).

Les caractéristiques d'un tel réseau sont donc les suivantes

$$L = 32,\ H = 32,\ Tp = 210ns$$

La puissance intrinsèque effective est alors d'environ 4,7 millions de facettes par seconde.

3.5.3 Unité de conversion séquentielle

Nous considérons le cas idéal d'unité de conversion séquentielle, à savoir celle générant un pixel par cycle (calcul en parallèle des valeurs RVB et Z). Par ailleurs, nous supposerons que les temps d'initialisation[1] des interpolateurs sont toujours inférieurs au temps de conversion d'une primitive[2]. De cette façon, la conversion d'une primitive peut débuter dès que celle de la précédente est terminée.

Ces hypothèses étant fixées, le temps de conversion d'une primitive ne dépend donc que de sa surface. Le temps de génération de N objets de surface moyenne S pixels est donc

$$t_s = N \times S \times Ts$$

L'unité de conversion séquentielle est donc capable de générer $\frac{1}{S \times Ts}$ objets/s.

A titre d'exemple, supposons qu'une telle unité utilise une mémoire suffisamment rapide pour effectuer un cycle lire/comparer/ecrire en 30 ns (33 MHz). Si l'on considère des facettes de 100 pixels, la puissance de l'unité séquentielle est alors d'environ 300.000 facettes par seconde.

3.5.4 Conclusions

A ce stade de l'étude, nous pouvons énoncer les conclusions suivantes :

- Les performances intrinsèques des unités de conversion massivement parallèles (objet ou pixel) dépendent uniquement des caractéristiques matérielles, jamais de la taille des primitives générées dans la zone de conversion.
- Les processeurs d'une unité de conversion objet peuvent être inactifs lors de la conversion.

---

1. setup
2. sur les systèmes réels, ceci est en général faux pour les petites facettes (moins de 50 pixels).

- Le chargement des objets est un problème crucial et délicat dans le cas d'une unité de conversion objet. Par ailleurs, la puissance de l'unité est limitée par le débit de chargement des objets.
- Pour un nombre donné de processeurs objet, la puissance de l'unité de conversion dépend de la taille des zones.
- La puissance de l'unité de conversion pixel ne dépend quant à elle que du facteur Tp, et donc du taux de parallélisme utilisé dans les processeurs pixels. Il faut cependant être conscient du fait qu'un réseau pixel doit utiliser au minimum 1024 processeurs pixels.

3.6 Accélération intrinsèque des unités de conversion

3.6.1 Unité de conversion objet

3.6.1.1 Calcul du facteur d'accélération

Nous allons ici comparer les performances d'une unité de conversion objet à celles d'un générateur monoprocesseur traitant les objets séquentiellement, mais ne générant pour chaque objet que les pixels utiles. Cette comparaison permettra de définir le taux de parallélisme effectif apporté par le parallélisme objet. Nous définissons l'accélération de l'unité de conversion objet par le rapport

$$A_o = \frac{t_s}{t_o}$$

Comme lors du calcul des performances intrinsèques de l'unité de conversion objet, il est nécessaire de distinguer deux cas :

- si N est un multiple de No, alors

$$A_o = \frac{N \times S \times Ts}{\frac{N}{No} \times L \times H \times To} = \frac{No}{L \times H} \times S \times \frac{Ts}{To}$$

- si N n'est pas un multiple de No, alors

$$A_o = \frac{N \times S \times Ts}{(1 + N \ div \ No) \times L \times H \times To}$$

Nous pouvons supposer que $To = Ts$[1]. Par ailleurs, $L \times H \times P = N \times S$. Les formules peuvent donc se réécrire sous la forme :

- si N est un multiple de No, alors

$$A_o = \frac{No \times S}{L \times H} = \frac{No}{N} \times P$$

- si N n'est pas un multiple de No, alors

$$A_o = \frac{N \times S}{(1 + N \ div \ No) \times L \times H} = \frac{P}{(1 + N \ div \ No)}$$

1. Implicitement, cela suppose que le générateur séquentiel est connecté à une mémoire très rapide...

Pour des valeurs données de *No*, *L*, *H* et *S*, la forme de la courbe obtenue est la suivante :

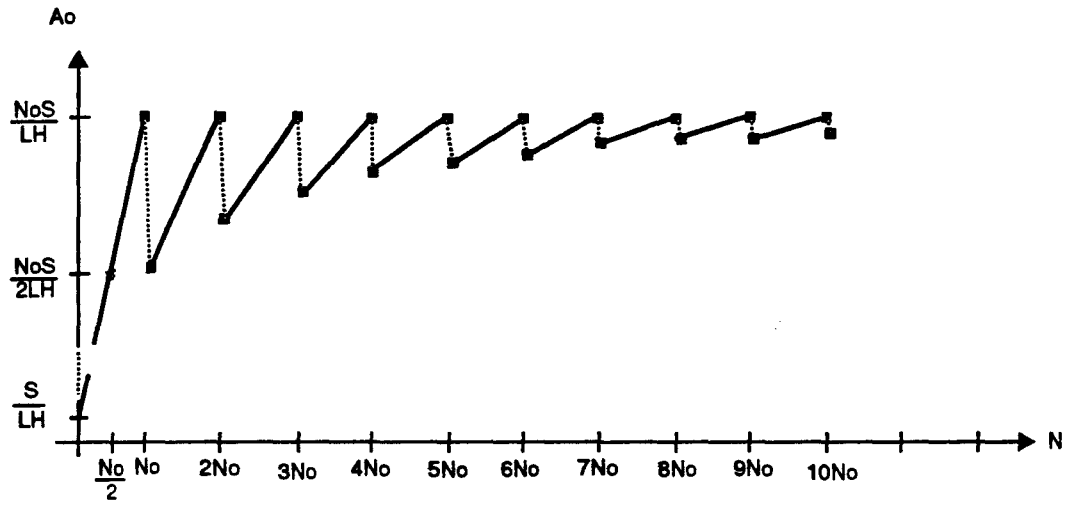


Figure 3.14 : courbe A_o en fonction de N

Nous allons par la suite donner des exemples numériques. Cependant, nous pouvons déjà tirer plusieurs enseignements de cette courbe :

- l'accélération du pipeline s'effondre si le nombre d'objets à générer dans la zone est inférieur à $(No)/2$. L'accélération minimale est obtenue quand $N = 1$. L'accélération est dans ce cas égale à $S/(L \times H)$, qui est généralement bien inférieur à 1.
- Dès que le nombre d'objets est supérieur à $(No)/2$, l'accélération est toujours comprise entre $(No \times S)/(2 \times L \times H)$ et $No \times S/(L \times H)$.
- le pipeline est d'autant mieux utilisé qu'il y a beaucoup d'objets dans la zone. En effet, plus il y a d'objets, plus la perte d'efficacité liée à la dernière passe devient négligeable.

3.6.1.2 exemple de référence

Nous calculons ici le facteur d'accélération obtenu par notre pipeline de référence (88 processeurs, 33 MHz, zones 32×32). La courbe obtenue lors de l'affichage de facettes de 100 pixels est la suivante

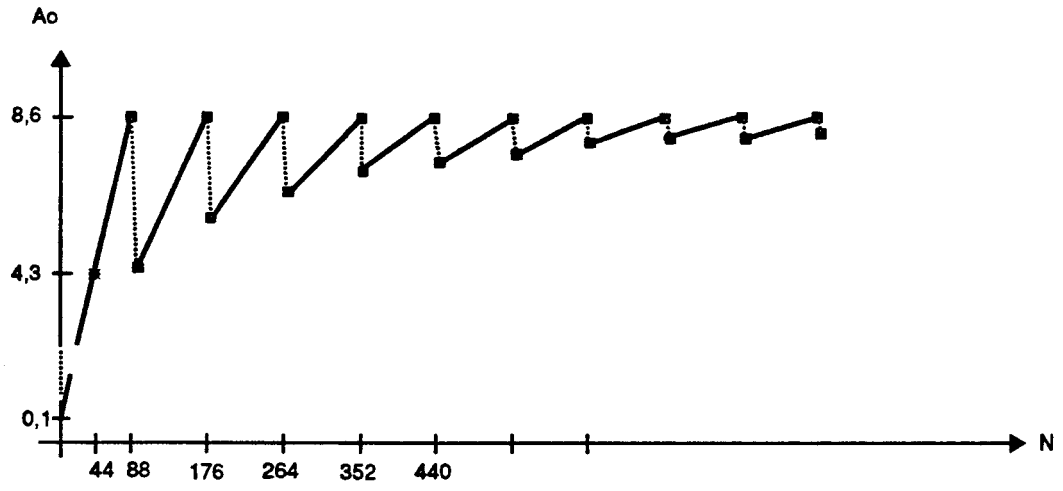


Figure 3.15 : accélération du pipeline de référence

Nous pouvons constater que dans le pire des cas (un objet dans la zone), le pipeline objet est dix fois moins rapide que le générateur séquentiel. Il devient plus rapide dès que le nombre d'objets dans la zone dépasse 10. Il apparaît donc que quand le nombre d'objets est inférieur à 10, il serait préférable de dégrader le fonctionnement du pipeline afin de n'utiliser qu'un processeur en mode séquentiel. Si l'on veut bénéficier d'un tel ajustement dynamique, il faudrait donc modifier la structure du processeur objet afin qu'il puisse générer un objet non plus dans tous les pixels de la zone, mais uniquement à l'intérieur du contour de l'objet (par exemple en utilisant l'algorithme de Pineda [PINE88]. Cette solution n'a pas été explorée dans cette thèse.

La perte d'accélération due à la dernière passe devient négligeable quand le nombre d'objets dans la zone est très élevé. L'accélération peut alors être considérée constante et égale à 8,6. Cependant, une scène comprenant en moyenne 88 objets par zone 32×32 (pour un écran 1024×1024) contient 90.000 objets au total, et peut donc déjà être considérée comme une scène complexe. La plupart des applications se situeront donc dans le début de la courbe.

3.6.2 Unité de conversion pixel

3.6.2.1 Calcul de l'accélération

L'accélération de l'unité de conversion pixel est définie par le rapport

$$A_p = \frac{t_s}{t_p}$$

Contrairement à l'unité de conversion objet, les performances de l'unité pixel sont linéaires. Le facteur d'accélération s'exprime donc par la formule suivante :

$$A_p = \frac{N \times S \times Ts}{N \times Tp} = S \times \frac{Ts}{Tp}$$

Une fois les caractéristiques matérielles de l'unité de conversion pixel choisies (i.e une fois Tp fixée), son facteur d'accélération ne dépend que de la surface moyenne des objets présents

dans la zone en cours de traitement.

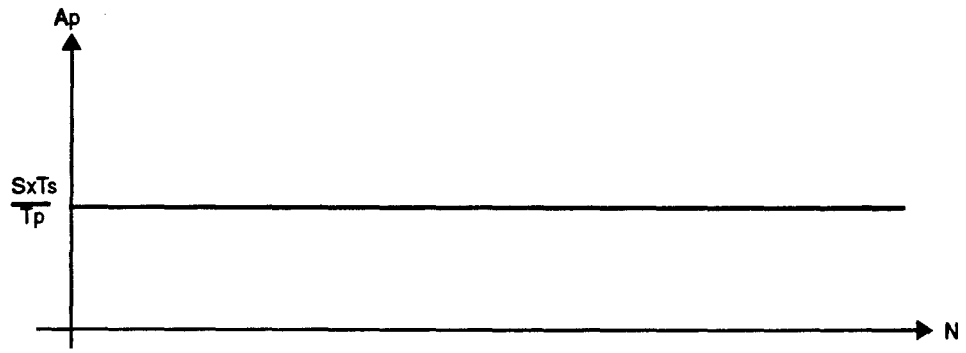


Figure 3.16 : courbe t_{ps} en fonction de N

On constate encore que les unités de conversion pixel sont beaucoup plus faciles à analyser que leurs homologues objet.

3.6.2.2 exemple de base

Nous reprenons ici notre exemple de base (réseau 32 x 32, 7 cycles par facette, 33 MHz). Son accélération est donc égale à $S/7$. Lors de l'affichage de facettes de 100 pixels, elle est donc égale à 14.

Si l'on considère notre deuxième exemple de référence (même réseau mais générant une facette par cycle), l'accélération obtenue est alors de 100.

3.7 Performances d'un système complet

Nous venons d'étudier les caractéristiques intrinsèques des unités de conversion objet et pixel. Il reste maintenant à étudier les performances effectives des systèmes à réalisation partielle séquentielle construit à partir de ces unités.

La puissance d'une machine graphique destinée à l'animation temps réel peut être divisée en trois facteurs :

- puissance de l'unité de traitement géométrique (exprimée en nombre de sommets par seconde).
- puissance de l'unité de conversion (exprimée en nombre de primitives par seconde).
- fréquence d'animation maximale (exprimée en Hz).

L'unité de traitement géométrique ne sera pas étudiée dans cette thèse. Il est cependant probable que celle ci soit une machine MIMD à base de processeurs généraux (le processeur couramment utilisé aujourd'hui est le i860XP)[1]. Dans toute la suite, nous supposerons que le puissance du système n'est pas limitée par l'unité de traitement géométrique.

1. c'est la solution retenue actuellement sur toutes les machines graphiques haut de gamme.

3.7.1 Facteur de duplication

Tous les chiffres que nous avons cités dans les paragraphes précédents représentent la puissance maximale d'une unité de conversion lors de la génération des objets appartenant à une zone donnée. Pour calculer les performances effectives d'un système construit à partir de cette unité de conversion, il est nécessaire de tenir compte du phénomème de duplication de primitives, conséquence de l'utilisation d'une unité de conversion massivement parallèle dans un système à parallélisme image haut niveau et réalisation partielle.

Le problème de la duplication de primitives a déjà été évoqué dans le chapitre 4 à propos de Pixel-Planes 5. Nous le rappelons brièvement ici. Considérons une unité de conversion massivement parallèle utilisé dans un système à parallélisme image haut niveau et réalisation partielle séquentielle. Dans ce cas, l'écran est divisé en plusieurs zones, et les objets sont classés en fonction des zones auxquelles ils appartiennent. L'unité de conversion génère alors tous les objets de la première zone, puis tous ceux de la deuxième zone, et ainsi de suite.

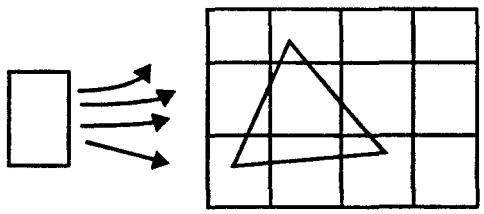


Figure 3.17 : duplication de primitives

Les performances du système (sur tout l'écran et non plus sur une seule zone) dépendent à la fois des performances intrinsèques de l'unité de conversion, et du facteur de duplication.

Le facteur de duplication dépend de la taille des primitives et de la taille des zones écran. Une étude est menée dans [MOLN91] afin de déterminer l'influence exacte de ces paramètres. Pour cela, un modèle analytique a été développé, puis confronté à des mesures effectuées sur des scènes réelles. Dans la plupart des cas, les valeurs prévues par le modèle sont très proches de celles effectivement mesurées. Nous utiliserons donc ce modèle pour estimer les performances des systèmes étudiés dans ce paragraphe.

Le modèle permet d'estimer le facteur de duplication en fonction de la taille des zones écran utilisées et de la taille moyenne des primitives composant la scène. En fait, la taille prise en compte est celle des boîtes englobantes des primitives. Soient l et h les largeurs et hauteurs moyenne des boîtes englobantes des primitives de la scène. En supposant une distribution aléatoire équiprobable des primitives sur l'écran, le facteur de duplication moyen est donné par la formule suivante (voir [MOLN91] pour une description complète du calcul) :

$$D = \left(\frac{l+L}{L}\right) \times \left(\frac{h+H}{H}\right)$$

Si l'on considère des zones carrées et des facettes incluses dans des boîtes englobantes carrées, on a alors $L = H$ et $l = h$. Le facteur de duplication est alors donné par la formule suivante :

$$D = \left(\frac{l+L}{L}\right)^2 = \left(1+\frac{l}{L}\right)^2$$

La Figure 3.18 présente les courbes $D = f(l)$ pour différentes valeurs de L. Les zones considérées ont pour taille respectivement 16 x 16, 32 x 32, 64 x 64 et 128 x 128 pixels.

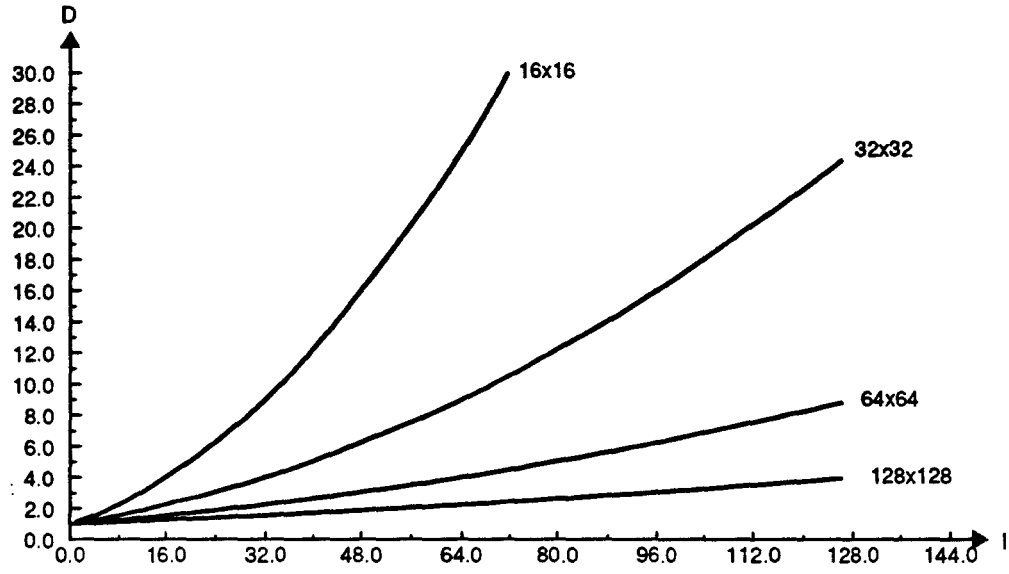


Figure 3.18 : facteur de duplication en fonction de la taille des facettes

Nous allons maintenant étudier l'influence du facteur de duplication sur les performances d'un système massivement parallèle.

3.7.2 Performances globales du système

Nous avons vu dans le paragraphe 3.5.4 que les performances intrinsèques d'une unité de conversion massivement parallèle ne dépendent pas de la surface des primitives générées dans la zone de conversion. En conséquence, la performance globale (en nombre d'objets par seconde) d'un système à réalisation partielle séquentielle est égale à la performance intrinsèque de son unité de conversion divisée par le facteur de duplication moyen.

Dans la suite, nous ne caractériserons que les systèmes utilisant un découpage par zones rectangulaires (généralement carrées). Nous reviendrons dans le paragraphe 3.8 sur les systèmes utilisant un découpage par lignes.

3.7.2.1 système à parallélisme objet

Comme nous l'avons déjà signalé, le principe même du parallélisme objet ne permet que de calculer la performance maximale théorique d'une unité de conversion objet. Il ne sera donc possible que de calculer la puissance maximale du système utilisant une telle unité.

La puissance de l'unité (si le temps de chargement est inférieur au temps de conversion) est de

$\frac{No}{L \times H \times To}$ objets/s (si To est exprimé en secondes).

La puissance maximale du système est donc (en supposant des zones carrées, i.e. $H = L$)

$$\frac{No}{L \times L \times To} \times \frac{1}{D} = \frac{No}{To} \times \left(\frac{1}{l+L}\right)^2$$

Si l'on considère notre pipeline de référence (88 processeurs, 33 MHz, zones 32×32), on obtient alors, pour des facettes de boîte englobante 20×20, une performance maximale d'environ 1 million de facettes par seconde. Par ailleurs, nous avons vu que le fait d'augmenter le nombre de processeurs sans augmenter le débit de chargement ne modifie pas les performances du système.

3.7.2.2 système à parallélisme pixel

La puissance effective du système est

$$\frac{1}{Tp} \times \frac{1}{D} = \frac{1}{Tp} \times \left(\frac{L}{l+L}\right)^2$$

A titre d'exemple, considérons un système basé sur le Renderer de Pixel-Planes 5. On a alors $Tp = 7,5\mu s$, $L = 128$. Si l'on considère l'affichage de scènes composées de petites facettes, on peut considérer que $l = 20$. Un tel système est donc capable d'afficher environ 100.000 facettes/s. On retrouve ainsi les performances annoncées dans [FUCH89]. Le même calcul permet de retrouver les performances du Renderer de PixelFlow (environ 1,5 Mfacettes/s [MOLN92]).

Si nous considérons notre exemple de référence (réseau multi-pipeline, 33 MHz, une expression générée à la fois), les performances obtenues en fonction du nombre de processeurs pixels utilisés sont les suivantes (affichage de facettes contenues dans des boîtes englobantes 20×20) :

- 128×128 : 3,5 millions de facettes par seconde.
- 64×64 : 2,7 millions de facettes par seconde.
- 32×32 : 1,8 millions de facettes par seconde.

Comme deuxième exemple, considérons un réseau multi-pipeline générant les sept expressions en parallèle (et donc sept fois plus complexe que l'exemple de référence). On obtient alors les performances suivantes :

- 128×128 : 24,9 millions de facettes par seconde.
- 64×64 : 19,3 millions de facettes par seconde.
- 32×32 : 12,6 millions de facettes par seconde.

Dans le cas d'un réseau pixel, la taille des zones influe directement sur la complexité matérielle du système (par exemple, un réseau 64×64 est quatre fois plus complexe qu'un réseau 32×32). Par ailleurs, augmenter la taille des zones permet de diminuer le facteur de duplication, et donc d'augmenter la puissance du système (car la performance intrinsèque d'une unité pixel ne dépend pas de la taille du réseau).

La Figure 3.19 présente les rapports de performances obtenus sur les réseaux de taille 32×32, 64×64, et 128×128. On peut ainsi constater, par exemple, que pour des applications utilisant des petites facettes (boîtes englobantes inférieures à 16×16), les écarts de performances entre les différents réseaux sont relativement faibles. Il peut alors être

intéressant de se limiter à un réseau 32×32, sachant que celui-ci est quatre fois moins complexe qu'un réseau 64×64, et seize fois moins qu'un réseau 128×128.

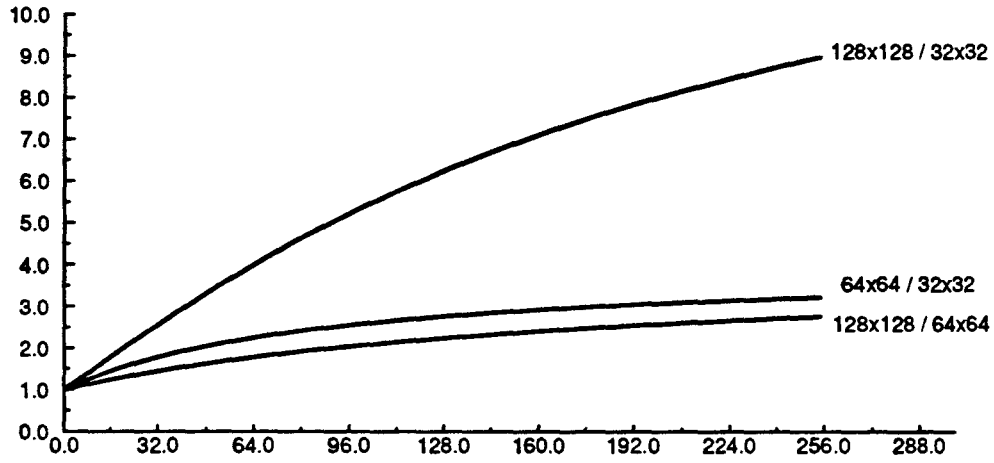


Figure 3.19 : performances comparées pour différentes tailles de réseaux

3.7.3 Fréquence maximale d'animation

La fréquence d'animation maximale supportée par un système graphique est un facteur essentiel pour juger de sa qualité et de son adéquation à son utilisation dans des applications temps réel de type simulation (simulation de vol ou de conduite, mondes virtuels, animation moléculaire...).

En effet, des mesures de performances exprimées en nombre de polygones par seconde ne permettent pas à elles seules de juger de la puissance d'un système graphique. Par exemple, imaginons un système capable de traiter 1.000.000 de polygones par seconde (100 pixels) . S'il garantit une fréquence d'animation de 30 Hz, il sera alors possible d'afficher environ 33.000 petits polygones en temps réel à 30 Hz. Un tel système est alors parfaitement adapté aux applications de simulations temps réel. Par ailleurs, la fréquence d'animation décroissant avec la complexités des scènes, il est possible d'afficher des scènes plus complexes en sacrifiant la rapidité d'animation. Un tel système est donc parfaitement adapté à tous les types d'applications. Par contre, si le système ne peut garantir que 1 Hz (exemple extrême), alors il sera possible d'afficher rapidement (une image par seconde) des scènes très complexes (un million de primitives), mais aucune manipulation interactive ne sera possible, même sur des scènes de complexité moindre.

Nous avons donné dans le premier chapitre une définition du temps réel en synthèse d'images. Nous rappelons ici quelques chiffres :

- une application interactive (visualisation CAO) nécessite une fréquence d'animation supérieure ou égale à 10 images par seconde (10 Hz).
- une application temps réel classique (simulateur bas de gamme) nécessite une fréquence d'animation de 30 images par seconde.
- une application temps réel haut de gamme (simulateur haut de gamme, mondes virtuels) nécessite une fréquence d'animation de 60 images par seconde.

- certaines applications peuvent nécessiter des fréquences d'animation encore plus élevées. Plus la fréquence d'affichage est élevée, plus le confort visuel est grand.

Par ailleurs, en visualisation monoscopique, il est inutile que la fréquence d'animation de la machine soit supérieure à la fréquence de rafraîchissement de l'écran de visualisation (100 Hz semble être une limite supérieure raisonnable).

Pour calculer la fréquence maximale d'animation des systèmes graphiques à réalisation partielle séquentielle, nous nous placerons dans le cas le plus défavorable du point de vue de l'animation sur les systèmes à découpage en zones, c'est à dire une scène comprenant au moins un objet par zone (cette situation est classique dans les applications de type simulation).

3.7.3.1 Unité de conversion objet

Pour une zone donnée, le temps total de traitement (conversion + transfert des pixels vers la mémoire de trame) s'exprime par $Max(t_o, So)$. Vues nos hypothèses de travail, le pipeline de conversion doit effectuer au minimum une passe de conversion dans chacune des zones écran, et ce quel que soit le découpage choisi. La fréquence maximale d'animation est obtenue quand, pour chaque zone, $t_o = L \times H \times To$ (une passe seulement). Le temps total de génération de l'image est alors $Nz \times Max(L \times H \times To, So)$ [1]

Le temps de génération de l'image est donc toujours supérieur ou égal à $Nz \times L \times H \times To$. Par conséquent, la fréquence d'animation est donc toujours inférieure ou égale à

$$\frac{1}{Nz \times L \times H \times To}.$$

Ce chiffre représente une borne supérieure (la machine ne pourra jamais afficher plus vite) qui dépend principalement de la fréquence de fonctionnement du pipeline.

Si l'on considère un écran 1280×1024 et des zones carrées ($L = H$), on a alors $Nz = 1280 \times 1024 / L^2$. La fréquence maximale d'animation est donc indépendante de la taille des zones et du nombre de processeurs. Elle est égale à

$$\frac{1}{1280 \times 1024 \times To}$$

Si le pipeline fonctionne à 33 MHz, sa fréquence maximale d'animation est donc de 25 Hz. Cette fréquence est conservée tant que le nombre d'objets par zone ne dépasse pas le nombre de processeurs du pipeline.

3.7.3.2 Unité de conversion pixel

La fréquence maximale d'animation est obtenue quand chaque zone contient un seul objet. On a alors, pour chaque zone, $t_p = Tp$. Le temps de génération total de l'image est alors $Nz \times Max(Sp, Tp)$.

Considérons notre exemple de référence (écran 1280×1024, réseau 32×32, 33 MHz, une expression générée à la fois). On a alors $Tp = 210ns$. Comme nous l'avons montré dans le paragraphe 3.4.3, Sp dépend du degré de parallélisme utilisé pour transférer les pixels du réseau vers la mémoire de trame. Nous envisagerons trois situations technologiquement maîtrisables (de la plus simple à la plus complexe) :

1. nous négligeons les pertes dues au chargement de la première zone et au transfert de la dernière.

- si le réseau ne possède qu'une seule sortie RVB, il faut alors 1024 cycles pour transférer tous les pixels. Si on utilise une sortie 24 bits (R,V et B en parallèle), on obtient alors un temps de génération de 39,3 ms, soit une fréquence maximale d'animation de 25 Hz. Cette fréquence est conservée tant que le nombre moyen d'objets par zone est inférieur à 146.
- si le réseau possède 32 sorties RVB (une par ligne) sur 8 bits (R, V et B en série), le temps total de transfert des pixels est 3×64 cycles. On obtient ainsi un temps de génération total de 7,3 ms, soit une fréquence maximale d'animation de 135 Hz. Cette frequence est conservée tant que le nombre moyen d'objets par zone est inférieur à 27.
- si le réseau possède 32 sorties RVB sur 24 bits, la fréquence maximale d'animation est alors d'environ 400 Hz. Cette fréquence est conservée tant que le nombre moyen d'objets par zone est inférieur à 9.

Les courbes de fréquence en fonction du nombre moyen d'objets par zone sont les suivantes :

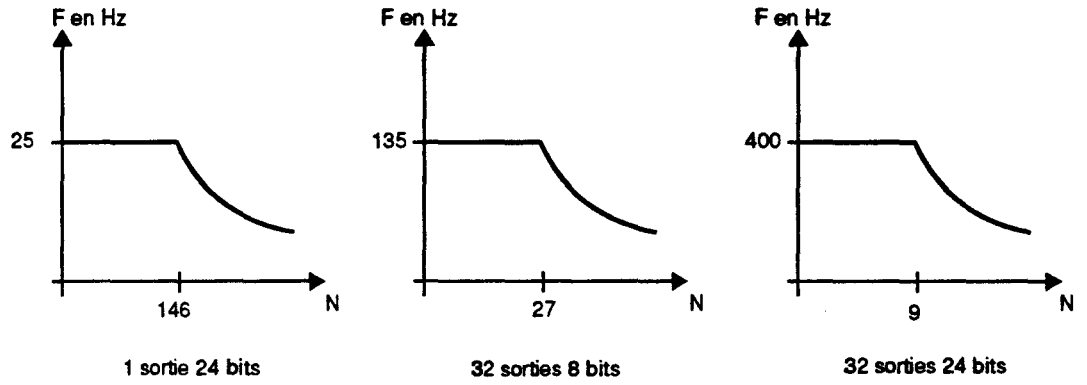


Figure 3.20 : fréquence d'animation du réseau 1

Nous pouvons refaire ces calculs avec notre deuxième exemple (écran 1280×1024, 33 MHz, zones 32×32, sept expressions générées en parallèle). On obtient alors les courbes suivantes :

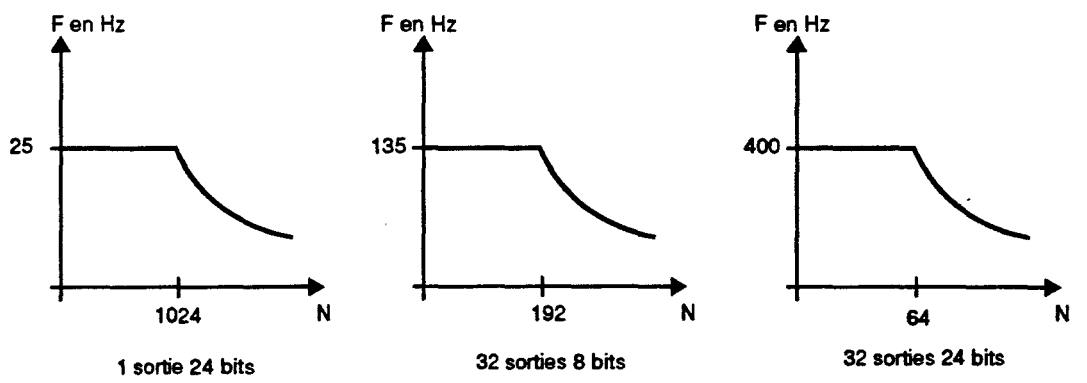


Figure 3.21 : fréquence d'animation du réseau 2

3.7.3.3 Conclusion

Au vu de cette étude, il apparaît clairement que les unités de conversion objet sont plus limitées que leurs homologues pixel. En effet, la fréquence maximale d'animation dépend uniquement de la fréquence de fonctionnement du pipeline. Tout accroissement de la fréquence d'animation repose donc soit sur une amélioration technologique (par exemple passer à 66 MHz), soit sur un accroissement de la complexité matérielle (il est possible d'utiliser en parallèle deux pipelines à 33 MHz pour obtenir l'équivalent d'un pipeline à 66 MHz).

En revanche, l'unité de conversion pixel (de type multi-pipeline) permet d'obtenir des fréquences d'animation très élevées (plus de 100 Hz) en utilisant certes beaucoup de boîtiers mémoire en parallèle (au minimum un par ligne), mais en utilisant une technologie courante (33 MHz).

3.8 Retour sur SAGE, GSP-NVS et Pixel-Planes

A ce stade de notre étude, il nous apparaît intéressant d'analyser les machines à réalisation partielle proposées ces dernières années, à savoir SAGE, GSP-NVS et la "gamme" Pixel-Planes[1], en fonction des différents critères que nous avons définis. Cette analyse est d'autant plus intéressante qu'elle permet d'une part de retrouver les performances annoncées dans les articles décrivant ces machines, et d'autre part de cerner avec précision leurs limites.

3.8.1 SAGE

SAGE implémente en quatre circuits intégrés un pipeline de 1024 processeurs pixels représentant une ligne de l'écran. Par ailleurs, les chemins de données étant entièrement parallèles, le système est capable de générer une primitive par cycle (le cycle de base du système est de 40 ns). On donc

$$L = 1024,\ H = 1,\ Tp = 40ns$$

La performance intrinsèque de l'unité de conversion de SAGE est donc de 25 millions de primitives par seconde (en configuration parallèle ou en configuration série).

Par ailleurs, SAGE utilisant un découpage par ligne, le taux de duplication moyen est égal à la hauteur moyenne des facettes. On en déduit donc les performances globales suivantes (en fonction de la taille des boîtes englobantes des facettes) :

- boîtes englobantes 10×10 : 2,5 Mfacettes/s
- boîtes englobantes 20×20 : 1,25 Mfacettes/s
- boîtes englobantes 30×30 : 840.000 facettes/s

On retrouve bien les performances annoncées dans [GHAR88] (près de 1 Mfacettes/s pour des polygones de hauteur moyenne 32 pixels).

Le temps de transfert des pixels dépend de la configuration choisie pour les quatre circuits intégrés :

- en configuration série, le système fournit une valeur RVB par cycle. On a alors $Sp = 1024 \times 40ns = 40\mu s$. La fréquence maximale d'animation est alors d'environ 25 Hz pour un écran 1024×1024.

1. se référer à la description de ces machines dans le chapitre 2

- en configuration parallèle, le système fournit quatre valeurs par cycle. On a alors $Sp = 1024 \times 10ns = 10\mu s$. La fréquence maximale d'animation est alors d'environ 100 Hz.

Il est intéressant de comparer les performances de SAGE avec celles d'un réseau cellulaire multi-pipeline carré de complexité similaire, c'est-à-dire 32×32 (la figure présente les deux configurations).

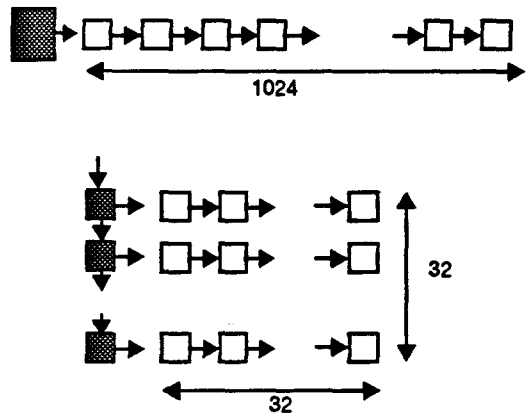


Figure 3.22 : pipeline scanline et réseau multi-pipeline

Les deux unités ont les mêmes performances intrinsèques (une facette par cycle si toutes les expressions sont générées en parallèle). Par contre, les facteurs de duplication des deux systèmes sont fort différents :

- dans le cas du pipeline, le facteur de duplication moyen (que nous appellerons D_p) est égal à la hauteur moyenne des facettes générées.
- dans le cas du multi-pipeline, le facteur de duplication moyen (que nous appellerons D_{mp}) peut être estimé en utilisant les courbes présentées au paragraphe précédent.

L'accélération obtenue sur le réseau multi-pipeline par rapport au pipeline s'exprime donc par

$$A_{m/mp} = \frac{D_{mp}}{D_p}$$

Le nombre de processeurs étant fixé (et donc la taille des zones dans les cas du multi-pipeline), ce rapport ne dépend que de la taille moyenne des facettes (soit T). de duplication. La Figure 3.23 présente courbe obtenue en utilisant la formule du paragraphe Facteur de duplication pour estimer le facteur de duplication (T représente la largeur des boîtes englobantes des facettes) :

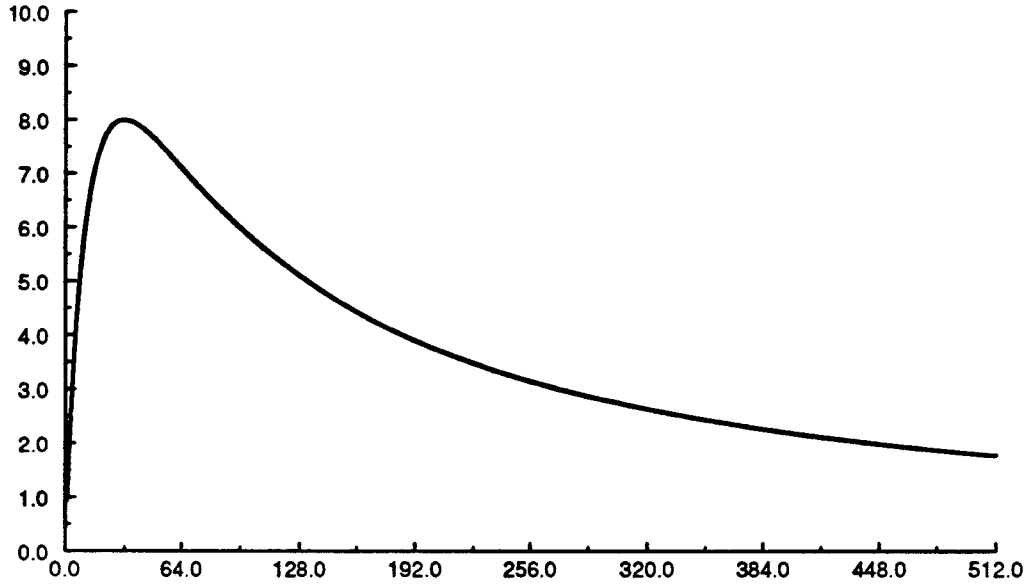


Figure 3.23 : accélération obtenue sur un multi-pipeline

On constate que le réseau a toujours des performances supérieures au pipeline. Par ailleurs, pour des facettes petites et moyennes (boîte englobante variant entre 5 et 100), le gain obtenu est compris entre 6 et 8 (voir Figure 3.24).

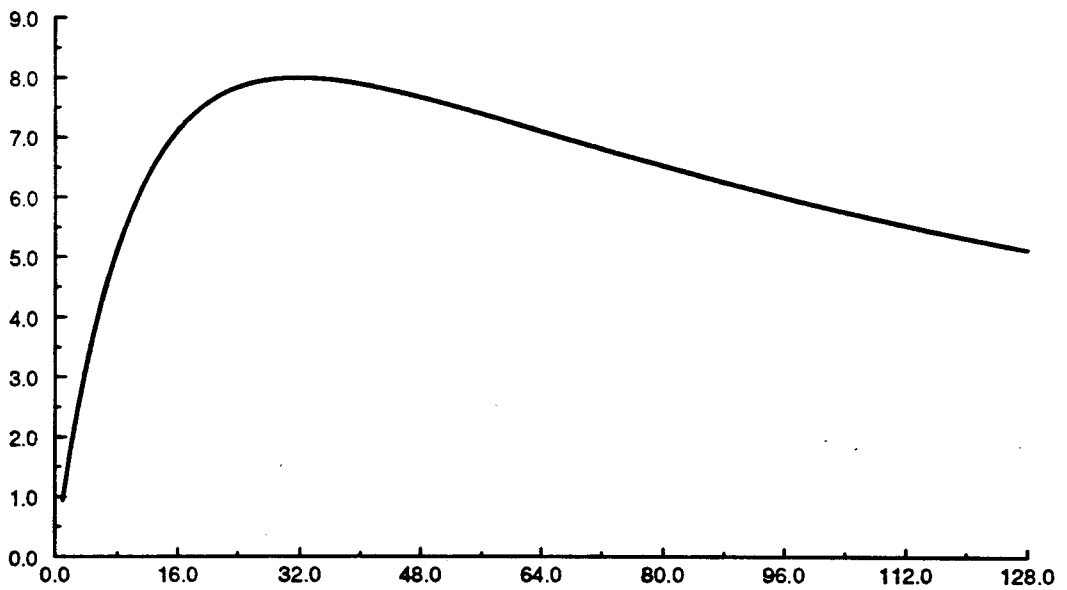


Figure 3.24 : accélération dans les cas courants

3.8.2 GSP-NVS

GSP-NVS implémente un pipeline de 1000 processeurs objets[1]travaillant sur une ligne écran(1280 pixels) à 20 MHz. Par ailleurs, le système n'utilise pas de mémoire locale, mais travaille directement sur la mémoire globale. Le temps de chargement d'une facette est de 200 ns. Il est inférieur à celui que nous avons proposé dans ce chapitre car GSP-NVS utilise un découpage en lignes et une méthode de suivi de contour (il y a donc moins de coefficients à transférer que dans le cas d'une méthode par équation). Notons que, dans la version proposée, chargement et conversion était effectués l'un après l'autre. Pour notre étude, nous supposerons qu'ils sont effectués en parallèle (cela demande uniquement de modifier les circuits afin de prévoir un chemin de données séparé pour le chargement). On a donc

$$No = 1000,\ L = 1280,\ H = 1,\ To = 50ns,\ So = 0,\ Co = 200ns$$

D'après les calculs effectués dans ce chapitre, le nombre optimum de processeurs est donc de 320. Il est donc a priori inutile d'en utiliser 1000. Ceci est vrai dans les cas d'un découpage en zone (le sujet principal de ce chapitre), mais faux dans le cas de GSP-NVS.

En effet, GSP-NVS utilise un découpage en lignes, ce qui permet de ne charger un objet que sur la première ligne où il est présent. Il reste alors dans son processeur jusqu'à ce qu'il ait été entièrement traité. Par ailleurs, quand, sur une ligne donnée, il y a plus d'objets que de processeurs, les triangles qui ne peuvent être traités sont conservés dans le Y-buffer (voir la description de GSP-NVS dans le chapitre 2) et traités lors d'une seconde passe (lors de cette passe, seules les lignes surchargées sont traitées). En définitive, chaque triangle de la base de données n'est chargé qu'une seule fois dans le pipeline tout au long de la création de l'image.

En conclusion, on peut considérer que, en moyenne, le temps de chargement est toujours inférieur au temps de conversion des 1000 objets sur la ligne. On peut donc calculer la puissance maximale de l'unité de conversion (rappelons toutefois que cette puissance n'est pas atteinte sur GSP-NVS car chargement et conversion utilisent les mêmes chemins de données, et ne peuvent donc pas être effectués en parallèle) :

$$\frac{No}{L \times H \times To} = \frac{1000}{1280 \times 50 \times 10^{-9}} = 15 Mfacettes/s$$

Comme SAGE, GSP-NVS utilise un découpage par lignes. Le facteur de duplication moyen est donc encore égal à la hauteur moyenne des facettes. Les performances globales maximales de GSP-NVS sont donc les suivantes :

- boîtes englobantes 10×10 : 1,5 Mfacettes/s
- boîtes englobantes 20×20 : 750,000 Mfacettes/s
- boîtes englobantes 30×30 : 500,000 facettes/s

On constate donc que le pipeline de 1000 processeurs utilisant un découpage en ligne a une puissance comparable à notre système de référence (qui utilise moins de 100 processeurs). En effet, la souplesse apportée par le découpage en lignes est entièrement contrebalancée par le facteur de duplication élevé (hauteur moyenne des triangles).

1. chiffre déduit car non cité clairement dans [DEER88]

3.8.3 Pixel-Planes

Pixel-Planes 5 et PixelFlow sont actuellement les seules machines massivement parallèles[1] à réalisation partielle utilisant un découpage en zones rectangulaires (128×128). Le principe du parallélisme massif pixel à alimentation par diffusion est utilisé depuis la première machine de la gamme (Pixel-Planes 4). Celui de réalisation partielle à découpage par zones rectangulaires 128×128 a été introduit sur Pixel-Planes 5, et conservé sur PixelFlow.

L'étude que nous venons de mener dans ce chapitre confirme de manière formelle que la combinaison parallélisme massif pixel/réalisation partielle par zones est une excellente solution pour réaliser des systèmes graphiques performants. Cependant, elle permet également de suggérer des améliorations augmentant le rapport performances/coût :

- pour beaucoup d'applications utilisant des petites facettes (100 pixels), il est possible d'utiliser des zones 64×64 au lieu de 128×128 sans dégrader exagérement les performances. Le coût de l'unité de conversion est alors quatre fois moindre.
- pour certaines applications utilisant des très petites facettes (mois de 50 pixels), il est envisageable d'utiliser un réseau 32×32.
- le multi-pipeline est moins complexe et plus régulier que le mécanisme d'arbres de diffusion. Par ailleurs, il semble mieux adapté à une réalisation VLSI, et doit ainsi permettre soit de construire une machine moins coûteuse, soit, à coût égal, de construire une machine plus puissante.

3.9 Conclusions

L'étude que nous avons menée dans ce chapitre avait pour but de comparer de manière formelle les unités de conversion objets et pixels utilisées dans des architectures à réalisation partielle.

Les unités de conversion objet sont fortement limitées par le débit de chargement des objets dans le cas d'un découpage en zones rectangulaires. Nous avons ainsi constaté que les performances d'un pipeline objet peuvent difficilement dépasser quelques millions de facettes par seconde. L'utilisation d'un découpage en ligne (type GSP-NVS) permet de s'affranchir du problème du chargement des objets, mais au prix d'un accroissement important du facteur de duplication qui limite les performances à un niveau comparable. Par ailleurs, la nature même du parallélisme massif objet (conversion des objets dans l'ordre du balayage écran) limite considérablement la fréquence maximale d'animation du système.

Les unités de conversion pixel ne possèdent aucun de ces défauts. En effet, elles ne nécessitent pas de phase de chargement. Par ailleurs, le fait qu'elles puissent traiter plusieurs lignes de pixel en parallèle autorise une parallélisation efficace de l'accès à la mémoire de trame, et donc des fréquences d'animation très importantes (supérieures à 100 Hz). A complexité égale, nous avons montré qu'un réseau multi-pipeline est environ deux fois plus rapide qu'un pipeline objet. La puissance des réseaux 32×32 que nous avons étudiés varie de 1,8 millions de facettes par seconde pour le plus simple, à environ 13 millions pour le plus complexe (pour des boîtes englobantes 20×20). Pour certaines applications utilisant des très petites facettes (boîtes englobantes 10×10), la puissance du réseau 32×32 peut atteindre 20 millions de facettes par seconde. Par ailleurs, il est possible, au prix d'une augmentation de la complexité du système, d'augmenter encore les performances en utilisant des réseaux plus grands (64×64 ou 128×128)

1. PixelFlow est en cours de construction

Par ailleurs, nous avons montré que, dans les cas du parallélisme pixel, l'utilisation d'un découpage en zones rectangulaires apporte un gain important par rapport au découpage en lignes utilisé par exemple sur SAGE. En effet, la réduction du facteur de duplication permet, à complexité matérielle égale, d'augmenter les performances (facteur 6 à 8 dans les cas courants).

Chapitre 4

Conversion de facettes

4.1 Définition d'une facette

Nous avons vu au chapitre précédent que l'utilisation d'un découpage en zones rectangulaires imposait d'utiliser une méthode par équations. Une facette sera donc définie par sept équations bilinéaires définissant le contour, la profondeur et la couleur (ou la normale en cas d'interpolation de Phong) :

- Contour

$$a1x+b1y+c1 \geq 0$$
$$a2x+b2y+c2 \geq 0$$
$$a3x+b3y+c3 \geq 0$$

- Profondeur

$$z = a4x+b4y+c$$

- Couleur

$$R = a5x+b5y+c5$$
$$V = a6x+b6y+c6$$
$$B = a7x+b7y+c7$$

Nous présentons dans le paragraphe suivant deux méthodes de calcul incrémental d'une expression bilinéaire : la première est destinée aux pipelines objet, la deuxième aux réseaux multi-pipelines pixels.

4.2 Calcul incrémental d'expressions linéaires

4.2.1 Unité de conversion objet

Nous détaillons ici le principe de calcul incrémental d'une expression bilinéaire en x et y de type $f(x,y) = ax+by+c$, où (x,y) sont les coordonnées du pixel courant et (a,b,c) des coefficients constants pour une expression donnée.

(x,y) peut se décomposer en $g(x)+h(y)$ avec $g(x) = ax$ et $h(y) = by+c$. Par ailleurs, $h(y+1) = h(y)+b$ et, sur une ligne donnée (i.e. y constant), $f(x+1,y) = f(x,y)+a$. La Figure 4.1 illustre ce principe de fonctionnement.

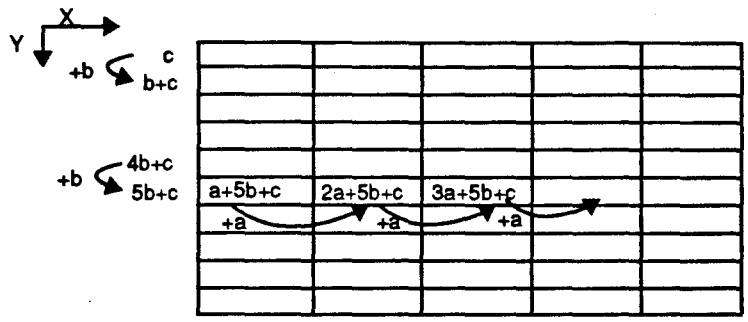


Figure 4.1 : calcul incrémental de ax+by+c

Une structure composée d'un registre et d'un additionneur suffit pour calculer incrémentalement une expression du type $\alpha x+\beta$. Une telle structure (que l'on peut appeler interpolateur linéaire) délivre une nouvelle valeur à chaque cycle horloge.

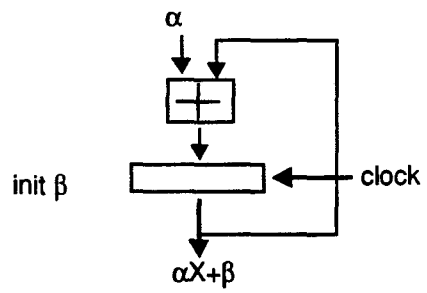


Figure 4.2 : calcul incrémental de αX+β

Pour calculer incrémentalement $ax+by+c$, on peut utiliser deux interpolateurs. Le premier est utilisé en début de ligne pour calculer la nouvelle valeur de $by+c$, cette valeur permettant d'initialiser le registre du deuxième interpolateur calculant la valeur de $ax+by+c$ pour tous les pixels de la ligne. Toutefois, il est également possible d'utiliser un unique interpolateur sous réserve de disposer de registres de sauvegarde pour mémoriser les valeurs de a, b et la valeur courante de $by+c$ (voir Figure 4.3).

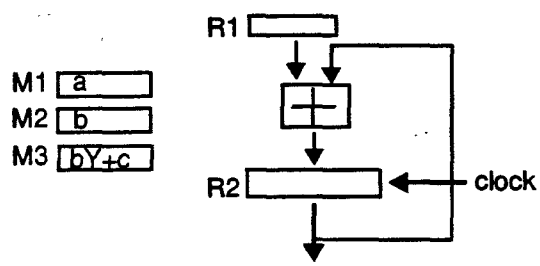


Figure 4.3 : interpolateur bilinéaire

L'interpolateur est utilisé en début de ligne pour calculer $by+c$, puis en chaque pixel pour calculer $ax+by+c$. Son algorithme de fonctionnement est alors le suivant :

```
a->M1
b->M2
c->M3
Pour chaque ligne Faire
  M3->R2
  M2->R1
  clock /* calcul nouveau bY+c */
  R2->M3 /* sauvegarde nouveau bY+c */
  M1->R1
  Pour chaque pixel de la ligne Faire
    clock /* calcul aX+bY+c */
```

Plusieurs solutions matérielles sont envisageables pour implémenter le principe de calcul que nous venons de décrire. Nous présenterons dans le paragraphe 4.3 celle que nous avons retenue.

4.2.2 Unité de conversion pixel

Nous nous intéressons ici au principe de calcul d'une expression bilinéaire $ax+by+c$ dans un réseau multi-pipeline pixel. Le réseau est composé de deux types de cellule :

- les cellules du pipeline vertical calculent l'expression linéaire $h(y) = by+c$.
- les cellules des pipelines horizontaux calculent, pour une valeur donnée de y, l'expression linéaire $f(x, y) = ax+h(y)$.

La cellule de base du réseau effectuant le calcul de $l(u) = \alpha u+\beta$ est schématisée sur la Figure 4.4

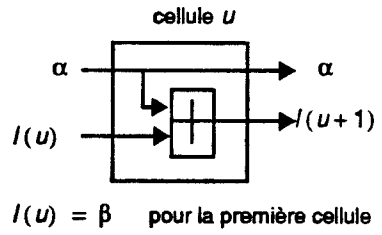


Figure 4.4 : cellule de base d'un multi-pipeline

Le réseau de calcul de $ax+by+c$ est donc le suivant :

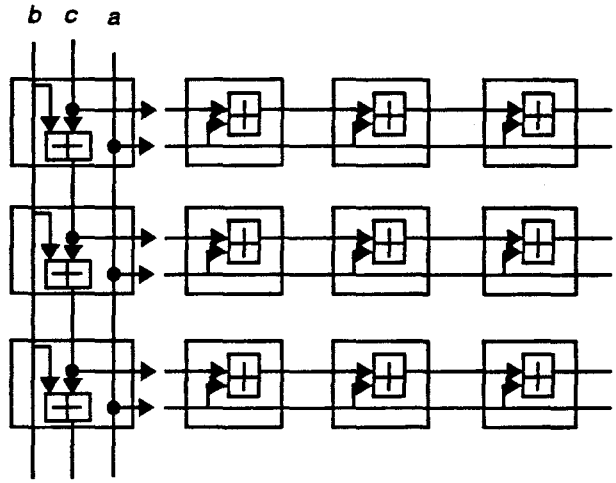


Figure 4.5 : réseau de calcul d'une expression bilinéaire

Nous étudions dans la suite un processeur facette avec interpolation de Phong (calcul des normales et non des couleurs) destiné à un pipeline objet. L'intégration VLSI de ce processeur a été réalisée avec le logiciel de CAO VLSI SOLO 1400. Nous présentons ensuite l'étude de la cellule élémentaire d'un réseau pixel

4.3 Processeur facette pour unité de conversion objet

Le processeur facette est principalement composé de sept interpolateurs bilinéaires. L'élimination des parties cachées est effectuée dans le pipeline. Chaque processeur est en effet muni d'un opérateur de Zbuffer permettant d'effectuer l'élimination des parties cachées entre son objet et celui de son voisin de gauche, l'objet visible étant alors transmis au voisin de droite.

Un contrôleur connecté au premier processeur gére le séquencement du pipeline. Chaque processeur reçoit une commande, la traite et la transmet au processeur voisin. Ce mode de contrôle permer d'assurer facilement le désynchronisme des processeurs indispensable au bon fonctionement du pipeline. Ainsi, quand un processeur travaille sur le pixel i, son prédécesseur travaille sur le pixel i+1 et son successeur sur le pixel i-1.

Les processeurs du pipeline sont également connectés à un processeur hôte effectuant les calculs géométriques sur les objets et la transmission des paramètres des objets à convertir vers les processeurs objet

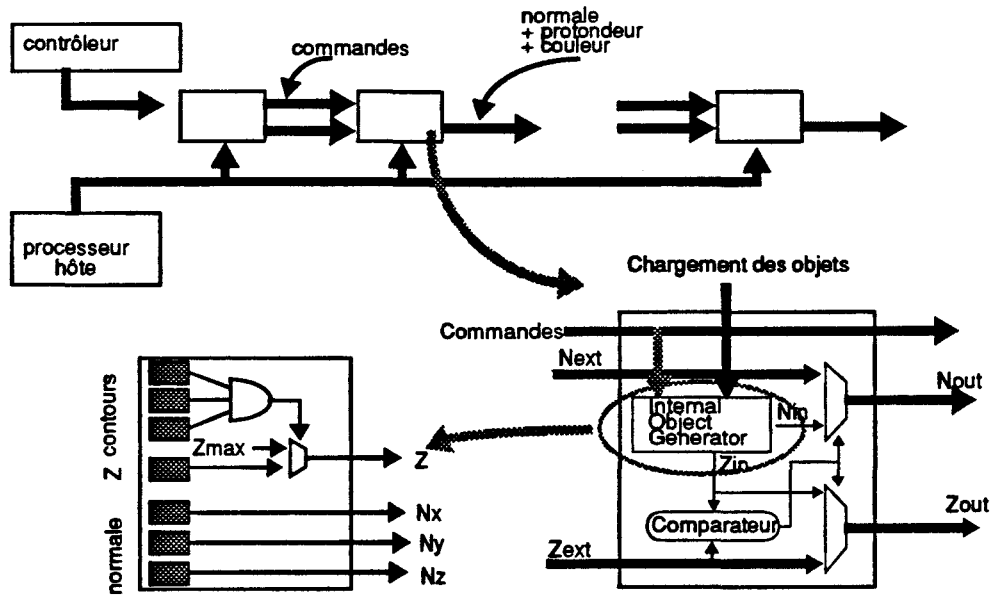


Figure 4.6 : le pipeline de conversion

Nous allons maintenant présenter en détail la réalisation VLSI du Processeur Facette. Le Circuit Intégré a été développé et simulé sous logiciel SOLO 1400 de la société ES2. Ce logiciel de CAO permet de concevoir des ASICs grâce à sa bibliothèque de cellules pré-caractérisées (technique dite semi-custom). Cette technique, simple à utiliser pour le concepteur puisqu'elle permet de ne pas travailler au niveau transistor (contrairement à la technique dite full-custom), ne conduit cependant pas à des niveaux d'intégration très poussés, ni à des vitesses de fonctionnement optimales. Par ailleurs, le logiciel SOLO 1400 laisse très peu de liberté au concepteur pour optimiser le placement des cellules. Ainsi, lors de la conception du processeur, certains choix architecturaux ont été dictés par les faiblesses du logiciel utilisé. Nous nous efforcerons donc de préciser dans la suite les raisons des choix effectués.

4.3.1 Description générale du processeur

Le processeur est principalement composé de sept PE1 calculant le contour, la profondeur et la normale. La couleur de base de l'objet est, quant à elle, mémorisée dans un registre puisque sa valeur est constante pour un objet donné. Chaque PE1 calcule une valeur sur 24 bits. Afin de réduire le nombre de broches du circuit (et ainsi de réduire la largeur du pipeline), seuls 12 bits sont conservés pour chacune des composantes de la normale. Pour cela, les PE1 calculant la normale sont connectés à une unité appelée pré-normalisateur effectuant le cadrage à droite de la normale. Par ailleurs, une unité effectuant la comparaison entre les profondeurs de l'objet interne et de l'objet précedent permet d'effectuer l'élimination des parties cachées.

Les PE1 sont commandés par un contrôleur transmettant ses commandes via le pipeline. L'envoi des commandes est synchronisé par une horloge commune à tous les processeurs du pipeline. Chaque PE1 est par ailleurs relié au processeur hôte via un module d'interface permettant le chargement de l'ensemble des coefficients (a,b,c) définissant l'objet (voir Figure

4.7).

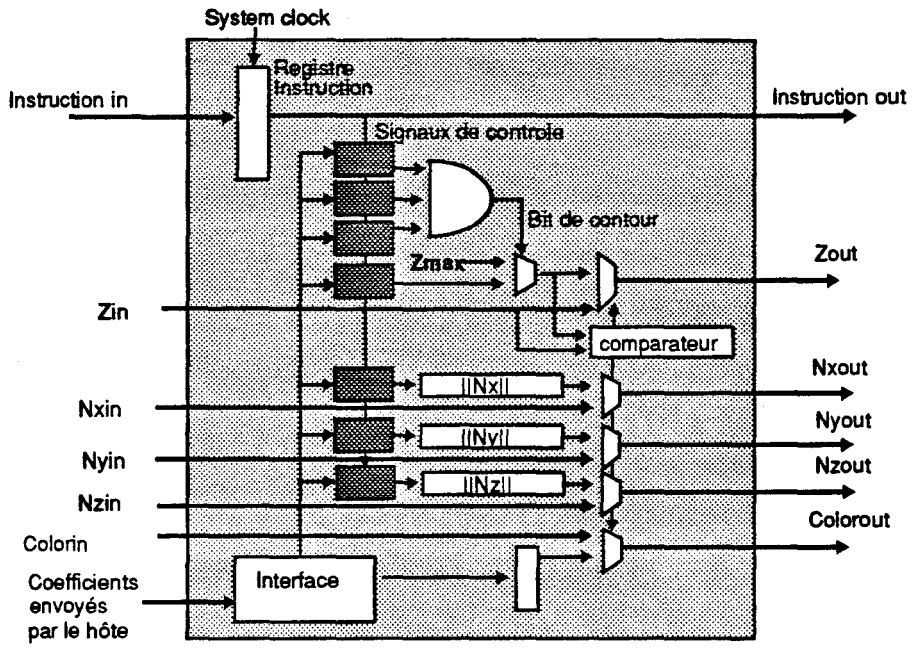


Figure 4.7 : le Processeur Facette

Nous allons maintenant décrire le fonctionnement des différentes unités. Nous commençons par décrire les unités de calcul (PE1, pré-normalisateur, élimination des parties cachées), puis nous présentons le principe des deux unités de contrôle du processeur (contrôleur et module d'interface).

4.3.2 Les opérateurs de calcul

4.3.2.1 Le PE1

Nous avons vu au paragraphe précédent que le PE1 est constitué d'un additionneur, d'un registre de travail et de trois registres de sauvegarde. Historiquement, notre étude sur les interpolateurs a débuté avec la conception d'un interpolateur du second ordre, nécessitant deux additionneurs, deux registres de travail et six registres de sauvegarde (au format 48 bits). Les connexions entre les registres de travail et de sauvegarde s'étant avérées complexes et coûteuses, une solution incorporant une unique RAM à la place des six registres de sauvegarde a été proposée. Cette solution a par la suite été reprise lors de la conception du PE1 (il faut noter que cette solution est parfaitement adaptée au logiciel SOLO 1400 puisque celui-ci dispose d'un générateur optimisé de RAM. Ainsi une RAM de trois mots de 24 bits est beaucoup plus compacte que trois registres statiques de 24 bits).

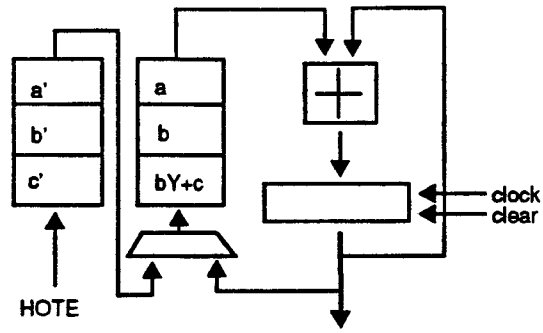


Figure 4.8 : architecture du PE1

Le programme de contrôle est alors le suivant

```
Pour chaque image Faire
  RAM1[0..2] -> RAM2[0..2]
  Pour chaque ligne Faire
    clear
    Lire_adresse_2                  /* lire bY+c */
    clock                           /* charger bY+c */
    Lire_adresse_1                  /* lire b */
    clock                           /* calcul nouveau bY+c */
    Ecrire_adresse_2                /* stocker bY+c */
    Lire_adresse_0                  /* lire a */
    Pour chaque pixel Faire
      clock                    /* calcul aX+bY+c */
```

Les données étant au format 24 bits, l'additionneur utilisé est également un additionneur 24 bits. Lors de la conception du PE1 sous SOLO 1400 s'est posé le problème de disposer d'additionneurs 24 bits suffisament rapides (16 MHz, i.e une addition en 60 ns). Une première étude effectuée par Samuel Degrande a montré que l'utilisation d'additionneurs parallèles 24 bits sous SOLO conduirait à une complexité matérielle (en nombre de transistors et en surface utilisée) beaucoup trop importante. Nous avons alors choisi d'utiliser des additionneurs série (la bibliothèque SOLO propose une cellule d'addition 1 bit) et de diviser l'additionneur 24 bits en deux additionneurs 12 bits pipelinés accompagnés d'une bascule permettant de mémoriser la retenue entre les poids faibles et les poids forts, ainsi que d'un registre 12 bits assurant la synchronisation entre la sortie des poids faibles et des poids forts (voir Figure 4.9).

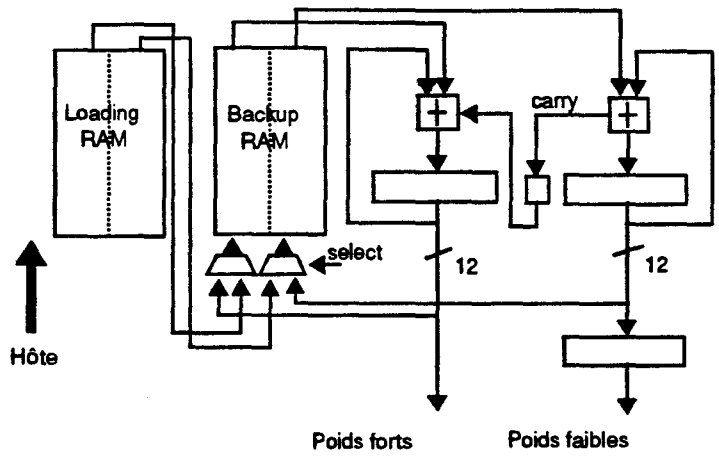


Figure 4.9 : architecture du PE1 (version pipelinée)

Le programme de contrôle est alors le suivant :

```
Pour chaque image Faire
  Select = 1
  Loading_RAM[0..2] -> Backup_RAM[0..2]  /* transfert des coeff */
  Select = 0
  Pour chaque ligne Faire
    Clear_Registres
    Read_backupRAM_adress2       /* lire bY+c */
    Clock1, Clock2               /* charger bY+c */
    Read_backupRAM_adress1       /* lire b */
    Clock1
    Clock2                        /* calculnouveau bY+c */
    Write_backupRAM_adress2       /* stocker bY+c */
    Read_backupRAM_adress0        /* lire a */
    Clock1                        /* amorcage du pipeline */
    Pour chaque pixel Faire
      Clock1, Clock2              /* calcul aX+bY+c */
```

Notons que le fait de pipeliner l'addition ne rajoute que deux cycles en début de chaque ligne (i.e. lors du calcul de bY+c). Lors de la génération des pixels de la ligne, le rythme est toujours d'un pixel par cycle horloge.

Les signaux de contrôle du PE1 sont donc les suivants :

- contrôle du calculateur : clock1, clock2, clear, select.
- contrôle de la RAM de travail : ad_t (adresse), l_t (signal de lecture), e_t (signal d'écriture).

- contrôle de la RAM de chargement : data (donnée 24 bits), ad_c, l_c, e_c.

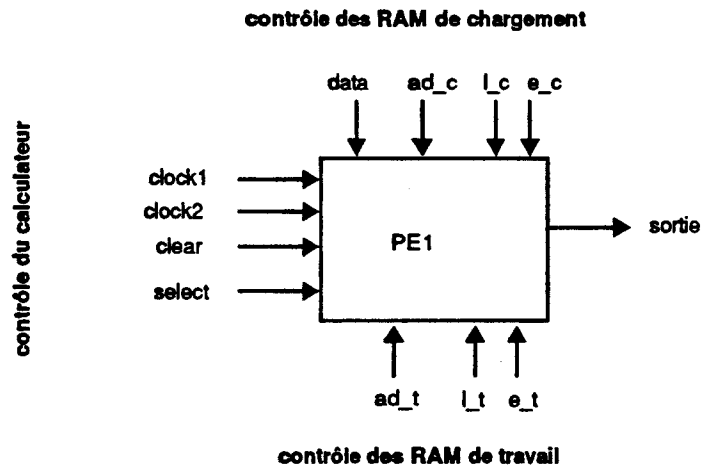


Figure 4.10 : signaux de contrôle d'un PE1

4.3.2.2 Calcul du contour

Le contour de l'objet est déterminé par l'intersection de trois demi-plans. L'équation d'un demi-plan est calculé par un PE1, mais seul le bit de signe du résultat est à prendre en compte (il sert à déterminer de quel coté du demi-plan se situe le pixel courant). Le registre de synchronisation du PE1 peut ainsi être supprimé (sans avoir à modifier son programme de contrôle), ce qui permet de reduire la complexité des trois PE1 calculant le contour.

Le choix d'orientation des demi-plans est le suivant : la valeur de l'équation est positive ou nulle (bit de signe = 0) à l'intérieur de la facette, et négative (bit de signe = 1) à l'extérieur. L'intersection des trois demi-plans peut donc être déterminée grace à un NOR logique entre les bits de signe des trois PE1. Si le résultat (appelé dans la suite *bit de contour*) est 0, le pixel courant est situé en dehors du contour de l'objet. Si le résultat est 1, le pixel est situé à l'intérieur du contour.

Nous avons vu lors de la description fonctionnelle du processeur que la profondeur de l'objet devait être forcée à Zmax en tous les pixels où celui-ci n'est pas présent. Lors de la réalisation du processeur, nous avons préféré conserver le bit de contour. La profondeur de l'objet est donc définie par 25 bits (24 bits pour la profondeur et 1 bit pour le contour).

Par ailleurs, il peut arriver que la profondeur d'un objet soit négative. Cela signifie que l'objet est situé derrière l'observateur, et ne doit alors pas être pris en compte lors de la visualisation. La solution retenue est de forcer à 0 le bit de contour quand la profondeur est négative. Le circuit de génération du bit de contour est donc le suivant :

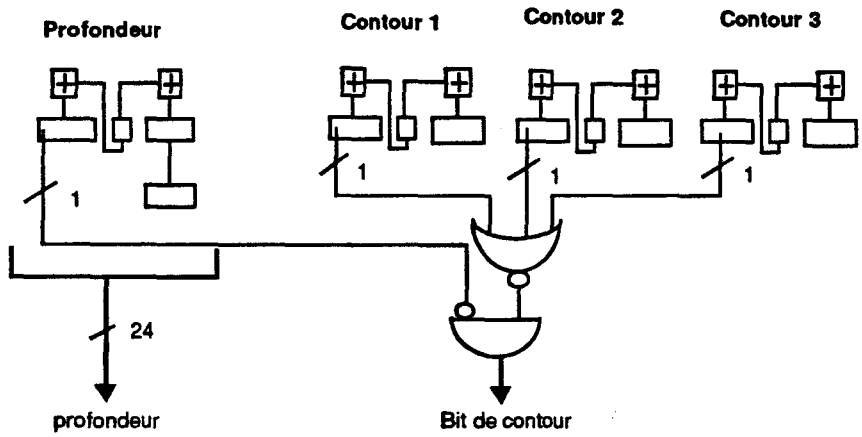


Figure 4.11 : génération du bit de contour

4.3.2.3 Calcul de la normale

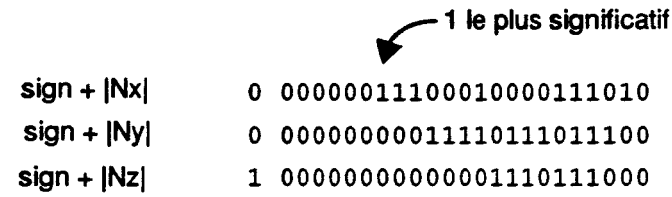
Les trois PE1 calculant les composantes Nx, Ny et Nz de la normale travaillent en format 24 bits. Toutefois, afin de réduire la largeur du pipeline de processeurs facette et de diminuer la complexité du processeur d'éclairement, Vincent Lefévère a proposé de n'utiliser que 12 bits par composante de la normale (au lieu des 24 calculés). Il a montré que ce format est suffisant pour assurer un rendu précis de l'éclairement diffus, et un rendu correct de l'éclairement spéculaire pour des valeurs limitées du coefficient de puissance spéculaire [LEFE92].

Nous avons donc défini une unité appelée pré-normalisateur permettant de transformer la normale sous la forme de trois composantes de 12 bits en conservant le maximum de précision. La méthode utilisée est la suivante (après chaque étape nous présentons un exemple) :

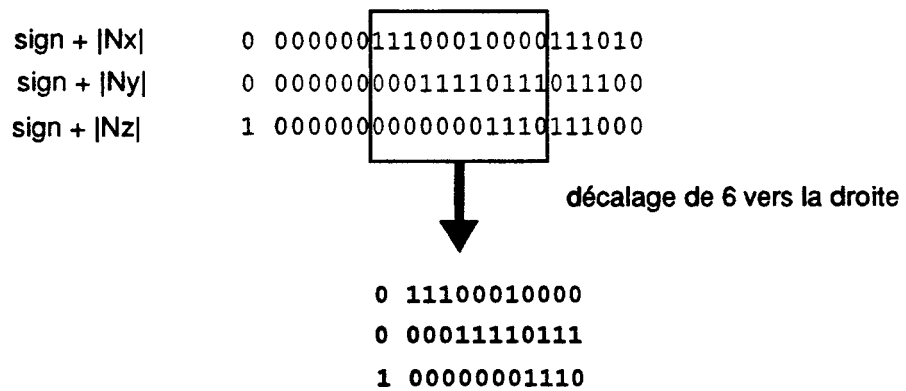
- Les trois composantes de la normale, calculées en complément à deux, sont transformées en codage signe+valeur absolue. Si le bit de signe est 0 (nombre positif), la composante n'est pas modifiée. Si le bit de signe est 1, tous les bits de la composante sont complémentés. Notons que cette transformation produit une erreur sur les nombres négatifs, puisque l'opposé d'un nombre X en complément à deux est $\overline{X}+1$, et non $\overline{X}$ comme nous le considérons. Cette erreur sur la normale produit toutefois un effet négligeable sur l'éclairement.

Nx	0 000000111000010000111010
Ny	0 000000000111110111011100
Nz	1 111111111111110001000111
	↓ calcul des valeurs absolues
sign + |Nx|	0 000000111000010000111010
sign + |Ny|	0 000000000111110111011100
sign + |Nz|	1 000000000000001110111000

- On détermine ensuite la plus grande des valeurs absolues des composantes. Cette valeur détermine les 11 bits à conserver parmi les 23 (en fait on détermine parmi les trois valeurs la position du 1 le plus significatif).



- Pour isoler les bits à conserver, on effectue un décalage variable vers la droite pour chacune des composantes (6 positions dans l'exemple). Les bits à conserver sont alors les 11 bits de poids faible. Le décaleur variable utilisé permet de décaler un nombre de 23 bits vers la droite d'un nombre de positions compris entre 0 et 12. Il est réalisé à partir de quatre décaleurs fixes de 8, 4, 2, et 1 positions. Par exemple, pour décaler un nombre de 7 positions vers la droite, on utilise le décaleur de 4, puis de 2, puis de 1 (le décaleur de 8 est shunté). Le décaleur variable est donc commandé par un signal 4 bits indiquant les décaleurs à activer.



Les figures suivantes présentent un schéma général du pré-normalisateur, le détail de l'unité de conversion complément à 2 / valeur absolue et le détail du décaleur variable.

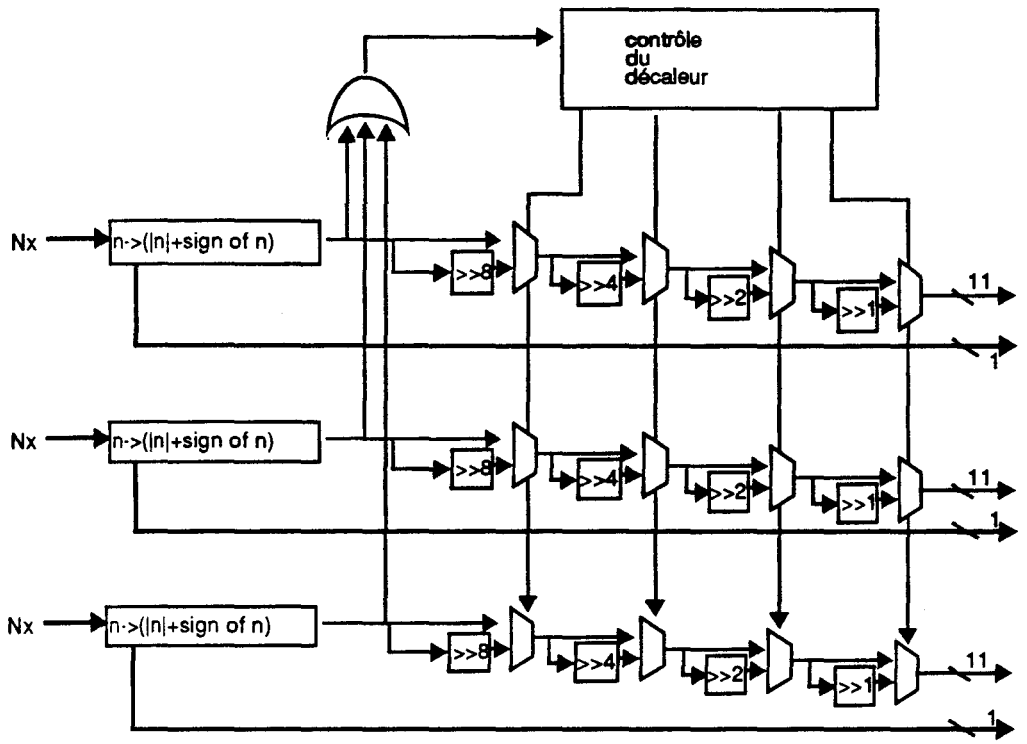


Figure 4.12 : le pré-normalisateur

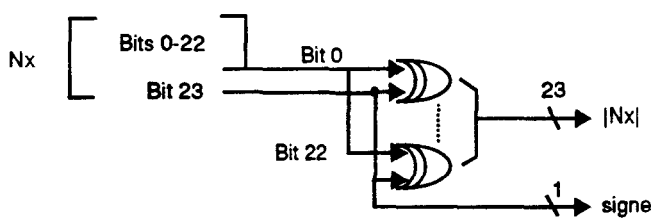


Figure 4.13 : transformation complément à 2 / valeur absolue

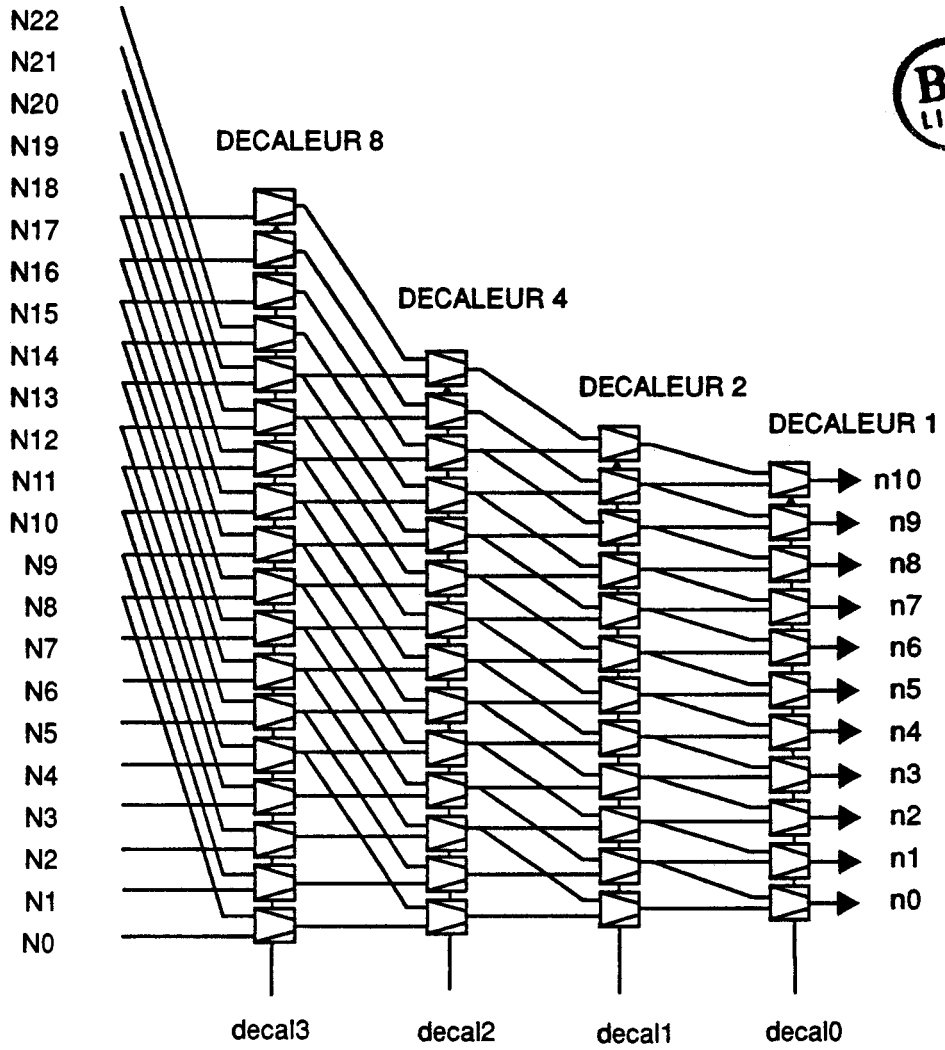


Figure 4.14 : décaleur variable 23->11

4.3.2.4 Mémorisation de la couleur de base

La couleur de base de la facette étant constante, un simple registre suffit pour la mémoriser. Toutefois, afin de rester compatible avec le système de chargement des coefficients des PE1, un double registre est utilisé. Le premier registre est utilisé pour stocker la nouvelle valeur de couleur fournie par le hôte (i.e. celle qui sera utilisée lors de la prochaine conversion), tandis que le second permet de stocker la couleur de l'objet en cours de conversion.

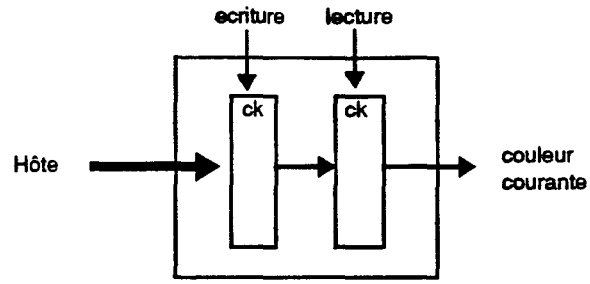


Figure 4.15 : mémorisation de la couleur de base

4.3.2.5 Elimination des parties cachées

L'élimination des parties cachées est effectuée dans le processeur facette au moyen d'un opérateur Zbuffer. Un comparateur 24 bits permet de choisir parmi les deux objets (celui du processeur et celui de son prédécesseur dans le pipeline) le plus proche de l'observateur (i.e celui qui possède la plus petite profondeur).

Le choix de l'objet à sélectionner dépend du résultat de la comparaison et des bits de contour des deux objets. En effet, la comparaison des deux profondeurs n'a de sens que si les deux objets sont effectivement présents au pixel courant (i.e. si les deux bits de contour sont à 1). Par ailleurs, afin de pouvoir éventuellement utiliser le processeur facette en monoprocesseur, un bit de commande externe (appelé MODE) permet, quand il est positionné, de désactiver l'opérateur Zbuffer et ainsi de ne tenir compte que de l'objet interne au processeur. La logique de sélection de l'objet visible est la suivante :

en mode pipeline

- si les deux objets sont présents, le plus proche est sélectionné et le bit de contour associé est positionné à 1.
- si un seul des deux objets est présent, il est sélectionné (quel que soit le résultat de la comparaison des profondeurs) et le bit de contour associé est positionné à 1.
- si aucun des objets n'est présent, le bit de contour de l'objet transmis est forçé à 0.

en mode monoprocesseur

- l'objet transmis est toujours l'objet interne. Le bit de contour est celui de l'objet interne.

La comparaison entre Z1 et Z2 est effectuée en calculant Z1-Z2 (i.e $Z1+\overline{Z2}+1$) et en testant le bit de signe du résultat. Toujours pour des problème de vitesse des additionneurs, l'addition 24 bits est pipelinée en deux additions 12 bits : les poids faibles sont d'abord comparés, puis les poids forts. Une bascule permet de mémoriser la retenue produite par la première

addition. Le fait de pipeliner ainsi l'opérateur Zbuffer oblige bien sûr à ajouter des registres de temporisation sur tous les chemins de données internes du processeur.

La Figure 4.16 présente l'organisation du Zbuffer. Pour simplifier le schéma, seule la composante Nx de la normale y figure. Bien évidemment, la sélection de l'objet visible s'opère également sur les composantes Ny et Nz, ainsi que sur la couleur de l'objet.

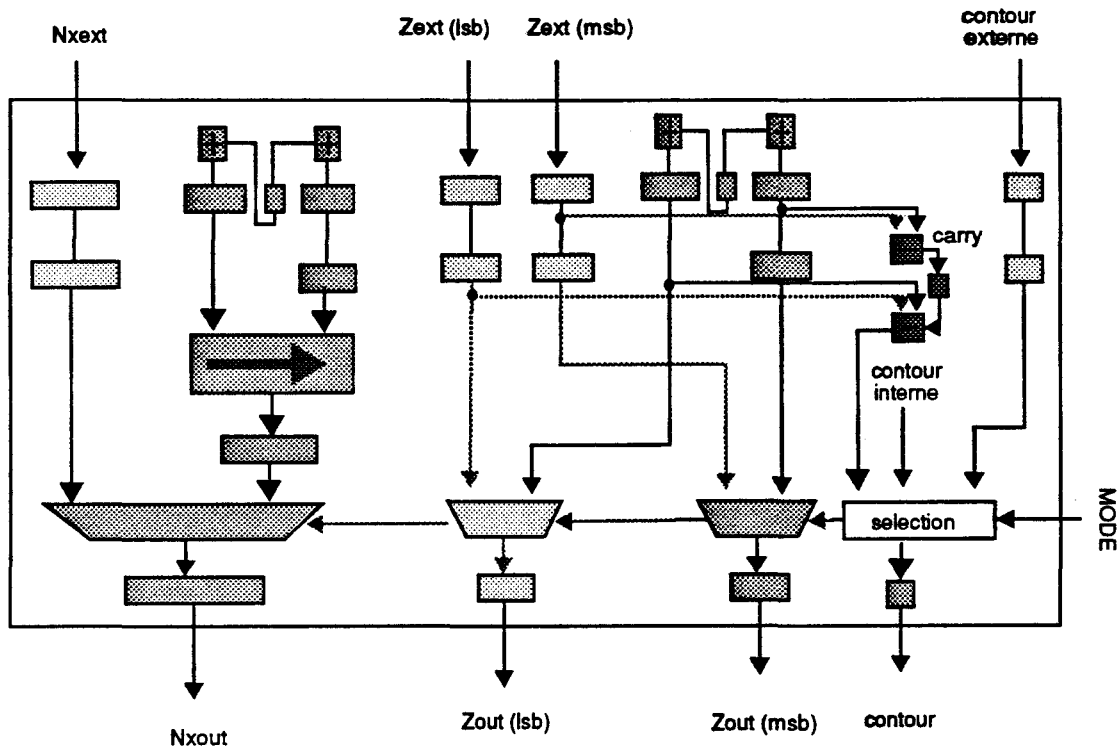


Figure 4.16 : l'élimination des parties cachées

Comme on peut le constater sur la figure précédente, la longueur du pipeline interne du processeur est de trois étages. Cependant, la présence du pré-normalisateur oblige à ajouter un étage de pipeline sur le chemin de données des normales internes (quatre étages). Pour synchroniser le calcul des normales et ceux du contour et de la profondeur, il est alors nécessaire de commencer le calcul des normales un cycle avant celui des autres valeurs.

Le programme des trois PE1 calculant la normale est donc différent de celui des quatre autres PE1. Afin d'uniformiser le contrôle des PE1, un bit de validation supplémentaire permet de désactiver les signaux d'horloge des registres des PE1 de contour et de profondeur. Le programme complet de contrôle des PE1 est donc le suivant (les instructions en gras sont celles permettant la synchronisation)

```
Pour chaque image Faire
  Select = 1
  Loading_RAM[0..2] -> Backup_RAM[0..2]  /* transfert des coeff */
  Select = 0
  Pour chaque ligne Faire
    Clear_Registres
    Read_backupRAM_adress2       /* lire bY+c */
    Clock1, Clock2               /* charger bY+c */
    Read_backupRAM_adress1       /* lire b */
    Clock1
    Clock2                        /* calculnouveau bY+c */
    Write_backupRAM_adress2       /* stocker bY+c */
    Read_backupRAM_adress0        /* lire a */
    Clock1                        /* amorcage du pipeline */
    Clock1, Clock2, normale       /* démarage du calcul des normales */
    Pour chaque pixel Faire
      Clock1, Clock2              /* calcul aX+bY+c */
```

4.3.3 Contrôle du processeur

4.3.3.1 Adressage du processeur par le hôte

Comme nous l'avons vu précédemment, les RAM de chargement des sept PE1 sont contrôlées la majeure partie du temps par le processeur hôte chargé du calcul et du transfert des coefficients. Deux solutions sont envisageables pour effectuer ce transfert :

- les coefficients sont transférés à chaque début d'image via le pipeline, en utilisant les chemins de données de profondeur et de normale. Cette solution est implémentée dans les machines GSP-NVS et PROOF. Son principal avantage est de réduire au maximum le nombre de broches d'Entrées/Sorties du processeur. Toutefois, le pipeline ne peut être utilisé pour la conversion lors du chargement des coefficients (voir chapitre 3)
- les coefficients sont chargés pendant la conversion, en utilisant un chemin d'accés indépendant. Cette solution permet d'effectuer en parallèle les phases de conversion et de chargement, et donc de réduire le temps total de génération d'une image.

Cette deuxième solution a été retenue lors de la conception du processeur facette. Par ailleurs, afin de simplifier l'interface entre le hôte et notre processeur, les RAM de chargement des PE1 sont directement connectées sur le bus du processeur hôte. Pour celui-ci, le chargement d'un coefficient est donc uniquement une opération d'écriture en mémoire (une zone de l'espace d'adressage du hôte est réservée à l'adressage des processeurs objets). L'adressage utilisé est le suivant :

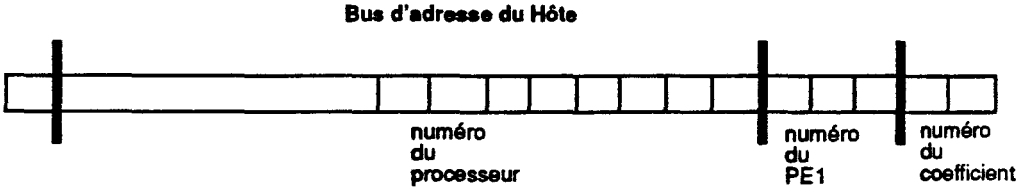


Figure 4.17 : adressage du processeur

Le numéro du processeur est analysé par un circuit de décodage externe générant pour chaque processeur du pipeline un signal de validation lui indiquant si le coefficient à transférer lui est destiné ou non. Les signaux provenant du processeur hôte et destinés au processeur facette sont donc les suivants :

- data (sur 24 bits) : bus de données du processeur hôte.
- e_hôte : signal d'écriture.
- validation (signalant que le coefficient est destiné au processeur).
- ad_hôte : adresse (les 5 bits de poids faible du bus d'adresse du hôte) indiquant le numéro du PE1 et le numéro du coefficient.

Le registre mémorisant la couleur de l'objet est également connecté sur le bus du processeur hôte. Afin de rester compatible avec le système d'adressage des PE1, il sera considéré par le hôte comme un huitième PE1 (numéro du PE1 = 111) dont le numéro du coefficient n'a pas à être précisé.

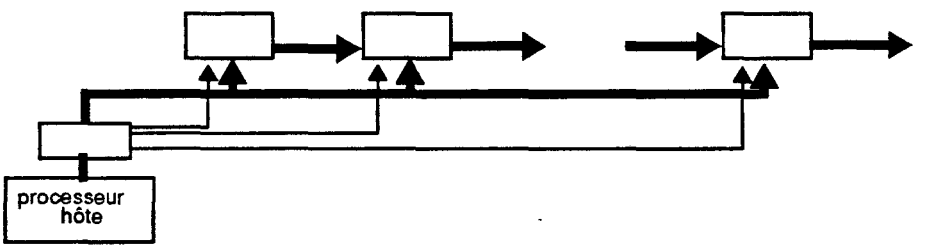


Figure 4.18 : liaison avec le hôte

4.3.3.2 Gestion des RAM des PE1

Le contrôle des RAM de chargement des PE1 se divise en deux phases :

- Phase 1 : à chaque début d'image (ou à chaque début de zone de conversion), les RAM de chargement sont controlées par le micro-contrôleur gérant le pipeline afin d'effectuer le transfert des coefficients des RAM de chargement vers les RAM de travail. Ce transfert est effectué simultanément par les sept PE1 du processeur facette, coefficient par coefficient. Pendant cette phase, les RAM de chargement sont en mode Lecture.

- Phase 2 : le reste du temps, les RAM de chargement sont controlées par l'ordinateur hôte s'occupant du calcul des coefficients de l'image suivante. Les RAM de chargement sont alors adressées directement dans l'espace mémoire du hôte, et sont chargées une à une, coefficient par coefficient via le bus du hôte. Pendant cette phase, les RAM de chargement sont en mode Ecriture.

Une logique de sélection permet de générer les signaux de contrôle des RAM de chargement des sept PE1 (adresse dans la RAM, signal de lecture et signal d'écriture), ainsi que les deux signaux de contrôle des registres de couleur :

- Pendant la phase 1, l'adresse et le signal de lecture proviennent du contrôleur et sont repercutés directement vers les sept PE1.
- Pendant la phase 2, l'adresse et le signal d'écriture proviennent du processeur hôte. L'adresse est repercutée vers les sept PE1, mais seul le signal d'écriture du PE1 (ou du registre) désigné par le hôte est activé.

La Figure 4.19 présente le schéma de contrôle des RAM de chargement.

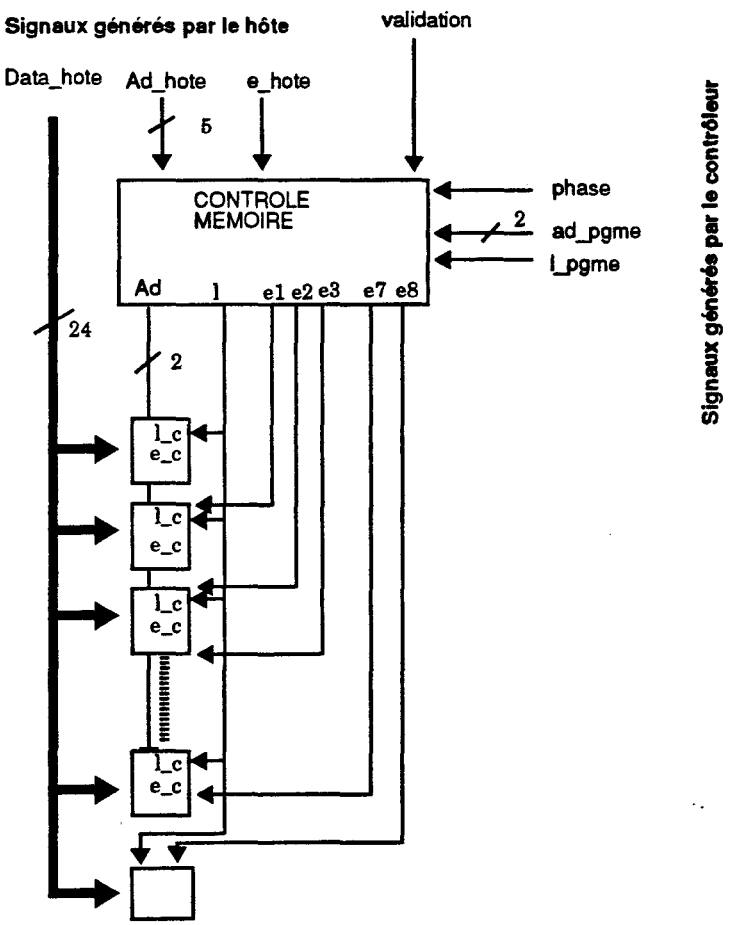


Figure 4.19 : génération des signaux de contôle des RAM de chargement

4.3.3.3 Commande du processeur

Comme nous l'avons signalé dans l'introduction, le pipeline de conversion est contrôlé par une unité de commande connectée au premier processeur du pipeline, et exécutant l'algorithme de fonctionnement des PE1. Cette unité peut être réalisée à l'aide d'un micro-contrôleur classique ou d'un automate cablé.

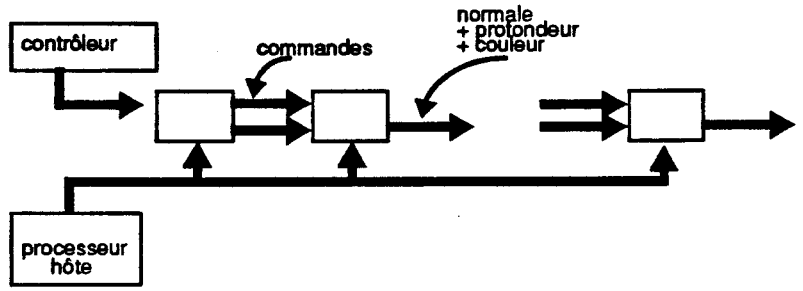


Figure 4.20 : contrôle du pipeline

La structure du mot de commande est la suivante :

synct	syncl	phase	norm	ad1	ad0	l_c	e_t	l_t	clear	ck2	ck1

- **ck1, ck2, clear** : signaux de contrôle des registres des PE1
- **l_t, e_t** : signaux de lecture et d'écriture des RAM de travail
- **l_c** : signal de lecture des RAM de chargement
- **ad** : adresse (soit dans la RAM de travail, soit dans la RAM de chargement)
- **norm** : signal d'inhibition des commandes des PE1 calculant le contour et la profondeur
- **phase** : signal indiquant qui contrôle les RAM de chargement
- **synct, syncl** : signaux de synchronisation.

La longueur du pipeline interne du processeur facette étant de trois étages, trois registres sont nécessaires pour transmettre la commande. Le premier sert à la mémorisation proprement dite et à l'envoi des signaux de contrôle vers les PE1, les deux autres permettent de synchroniser la transmission de la commande d'un processeur facette vers son successeur dans le pipeline. Ces trois registres sont pilotés par l'horloge système (commune à tous les processeurs) assurant le cadencement du pipeline.

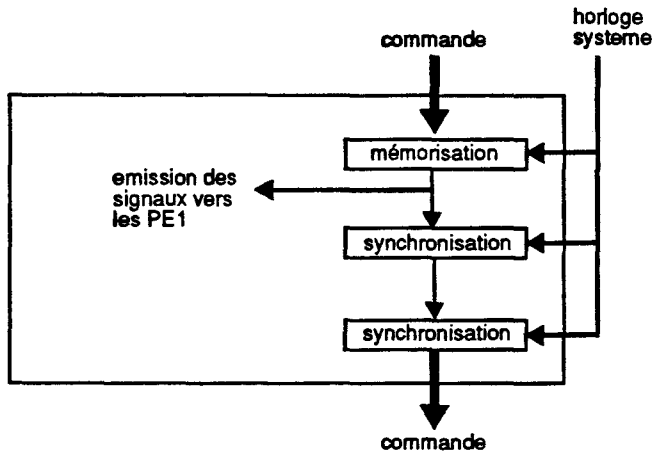
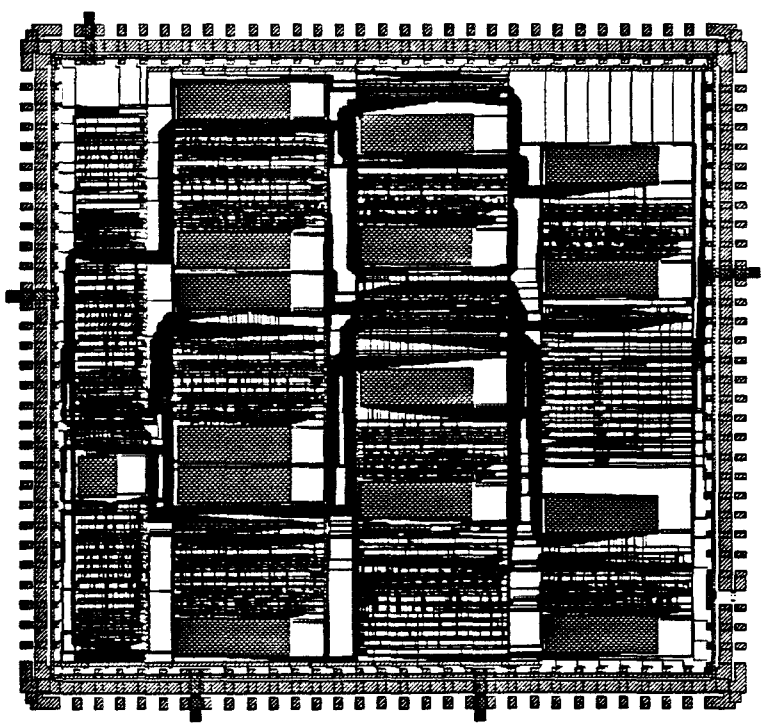
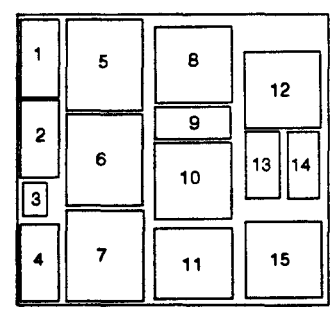


Figure 4.21 : le pipeline de commande interne

4.3.4 Conclusion

Le processeur que nous venons de décrire a été conçu et testé sous SOLO 1400. Le circuit, réalisé en technologie CMOS 1.5 microns, mesure $8,4 \times 7,86 mm$ et intègre environ 40.000 transistors. Il est encapsulé dans un boitier PGA 120 broches. Les simulations sous SOLO 1400 ont validé le circuit pour une fréquence de fonctionnement de 16 MHz. Les échantillons que nous a livré la fonderie (via le CMP) doivent encore être testés.





1,2,4: pre-normalisateurs
3: contrôle pre-normalisateurs
8, 10, 12: PE1s (contour)
9: buffers
5,6,7: PE1s (vecteur normal)
15: PE1 (Z)
11: Zbuffer
13: couleur de base
14: unité de commande

Figure 4.22 : masque VLSI du processeur facette

4.4 Cellule de base d'un réseau multi-pipeline

Contrairement au processeur facette, le circuit de la cellule de base du réseau multi-pipeline n'a pas encore était réalisé. Nous présentons cependant dans la suite son principe général de fonctionnement, et quelques idées pouvant faciliter une intégration VLSI.

4.4.1 Principe de base

Le principe de la cellule de base du pipeline horizontal d'un réseau délivrant une facette par cycle (sept expressions générées en parallèle) est illustré sur la Figure 4.23.

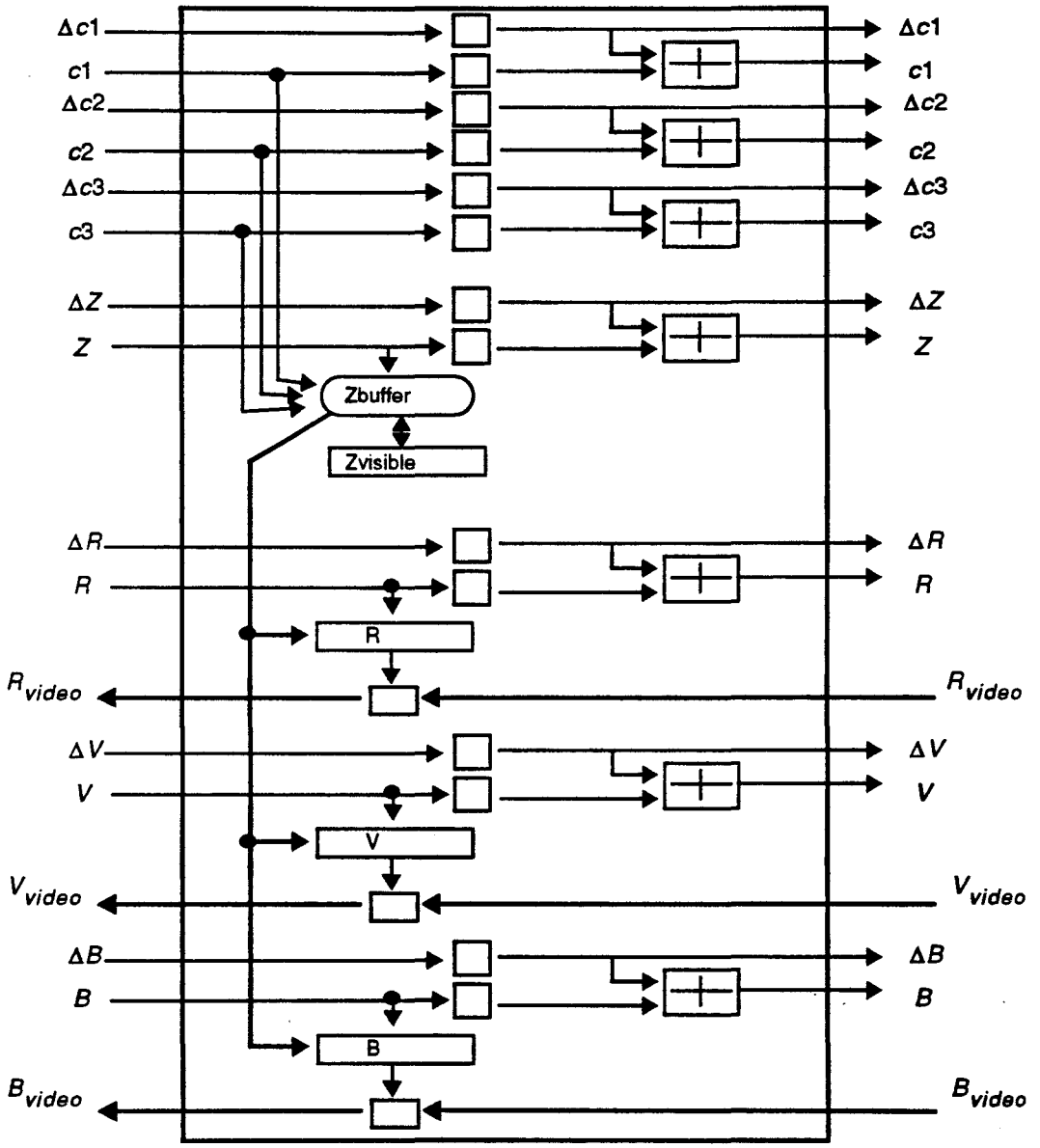


Figure 4.23 : schéma général de la cellule

4.4.2 Vers une réalisation VLSI

Il serait intéressant d'intègrer une ligne de 32 processeurs pixels (plus la cellule du pipeline vertical) dans un seul circuit. Le réseau 32 x 32 nécessiterait alors 32 circuits. Le circuit de base possède alors le brochage suivant :

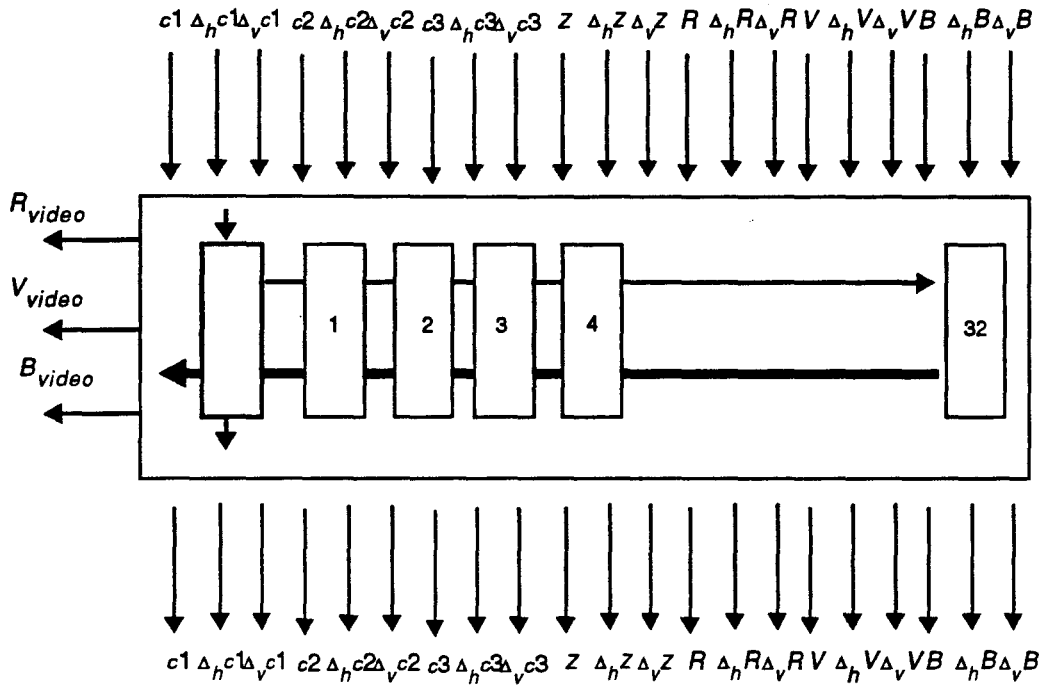


Figure 4.24 : circuit de base

Si on suppose que toutes les valeurs sont transmises sur 16 bits (sauf les valeurs video sur 8 bits), le circuit nécessite alors environ 700 signaux d'entrée/sortie. Si l'on veut réduire le nombre d'entrées/sorties, la solution la plus simple est de sérialiser le transfert des données sur les liaisons.

Dans le chapitre 3, nous avons pris comme exemple de référence un multi-pipeline calculant une expression (sur toute sa largeur) à la fois. En fait, il est beaucoup plus intéressant de calculer les sept expressions en parallèle, mais sur peu de bits de largeur (par exemple 4 bits seulement). Une telle technique permet d'utiliser des additionneurs très simples (4 bits), et donc très rapides, ce qui permet facilement d'utiliser des fréquences d'horloge élevées (même avec un logiciel de CAO moyennement performant).

A titre d'exemple, si toutes les valeurs sont transmises sur quatre bits (excepté les sorties vidéo), le circuit ne doit plus utiliser qu'environ 200 broches. Un réseau composé de 32 de ces circuits générerait alors une facette tous les quatre cycles (tous les huit cycles si l'on calcule sur 32 bits). D'après les calculs menés dans le chapitre précédant, un tel système délivrerait une puissance d'environ 3 millions de facettes par seconde (33 MHz, boîtes englobantes 20 x 20).

Chapitre 5

Affichage de primitives de haut niveau

Comme nous l'avons détaillé dans les chapitres précédents, toutes les stations graphiques actuelles utilisent comme primitives de visualisation les facettes triangulaires. Ces facettes sont généralement utilisées pour approcher les surfaces plus complexes utilisées en modélisation (quadriques, Béziers, NURBS...). Par ailleurs, certaines méthodes de calcul d'images comme la radiosité s'appuient sur une discrétisation de la scène en petites facettes afin de résoudre les équations de conservation de l'énergie lumineuse.

En marge des stations graphiques commerciales, plusieurs projets de recherche ont abouti à des machines capables d'afficher directement des primitives graphiques de plus haut niveau que les facettes. Ces projets sont restés à l'état de prototypes, mais il n'est pas interdit d'envisager dans les prochaines années la commercialisation de machines tirant profit de ces résultats.

Nous présentons dans la suite les projets les plus significatifs (certains ont été décrits en détail dans le chapitre 2). Nous rappelons brièvement l'architecture de ces machines, puis les méthodes utilisées pour l'affichage des primitives graphiques de haut niveau.

Nous présentons ensuite les travaux menés depuis plusieurs années par notre équipe sur la conception d'un processeur quadrique destiné à être integré dans une architecture à parallélisme objet. Ces travaux feront l'objet de la thèse d'E. Nyiri, aussi nous nous bornerons ici à une présentation générale des différentes architectures possibles.

5.1 Etat de l'art

5.1.1 Pixel-Planes 4

Pixel-Planes 4 (voir paragraphe 2.2.1.1) est une machine SIMD massivement parallèle possédant un processeur par pixel de l'écran (i.e. 512 x 512 processeurs). Chaque processeur est composé d'une ALU 1 bit et d'une zone mémoire. La grille de processeurs est reliée à l'ordinateur hôte via un arbre d'additionneurs permettant à tous les processeurs de calculer simultanément une expression du premier degré aX+bY+C, (X,Y) étant les coordonnées du processeur-pixel, et (a,b,c) des coefficients fournis par le hôte.

Tous les algorithmes de rendu de primitives développés pour Pixel-Planes 4 se décomposent en trois étapes :

- détermination du contour de la primitive. A l'issue de cette étape, chaque processeur sait si le pixel dont il s'occupe est à l'intérieur ou à l'extérieur du contour de la primitive.
- élimination des parties cachées en tout pixel où l'objet est présent (algorithme du Z-buffer).
- calcul des valeurs RVB en tout pixel où l'objet est présent et visible.

Nous présentons dans la suite une mise en oeuvre de cette méthode dans le cas des polygones convexes, puis dans le cas de primitives de plus haut niveau telles que les sphères.

5.1.1.1 Génération de polygones convexes

Les trois étapes de la génération d'un polygone convexe sont les suivantes :

- Le contour du polygone à n côtés est déterminé par l'intersection de n demi-plans. Les équations des n demi-plans délimitant la facette sont envoyées aux processeurs-pixel qui effectuent grâce à leurs ALUs une combinaison des signes des expressions. Au terme de cette étape, chaque processeur sait si le pixel dont il s'occupe est à l'intérieur ou à l'extérieur du contour de la facette.
- L'équation du plan support de la facette est envoyée aux processeurs-pixel. Les processeurs pour lesquels la facette est présente effectuent l'élimination des parties cachées en comparant la profondeur de la facette visible à celle de la facette en cours de traitement.
- Les trois équations permettant d'interpoler les valeurs RVB sont envoyés aux processeurs-pixel. Seuls les processeurs pour lesquels la facette est présente et visible stockent la nouvelle valeur RVB dans leur zone mémoire.

5.1.1.2 Génération de sphères.

Pixel-Planes 4 ne peut calculer que des expressions du premier degré, et ne semble donc pas adaptée à l'affichage direct de sphères, celui-ci nécessitant le calcul d'expressions du second degré. L'équipe de Pixel-Planes a pourtant développé pour leur machine un algorithme efficace de rendu de sphères :

- L'équation d'un cercle de centre (a, b) et de rayon r peut s'exprimer sous la forme $g(x, y) = Ax + By + C - Q = 0$ avec $A = 2a$, $B = 2b$, $C = r^2 - a^2 - b^2$ et $Q = x^2 + y^2$. L'expression Q est pré-chargée par le hôte dans la mémoire de chaque processeur-pixel. Puis chaque processeur calcule $Ax + By + C$, et soustrait la valeur de Q présente dans sa mémoire. Tous les pixels pour lesquels la valeur obtenue est négative sont situés hors du contour de la sphère.
- La profondeur de la sphère au pixel (x,y) s'exprime par $z = c - \sqrt{r^2 - (x-a)^2 - (y-b)^2}$ avec r le rayon et (a, b, c) le centre de la sphère. Cette expression peut être approchée par $z = c - (r^2 - (x-a)^2 - (y-b)^2)/r$. Ceci revient à approcher la demi sphère visible par un paraboloïde. L'élimination des parties cachées peut alors être effectuée en utilisant la même méthode de pré-chargement des termes du second degré que pour le calcul du contour. La Figure 5.1 présente le paraboloïde utilisé (tangent au sommet de la sphère et passant par les pôles).

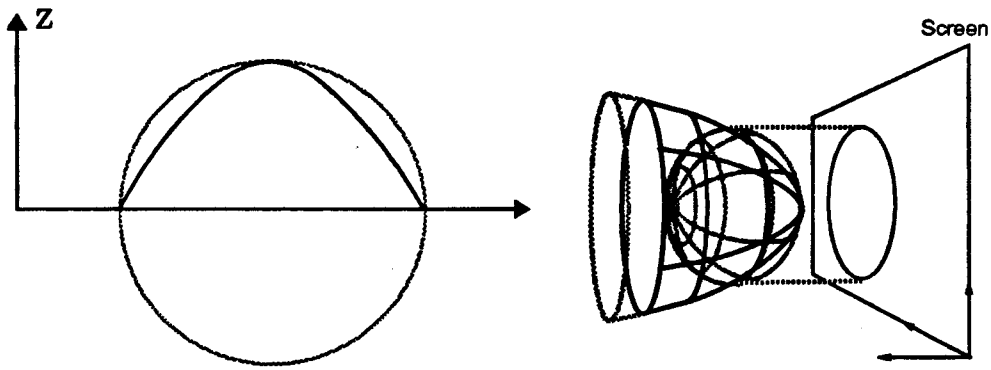


Figure 5.1 : approximation de la profondeur de la demi-sphère avant

- Le calcul des valeurs RVB s'effectue également par approximation des formules exactes (prise en compte uniquement de l'éclairement ambiant et diffus pour des sources à l'infini).

5.1.2 Pixel-Planes 5

La principale amélioration apportée sur Pixel-Planes 5 (voir paragraphe 2.4.2) facilitant l'affichage de primitives de haut niveau est l'évaluation direct en chaque pixel d'une expression quadratique et non plus linéaire. D'autres modifications architecturales de plus haut niveau ont également été apportées (voir chapitre 2), mais celles-ci n'interviennent pas fondamentalement sur les algorithmes de rendu. Dans la suite, et afin de simplifier la description des algorithmes, nous considérerons que Pixel-Planes 5 est une machine SIMD possédant un processeur par pixel, tous les processeurs pouvant évaluer simultanément pour leur pixel (x, y) une expression quadratique $Q(x, y) = Ax^2 + By^2 + Cxy + Dx + Ey + F$.

5.1.2.1 Génération de sphères

La méthode de génération de sphères est semblable à celle utilisée sur Pixel-Planes 4, mais bénéficie pleinement des possibilités de calcul d'expressions du second degré.

Le contour de la sphère, ainsi que la profondeur du paraboloide approchant la demi-sphère visible se calculent directement par des expressions du second degré (et non plus par pré-chargement des valeurs quadratiques). Le calcul des valeurs RVB s'effectue par l'algorithme de Phong et nécessite donc de calculer le vecteur normal $\vec{N}(Nx, Ny, Nz)$. Les coordonnées du vecteur normal à la sphère de centre (a, b, c) et de rayon r sont

$$Nx = \frac{x-a}{r} \qquad Ny = \frac{y-b}{r} \qquad Nz = \frac{\sqrt{r^2-(x-a)^2-(y-b)^2}}{r^2}$$

La composante Nz, qui ne peut être calculée directement, est approchée par

$$Nz = \frac{r^2 - (x-a)^2 - (y-b)^2}{r^2}$$

Les trois composantes peuvent donc être calculées directement en chaque pixel. L'éclairement est ensuite calculé (formule de Phong avec prise en compte du spéculaire) par développement limité en utilisant les évaluateurs quadratiques. Le calcul de l'éclairement de Phong avec une seule source lumineuse nécessite 23000 cycles.

5.1.2.2 Génération de cylindres

L'équipe de Fuchs a développé un algorithme efficace pour afficher directement des cylindres sur Pixel-Planes 5. La méthode est la suivante :

- Le contour écran du cylindre est composé de droites et d'ellipses. Il nécessite donc uniquement l'évaluation de quatre expressions quadratiques. Soient $L1(x,y)$, $L2(x,y)$, $L3(x,y)$, $L4(x,y)$ les équations des demi-plans délimitant le cylindre et $Q1(x,y)$, $Q2(x,y)$ les équations des deux ellipses formant les extrémités du cylindre (voir Figure 5.2). Les équations linéaires sont combinées en deux équations quadratiques $Q3(x,y) = L1(x,y)\,L2(x,y)$ et $Q4(x,y) = L3(x,y)\,L4(x,y)$. Un point $P(x,y)$ appartient au contour du cylindre si et seulement si

$$(Q3(x,y) > 0)\ et\ (Q4(x,y) > 0)\ et\ (Q2(x,y) > 0)$$
$$ou$$
$$Q1(x,y) < 0$$

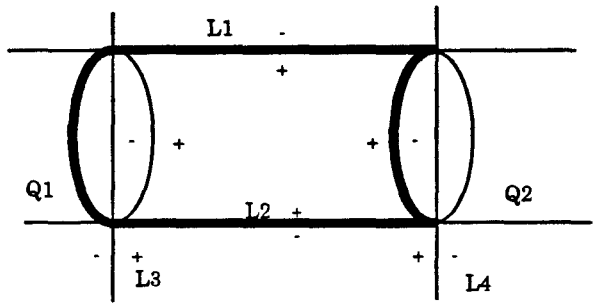


Figure 5.2 : contour du cylindre

- L'équation de la partie frontale du cylindre peut s'exprimer sous la forme $z = L - \sqrt{Q}$, L étant une expression linéaire et Q une expression quadratique. Pixel-Planes 5 ne pouvant pas effectuer de calcul de racine carrée, la profondeur du cylindre est approchée par $z = L - s - tQ$, s et t étant des constantes. Cela revient à approcher le cylindre par un cylindre parabolique (voir Figure 5.3).

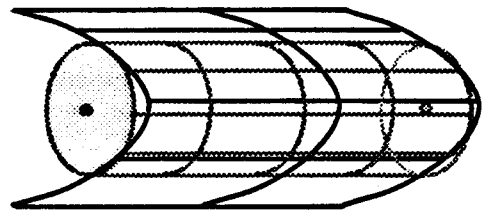


Figure 5.3 : cylindre parabolique

Afin d'améliorer la précision de l'approximation, plusieurs morceaux de cylindres paraboliques sont utilisés pour modéliser la surface cylindrique. La Figure 5.4 donne un exemple d'une telle approximation avec trois cylindres paraboliques. Selon la précision recherchée, deux à huit expressions quadratiques peuvent être utilisées. Par ailleurs, certaines applications comme l'affichage d'arbres CSG nécessitent de calculer pour chaque primitive les profondeurs avant et arrière. Dans ce cas, plusieurs cylindres paraboliques sont également utilisés pour approcher la profondeur de la partie dorsale du cylindre

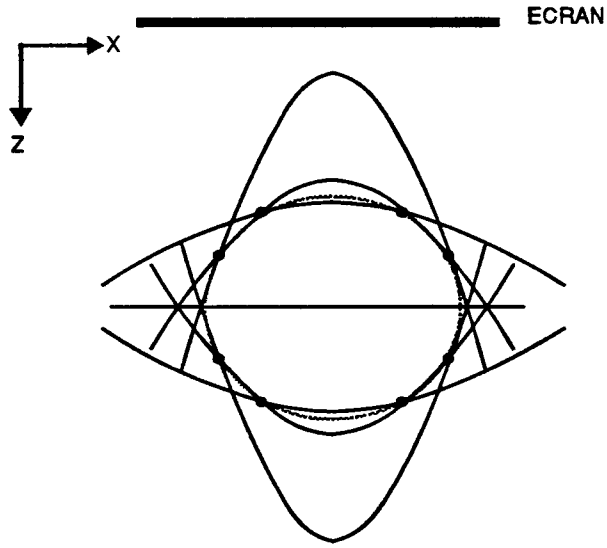


Figure 5.4 : approximation de la profondeur du cylindre par plusieurs cylindres paraboliques

- Le calcul des valeurs RVB est également obtenu en approchant une expression contenant une racine carrée par une expression quadratique. Notons qu'une unique expression quadratique est utilisée pour calculer l'éclairement sur l'ensemble du cylindre, contrairement à la profondeur où plusieurs cylindres paraboliques sont utilisés.

5.1.3 L'Apollo DN 10000

La DN 10000 [KIRK90] est une station graphique commerciale à l'architecture relativement

classique. La description complète de cette machine dépasse le cadre de notre présentation. Nous tenons uniquement à signaler ici que l'affichage de facettes est réalisé à l'aide de cinq interpolateurs quadratiques travaillant en parallèle et calculant les valeurs R, V, B, Z et alpha. Les interpolateurs quadratiques ne sont pas utilisés pour visualiser des primitives de haut niveau (comme Pixel-Planes 5), mais pour améliorer la qualité du rendu des facettes 3D lors des calculs de textures et d'éclairement :

- interpolation quadratique sur les couleurs permettant de simuler un éclairement de Phong (algorithme de Bishop et Weiner [BISH86]).
- interpolation quadratique des paramètres u et v lors du mapping de textures. Pour conserver une perspective correcte lors de l'application de textures, les paramètres u et v (coordonnées dans la table de texture) s'expriment par $u = (Ax+By+C)/(Dx+Ey+F)$ et $v = (Gx+Hy+I)/(Dx+Ey+F)$. L'interpolation quadratique permet une meilleure approximation de ces fonctions que les classiques interpolations linéaires.

L'Apollo DN10000 est à notre connaissance la seule station commerciale à avoir proposé l'utilisation d'interpolateurs quadratiques pour améliorer la qualité du rendu. Aucun essai de rendu de primitives de haut niveau n'a toutefois été tenté sur cette machine.

Notons que de nos jours, le développement des ASICs permet d'intégrer des diviseurs cablés dans les systèmes graphiques haut de gamme, permettant ainsi la mise en perspective exacte (et non plus approchée) des textures (par exemple sur la SGI Reality Engine [SILI92]).

5.1.4 La Ray-Casting Machine

La Ray-Casting Machine (maintenant appelé Ray-Casting Engine) est une machine spécifiquement concue pour afficher des images CSG d'objets quadriques (voir paragraphe 2.2.2.5).

Elle est principalement composée de deux types de processeurs :

- Les processeurs objets appelés PC (Primitive Classifier). Chaque PC permet de calculer en tout point de l'écran les profondeurs avant et arrière de l'objet dont il a la charge. Les objets utilisés par le Ray-Casting Engine sont les quadriques.
- Les processeurs de combinaison appelés CC (Combine Processor) permettant de réaliser les opérations logiques CSG (union, intersection et différence) entre les primitives.

Contrairement à la machine Pixel-Planes 5, où l'affichage de primitives graphiques de haut niveau est effectué par programmation des ressources de la machine, les processeurs PC de la Ray-Casting Machine sont spécifiquement cablés pour calculer la profondeur d'une quadrique.

La résolution en z de l'équation d'une quadrique donne (les détails de la résolution seront donnés au paragraphe)

$z1 = L(x,y) + \sqrt{Q(x,y)}$ et $z2 = L(x,y) - \sqrt{Q(x,y)}$, avec L expression linéaire et Q expression quadratique.

Un processeur PC est donc composé d'une unité de calcul linéaire, d'une unité de calcul quadratique, d'une unité de calcul de racine carrée, d'un additionneur et d'un soustracteur. Afin de diminuer la complexité matérielle du processeur, tous les opérateurs ne calculent qu'un bit à la fois (bien évidemment au détriment de la vitesse d'exécution).

Le Ray-Casting Engine est à notre connaissance le seul système spécifiquement cablé pour

afficher des quadriques. La combinaison des quadriques et de la représentation CSG font de cette machine un remarquable outil pour le domaine de la CAO. A notre connaissance, une industrialisation de cette machine est envisagée dans un avenir proche.

5.1.5 Conclusion

Ce rapide tour d'horizon montre que des efforts sérieux sont entrepris depuis quelques années pour tenter de détrôner la reine facette. Dans le cadre de notre étude sur les machines à parallélisme objet, nous avons étudié la possibilité d'intègrer un processeur quadrique dans un pipeline objet. La description de ce processeur est présentée dans la suite de ce chapitre.

5.2 Opérateurs incrémentaux pour le calcul de quadriques

5.2.1 Définitions mathématiques

Les surfaces quadriques sont définies par une équation de degré 2 en (x, y, z). La forme générale de cette équation est

$$Q(x, y, z) = ax^2 + by^2 + cz^2 + dxy + eyz + fxz + gx + hy + iz + j = 0$$

Notre but est de proposer un processeur quadrique pouvant être utilisé dans un système à parallélisme objet possèdant un post-processeur d'éclairement. Les valeurs à calculer sont donc la profondeur de la quadrique en tout pixel, ainsi que son vecteur normal. Par ailleurs, afin de pouvoir effectuer l'élimination des parties cachées dans le pipeline objet, ces valeurs doivent nécessairement être calculées dans l'ordre du balayage écran. La seule méthode envisageable pour réaliser le processeur quadrique est donc de résoudre en z l'équation de la quadrique. Celle-ci peut alors se réécrire sous la forme

$$Q(z) = cz^2 + (ey + fx + i)z + ax^2 + by^2 + dxy + gx + hy + j$$

Dans le cas général (i.e si le discriminant est positif), cette équation possède deux solutions données par

$$z1 = l1(x, y) + \sqrt{q1(x, y)} \text{ et } z2 = l1(x, y) - \sqrt{q1(x, y)}$$

avec $l1(x, y)$ fonction linéaire en (x, y) et $q1(x, y)$ fonction quadratique en (x, y).

Par ailleurs, le vecteur normal à la quadrique est obtenu en dérivant l'équation initiale par rapport aux trois variables. On a ainsi

$$N_x = \frac{\partial}{\partial x} Q(x, y, z)$$
$$N_y = \frac{\partial}{\partial y} Q(x, y, z)$$
$$N_z = \frac{\partial}{\partial z} Q(x, y, z)$$

On obtient alors

$$N_x = l2(x, y) + \alpha z = l3(x, y) \pm \sqrt{q3(x, y)}$$
$$N_y = l4(x, y) + \beta z = l5(x, y) \pm \sqrt{q5(x, y)}$$
$$N_z = l6(x, y) + \Gamma z = l7(x, y) \pm \sqrt{q7(x, y)}$$

Il apparait donc que l'affichage de quadrique requiert trois opérateurs distincts : un opérateur de calcul d'expressions linéaires, un opérateur de calcul d'expressions quadratiques, et un opérateur de calcul de racine carrée. Le calcul incrémental d'expressions linéaires a été étudié dans le chapitre précédent dans le cadre du processeur facette (opérateur PE1). Nous présentons donc maintenant un opérateur quadratique (appelé PE2) basé sur le même principe (calcul incrémental), puis un opérateur de calcul de racine carrée. Rappelons que ces travaux constituent une pré-étude avant une éventuelle intégration d'un processeur quadrique (d'où la nécessité de disposer de ces opérateurs). Il serait bien évidemment possible d'utiliser des processeurs généraux (RISC ou DSP[1]) pour effectuer ces calculs, mais au prix d'une diminution importante des performances (les opérateurs que nous définissons sont tous capables de fournir une valeur par cycle élémentaire).

5.2.2 Calcul d'une expression du second degré

Le Processeur Elémentaire du Second Degré, appelé PE2 [KARP89], calcule incrémentalement dans l'ordre du balayage écran une expression quadratique de la forme $Q(x,y) = Ax^2 + By^2 + Cxy + Dx + Ey + F$.

En isolant les termes en y, on obtient $Q(x,y) = Ax^2 + (Cy + D)x + (By^2 + Ey + F)$. Le calcul incrémental de $Q(x,y)$ peut donc se décomposer en deux étapes :

- à chaque début de ligne, calcul de $L(y) = Cy + D$ et de $Q'(y) = By^2 + Ey + F$
- pour chaque pixel de la ligne (y constant), calcul de $Q(x) = Ax^2 + Lx + Q'$

Le calcul incrémental d'une expression quadratique d'une seule variable est effectué à l'aide d'un intégrateur à deux niveaux (Figure 5.5)

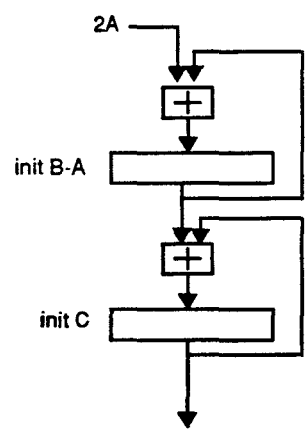


Figure 5.5 : calcul incrémental de Ax^2+Bx+C

Sous réserve de disposer de registres de sauvegarde, la même structure peut être utilisée pour le calcul de $L(y)$ et $Q'(y)$ en début de ligne, puis pour le calcul de $Q(x)$ en chaque pixel de la ligne. Afin de limiter la complexité du PE2 (tant au niveau des regitres que de la connectique), nous avons choisi d'utiliser une RAM de 6 mots (au lieu de six registres indépendants) pour mémoriser tous les calculs intermédiaires. Par ailleurs, une RAM duale

1. Digital Signal Processor (processeur de traitement de signal)

permetà l'hôte de charger une nouvelle série de coefficients pendant le calcul de l'expression quadratique (comme sur le PE1). Le schéma fonctionnel du PE2 est donc le suivant :

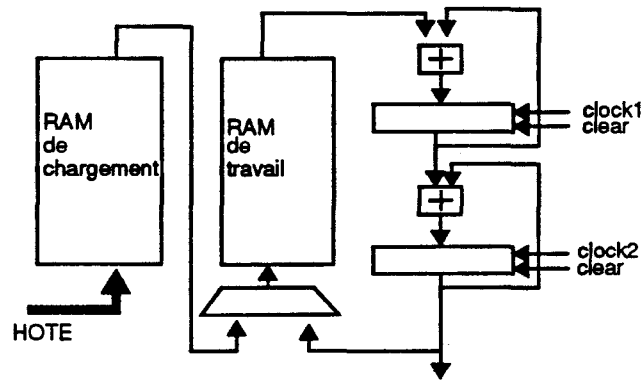


Figure 5.6 : schéma fonctionnel du PE2

Remarque

La vitesse de fonctionnement du PE2 est déterminée par le plus lent des éléments le constituant (RAM, registre et additionneur). Etant donnée la largeur de mots utilisée (48 bits), l'additionneur constitue, comme dans le cas du PE1, l'élement critique du processus de calcul. Si on ne désire pas utiliser d'additionneurs parallèles (étant donné leur fort coût silicium), il est possible, comme dans le cas du PE1, d'effectuer l'addition en pipeline à l'aide de plusieurs additionneurs série et de bascules mémorisant la retenue d'un additionneur à un autre.

5.2.3 Calcul de la racine carrée

Comme nous venons de le constater, le PE2 est une unité de calcul relativement simple et facile à intéger en VLSI. La principale difficulté pour réaliser un processeur quadrique réside dans l'extracteur de racine carrée. Nous avons étudié deux méthodes pour calculer la racine carrée : une méthode par approximation et une méthode par calcul exact.

5.2.3.1 Méthode par approximation

La méthode utilisée pour approcher la racine carrée s'inspire de celle mise au point par Hashemian, et améliorée par S. Degrande (une description précise des fondements mathématiques de l'amélioration apportée est fournie dans [DEGR93]).

Le calcul de la racine carrée se divise en deux étapes. Un premier calcul simple permet de déterminer une première approximation (erreur de 6% environ). Un processus de calcul itératif permet ensuite d'affiner le calcul (la précision du résultat dépend du nombre d'itérations).

La première étape d'approximation est la suivante :

- si X est codé sur 2m bits , alors $\sqrt{X} \sim \frac{X}{2^{m+1}} + 2^{m-1}$
- si X est codé sur 2m-1 bits, alors $\sqrt{X} \sim \frac{2X}{2^{m+1}} + 2^{m-1} - 2^{m-2}$

Ces expressions peuvent être simplifiées afin de faciliter une implémentation matérielle. En effet, les additions et soustractions intervenant dans les formules ne portent que sur un bit dont la valeur est toujours 0 du fait du décalage les précédant. Ces opérations peuvent donc être remplacées par des masques OU (pour l'addition) et ET (pour la soustraction).

La formule itérative de correction est ensuite la suivante ($A(X)$ étant l'expression calculée lors de l'étape d'approximation) :

$$\sqrt{X} - A(X) - \frac{((A(X))^2 - X)}{2^{m+1}}$$

Une approximation précise de la racine carrée est obtenue après quatre itérations. Cependant, des simulations ont montré que deux itérations suffisent pour approcher correctement la profondeur d'une quadrique (en particulier les intersections entre objets, témoins de la qualité du calcul, sont très proches des résultats obtenus par calcul exact de la racine carrée).

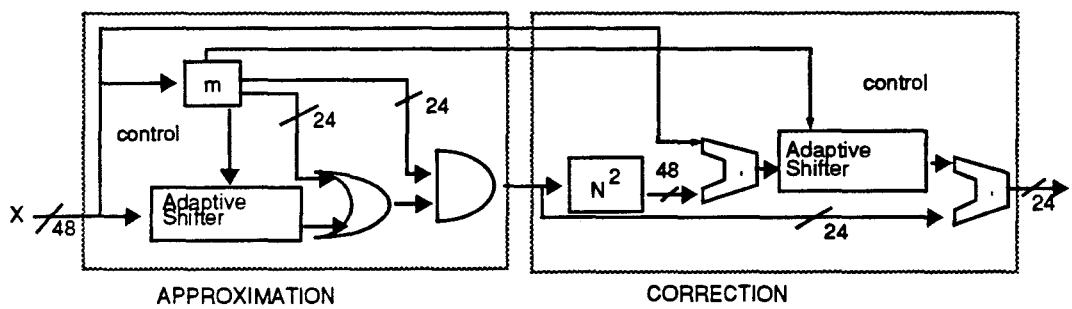


Figure 5.7 : approximation avec une étape de correction

Cette méthode semble séduisante puisque l'implémentation matérielle de l'étape d'approximation est très simple (un décaleur variable, une PLA pour générer les masques ET et OU, un masque ET et un masque OU). Malheureusement, l'étape de correction est beaucoup plus délicate à réaliser à cause de l'évaluation du carré. Par ailleurs, nos simulations ont montré que dans beaucoup de cas, deux étapes de correction sont nécessaires pour obtenir un résultat visuel correct. Etant données ces difficultés (en particulier la complexité de l'opérateur d'élévation au carré), nous nous sommes orientés vers un calcul exact de la racine carrée.

5.2.3.2 Métode de calcul exacte

Le calcul exact de la racine carrée s'appuie sur une méthode itérative (il s'agit d'effectuer une division dont le diviseur est également le résultat). Le lecteur pourra trouver une description précise de la méthode dans [LAPO91], nous nous bornerons ici à une description rapide de l'implémentation matérielle.

Le calcul est implémenté dans un réseau dont chaque ligne fournit un bit du résultat. La

cellule de base du réseau peut effectuer soit une soustraction, soit une sélection (du reste précédent ou du résultat de la soustraction). LaFigure 5.8 présente la cellule de base et le réseau construit à partir de celle-ci.

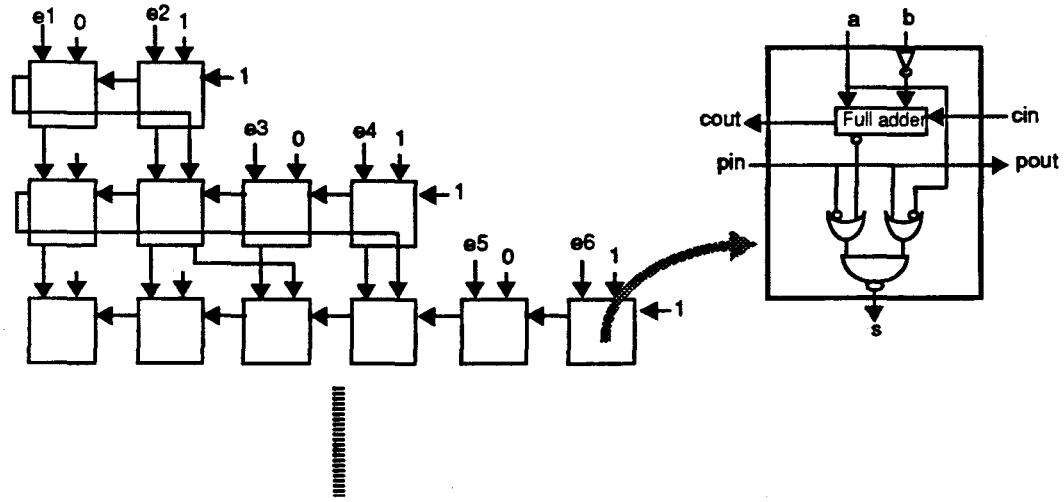


Figure 5.8 : réseau pour la racine carrée

Pour calculer une racine carrée sur 24 bits (i.e la racine carrée d'un nombre 48 bits), le réseau de calcul doit possèder 600 cellules. Cependant, la méthode de calcul retenue produit d'abord les bits de poids forts du résultat. Il est donc possible, si une moindre précision est suffisante (perte des poids faibles), d'utiliser un nombre plus réduit de cellules. A titre d'exemple, un réseau de 156 cellules suffit à calculer une racine sur 12 bits.

5.2.4 le Processeur Elémentaire Quadrique

Le Processeur Elémentaire Quadrique calcule une expression du type $z = f1(x, y) \pm \sqrt{f2(x, y)}$, où $f1(x, y)$ est une expression linéaire et $f2(x, y)$ une expression quadratique. Il est donc composé d'un PE1, d'un PE2, d'un extracteur de racine carrée et de deux additionneurs (voir Figure 5.9)

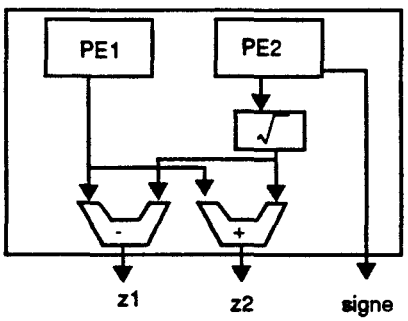


Figure 5.9 : le Processeur Elémentaire Quadrique

Le signe de $f2(x, y)$ sera utilisé pour déterminer le contour de la quadrique. En effet, quand $f2(x, y) < 0$, le pixel courant (x, y) est en dehors du contour de la quadrique. Le résultat fourni par l'opérateur de racine carrée n'a alors aucun sens et ne doit pas être pris en compte.

5.3 Première génération de quadriques

Exceptées la sphère et l'ellispoïde, les quadriques sont infinies dans une, deux ou trois directions. Afin de disposer de primitives graphiques finies, nous avons dans un premier temps choisi de délimiter les quadriques infinies par deux demi-plans perpendiculaires à l'axe principal. Cette méthode permet de générer aisément le contour écran des quadriques de révolution, mais en contrepartie ne permet pas d'afficher les quadriques infinies dans deux directions comme le paraboloïde hyperbolique

5.3.1 Génération du contour

La première méthode de rendu de quadriques que nous avons développée [KARP91] s'inspire directement de celle utilisée dans Pixel-Planes 5 pour approcher le cylindre. En effet, le contour de la quadrique peut être généré de la même facon que celui du cylindre, la différence résidant dans le fait que le corps du contour est une conique (et non plus deux demi-plans).

La Figure 5.10 présente un exemple de construction du contour dans le cas d'un hyperboloide à une nappe.

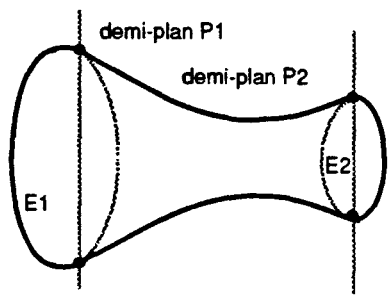


Figure 5.10 : contour écran d'une quadrique

Le calcul explicite du contour écran d'une quadrique nécessite donc les processeurs élémentaires suivant :

- deux PE1 pour déterminer les demi-plans P1 et P2.
- deux PE2 pour déterminer les deux ellipses E1 et E2.
- récupération du signe de l'expression quadratique intervenant dans le calcul du Z pour déterminer la partie conique du contour.

5.3.2 Calcul de la profondeur et de la normale

Le calcul de la profondeur en tout point de la quadrique nécessite les processeurs suivants :

- un PEQ pour déterminer les profondeurs avant (ZQav) et arrière (ZQar) de la quadrique infinie.

- deux PE1 pour déterminer les profondeurs des disques avant (ZE1) et arrière (ZE2).

Le calcul du vecteur normal nécessite quant à lui les processeurs suivants :

- trois PEQ pour calculer les composantes Nx, Ny et Nz du vecteur normal avant (NQav) et arrière (NQar) de la quadrique.
- les vecteurs normaux aux disques avant (NE1) et arrière (NE2) sont constants, et seront mémorisés dans des registres.

5.3.3 Construction d'une quadrique

Nous avons défini trois types d'objets quadriques : les objets positifs, les objets négatifs et les objets creux. Les objets négatifs sont utilisés lors de certaines opérations booléennes entre objets (dans le cas de l'opérateur différence du CSG par exemple).

La Figure 5.11 présente le contour écran d'un cône, ainsi que les trois interprétations possibles (positif, négatif, creux).

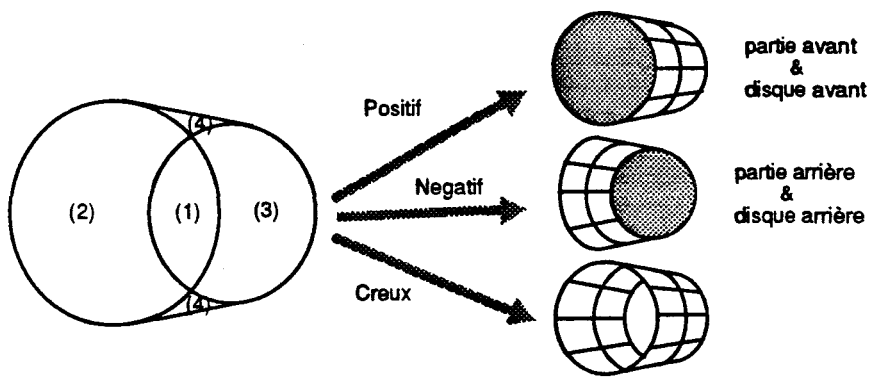


Figure 5.11 : les trois représentations d'une quadrique

Pour afficher la quadrique, il est nécessaire de déterminer en chaque point du contour la profondeur et le vecteur normal de l'objet, ces valeurs pouvant être soit celles de la quadrique, soit celles du disque avant, soit celles du disque arrière.

Pour cela, le contour écran de la quadrique est divisé en quatre zones disjointes (1), (2), (3) et (4) (voir Figure 5.11). Pour un pixel P de coordonnées (x,y), l'algorithme permettant de déterminer les valeurs à choisir en fonction du type de la quadrique (positif, négatif, creux) et de la zone dans laquelle le pixel se trouve est le suivant :

```
Si P∈E1∩E2 Alors /* zone 1 */
  Si Positif Alors
    Zav = ZE1, Zar = ZE2, N = NE1
  Si Negatif Alors
    Zav = ZE1, Zar = ZE2, N = NE2
  Si Creux Alors
    Zav = Zmax
Sinon Si P∈E1 Alors /* zone 2 */
  Si Positif Alors
```

```
    Zav = ZE1, Zar = ZQar, N = NE1
  Si Negatif Alors
    Zav = ZE1, Zar = ZQar, N = NQar
  Si Creux Alors
    Zav = ZQar
Sinon Si P∈E2 Alors /* zone 3 */
  Si Positif Alors
    Zav = ZQav, Zar = ZE2, N = NQav
  Si Negatif Alors
    Zav = ZQav, Zar = ZE2, N = NE2
  Si Creux Alors
    Zav = ZQav, N = NQav
Sinon si P∈contour Alors /* zone 4 */
  Si Positif Alors
    Zav = ZQav, Zar = ZQar, N = NQav
  Si Negatif Alors
    Zav = ZQav, Zar = ZQar, N = NQar
```

La Figure 5.12 présente le schéma fonctionnel du Processeur Quadrique. L'unité de contrôle (par exemple une PLA) implémente l'algorithme décrit ci-dessus.

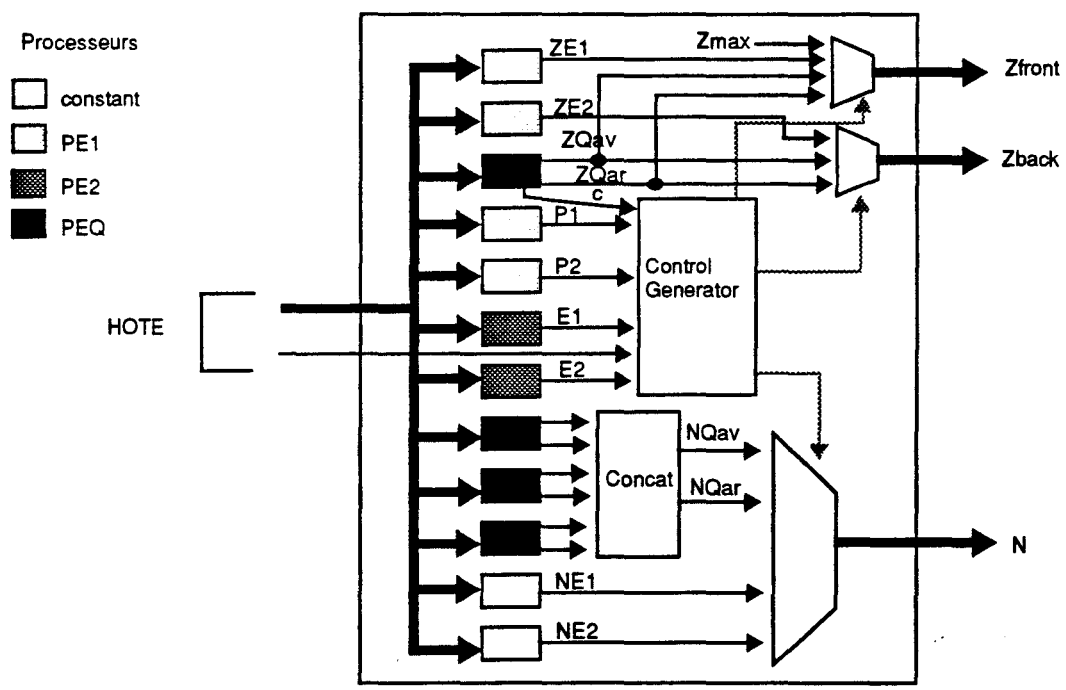


Figure 5.12 : le processeur quadrique

5.4 Deuxième génération de quadriques

Le principal défaut du processeur précédemment défini vient du fait que le contour de la quadrique doit être explicitement calculé par le hôte. Cette opération nécessite le calcul des

coefficients de deux expressions linéaires (6 coefficients) et de deux expressions quadratiques (12 coefficients). Par ailleurs, la méthode utilisée pour délimiter la quadrique se généralise difficilement à un nombre de plans de coupe supérieur à 2, car la forme du contour peut alors ne plus être conique (voir Figure 5.13). Or, ajouter un troisième plan de coupe permettrait de visualiser le paraboloide hyperbolique.

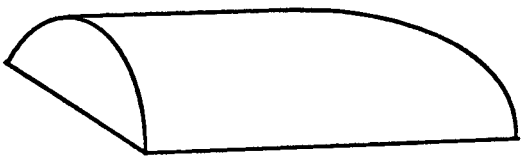


Figure 5.13 : trois plans de coupe non parallèles

Afin de pallier ces défauts, Eric Nyiri a proposé une méthode permettant de calculer directement le contour de la quadrique en utilisant uniquement les équations des plans de coupe (un plan est défini par son équation $z = ax + by + c$. En effet, si l'on considère que le plan de coupe détermine deux demi-espaces $E1$ et $E2$, délimiter la quadrique Q revient à effectuer l'opération booléenne $Q - E1$.

En tout point où la quadrique est présente, celle-ci est définie par sa profondeur avant *ZQav*, sa profondeur arrière *ZQar*, le vecteur normal à la face avant *NQav* et le vecteur normal à sa face arrière *NQar*. Le plan de coupe est quant à lui défini par sa profondeur *Zp* et son vecteur normal *Np*. L'algorithme de construction de la quadrique est alors les suivant [NYIR92a]

```
Si Zp > ZQar Alors
  Zav = Zar = Zmax /* la quadrique n'est pas visible */
Si (ZQar >= Zp >= ZQav) Alors
  Zav = Zp, Zar = ZQar /* le plan coupe la quadrique */
Si ZQav >Zp Alors
  Zav = ZQav, Zar = ZQar /* seule la quadrique est visible */
```

Cet algorithme doit être exécuté pour chaque plan de coupe (l'ordre de traitement des plans de coupe n'a pas d'importance). Le schéma fonctionnel (simplifié) du Processeur Quadrique à deux plans de coupe est alors le suivant

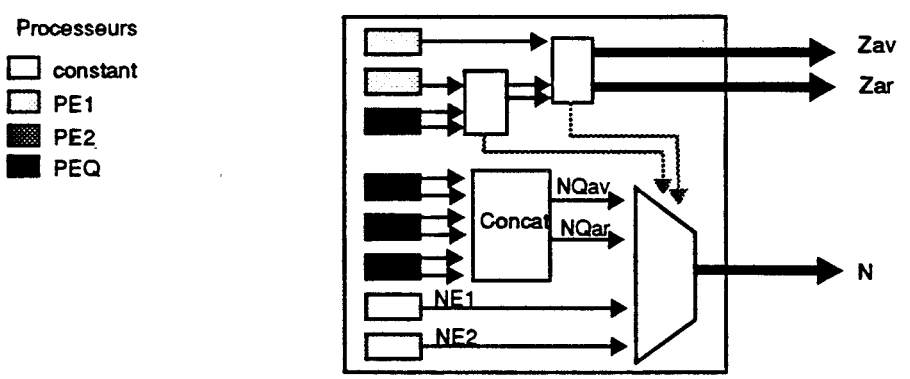


Figure 5.14 : le processeur quadrique de 2ème génération

Remarque

Il est possible de remplacer chaque PEQ calculant une composante de la normale par un PE1, un multiplieur et un additionneur puisqu'elle s'exprime par

$$N_x = L(x, y) + \alpha z$$

Rappelons enfin que ce processeur étant destiné à être intégré dans un pipeline objet, il est nécessaire d'ajouter un opérateur de Zbuffer effectuant l'élimination des parties cachées entre la quadrique générée par le processeur et celle générée par son prédécesseur dans le pipeline.

5.5 Conclusion

Les nombreuses tentatives effectuées ces dernières années pour proposer une alternative crédible à la facette dans le domaine de l'affichage (à ne pas confondre avec la modélisation) prouvent que cette voie, encore balbutiante, mérite d'être explorée. Dans certaines applications, les mêmes primitives peuvent être utilisées à la fois pour la modélisation et l'affichage. C'est par exemple le cas de la construction CSG à base de quadriques (c.f. le Ray Casting Engine et Pixel-Planes 5), ou encore de la modélisation moléculaire (principalement à base de quadriques). Ces applications semblent être un terrain prometteur d'investigations pour étudier des primitives de haut niveau.

Notre contribution à ce domaine s'est jusqu'à maintenant orientée vers l'affichage de quadriques. Nous avons présenté dans ce chapitre plusieurs possibilités pour réaliser un processeur quadrique dans le cadre d'un système graphique à parallélisme objet.

Nous avons exposé dans le chapitre 3 des conclusions assez pessimistes quant à l'avenir du parallélisme objet (de bas niveau) dans le domaine de l'affichage de facettes (en particulier à cause de la puissance du parallélisme pixel). Nous souhaitons conclure ce chapitre en évoquant le regain d'intérêt de ce type de parallélisme pour les primitives de haut niveau telles que les quadriques.

Supposons que nous disposions du processeur quadrique (fonctionnant à 33 MHz) décrit dans ce chapitre (la réalisation d'un tel circuit est complexe mais envisageable). Pour déterminer le nombre optimal de processeurs à utiliser dans le pipeline, il nout faut d'abord connaître le temps de chargement d'une quadrique. Pour cela, nous supposerons qu'une expression linéaire (i.e ses trois coefficients a, b et c) peut être chargée en un cycle du bus (64 bits type i860XP), et une expression quadratique (définie par ses six coefficients) en six cycles. Une quadrique à deux plans de coupe étant définie par six expressions linéaires et une expression quadratique, son temps de chargement est donc d'environ 600 ns. Le nombre optimum de processeurs dépend du découpage de l'écran (voir chapitre 3). Etant donné que la taille moyenne d'une quadrique à l'écran est relativement grande (beaucoup plus que celle d'une facette), il semble raisonnable d'opter pour un découpage en zones 128×128. Le nombre maximum de processeurs est alors d'environ 800. Ce chiffre signifie que, contrairement au pipeline facette, le pipeline quadrique n'est que peu limité par le débit de chargement des objets.

Considérons un pipeline composé de 10 processeurs quadriques fonctionnant sur des zones 128×128. Sa puissance intrinsèque est alors d'environ 20000 quadriques par seconde. En supposant un taux moyen de duplication de 3 (i.e. des quadriques contenues dans des boîtes englobantes 128×128, le système est alors capable de générer environ 7000 quadriques par seconde. Il est possible d'utiliser plus de processeurs pour augmenter la puissance du

système. Ainsi, une machine utilisant un pipeline de 100 processeurs quadriques développerait une puissance maximale d'environ 70.000 quadriques par seconde (toujours pour un facteur de duplication de 3). Ces chiffres montrent qu'il est possible de construire un système graphique puissant avec relativement peu de processeurs. Par ailleurs, l'utilisation de primitives d'affichage de haut niveau permet de soulager considérablement le travail du hôte [NYIR92b].

Ce travail montre qu'il est envisageable de développer un processeur quadrique pour un pipeline objet. Cependant, il faut noter que la quadrique est encore trop éloignée des primitives utilisées en modélisation (exceptées certaines applications très spécifiques comme la modélisation moléculaire). Les travaux sur les primitives de haut niveau méritent donc d'être poursuivis (notre équipe commence actuellement une étude sur la quartique).

Conclusion

Au terme de l'étude que nous venons de mener, il apparaît qu'une machine graphique massivement parallèle doit utiliser une réalisation partielle avec découpage de l'écran et classement des objets en fonction des zones où ils apparaissent. Nous avons effectué dans cette thèse une analyse formelle des unités de conversion objet et pixel utilisant un découpage de l'écran en zones rectangulaires, afin de déterminer la meilleure solution pour réaliser une machine graphique performante (la première partie de notre étude s'est limitée à l'affichage de facettes par la méthode de Gouraud). Nos critères de jugement ont été la puissance du système (en nombre de facettes par seconde), sa fréquence d'animation, et sa complexité matérielle. Les résultats que nous avons obtenus sont les suivants :

Les unités de conversion objet sont fortement limités par le débit de chargement des objets. Un pipeline objet optimal, utilisant une centaine de processeurs sur des zones 32×32, délivre une puissance maximale d'environ un million de facettes par seconde. Par ailleurs, nous avons également montré que la fréquence maximale d'animation d'un système objet est limitée par la fréquence de fonctionnement de son unité de conversion. Il est ainsi technologiquement difficile de dépasser les 30 Hz sur un écran 1280×1024.

Les unités de conversion pixel sont beaucoup moins limitées. Parmi toutes les méthodes envisageables pour réaliser un réseau de processeurs pixels, nous avons montré que l'approche multi-pipeline possède le meilleur rapport performances/coût. Ainsi, le réseau le plus simple que nous avons étudié (32×32 processeurs pixels) délivre une puissance d'environ deux millions de facettes par seconde, à une fréquence pouvant dépasser les 100 Hz. Sa complexité matérielle est raisonnable, puisqu'il peut être réalisé en 32 circuits intégrés de complexité modérée. Par ailleurs, nous avons montré que la puissance d'un tel réseau peut être facilement accrue soit en augmentant le degré de parallélisme interne des processeurs pixels, soit en accroissant la taille du réseau. Les réseaux les plus performants que nous avons étudiés peuvent atteindre, avec des technologies VLSI courantes, des puissances dépassant les vingt millions de (petits) polygones par seconde.

Par ailleurs, nous avons également montré, en généralisant la notion de facteur de duplication, que les systèmes pixel massivement parallèles à découpage en lignes (type SAGE) étaient, à complexité égale, moins puissants que les réseaux utilisant un découpage en zones. Il semble que les habitudes prises aux débuts de l'informatique graphique (les algorithmes scan-line ont toujours été très utilisés) aient eu dans ce domaine une influence négative.

La dernière partie de cette thèse a été consacrée à l'étude de primitives de plus haut niveau que la classique facette triangulaire. Le passage à de telles primitives permet d'une part d'améliorer la qualité des images produites, et d'autre part de soulager considérablement le travail du hôte. Nous avons montré qu'il est envisageable, dans le cadre d'une machine objet, de développer un processeur d'affichage de quadriques. Ces travaux méritent d'être poursuivis, car la quadrique est encore trop éloignée des primitives utilisées en modélisation (excepté des applications particulières telle que la visualisation moléculaire). Le passage des quadriques aux quartiques apportera peut-être un début de solution.

Nous souhaitons conclure ce travail en évoquant de nouveau le projet PixelFlow, développé à l'université de Caroline du Nord. Ce projet est basé sur la constatation suivante : puisque

la puissance des unités de conversion pixel est limitée par un facteur technologique (ne serait-ce que la fréquence de fonctionnement du réseau), un système puissant (plusieurs dizaines de millions de polygones par seconde) doit nécessairement utiliser plusieurs de ces unités en parallèle. La solution proposée pour PixelFlow est de répartir la base de données sur l'ensemble des unités de conversion, et de composer les images ainsi obtenues. Steve Molnar a montré que cette solution était la mieux adaptée pour construire une architecture puissante et extensible. Cependant, il faut souligner que la complexité imposée par ce type d'architecture est très importante comparée à celle de l'unité de conversion de base. Ce surcoût se traduit par un réseau de composition extrêmement coûteux (fond de panier contenant 256 voies à 132 MHz) et par une augmentation considérable de la complexité du réseau pixel (ajout de mémoire) afin de résoudre les problèmes d'équilibrage de charge entre les différentes unités. Ces coûts étant fixes quel que soit le nombre d'unités de conversion utilisées, un tel système n'est rentable que s'il en utilise beaucoup.

L'approche que nous proposons est différente, puisqu'elle suggère d'augmenter au maximum la puissance de l'unité de conversion, afin de différer le moment critique où il devient nécessaire d'en utiliser plusieurs en parallèle. Le réseau multi-pipeline proposé dans cette thèse, de par sa puissance et sa relative simplicité, nous semble à même de repousser cette limite.

Annexe A

Réalisation du processeur facette sous SOLO 1400

A.1 Réalisation sous SOLO 1400

Nous présentons dans ce paragraphe quelques détails de la conception du processeur sous SOLO 1400. Nous insistons en particulier sur la réalisation du PE1.

A.1.1 Le PE1

A.1.1.1 Registres utilisés

Les registres du PE1 sont conçus à partie de bascule D validées sur front descendant du signal d'horloge. Deux types de bascule sont utilisées: basule avec clear asynchrone (pour les registres de calcul) et bascule sans clear (pour le regitre de synchronisation). Nous donnons à titre d'indication la complexité des cellules de base utilisées sous SOLO:

- basule D sans clear: 26 transistors
- bascule D avec clear: 32 transistors

A.1.1.2 Additionneurs utilisés

Les additionneurs 12 bits utilisés pour le PE1 sont conçus à l'aide de 12 cellules d'addition 1 bit disponible dans la bibliothèque SOLO.

A.1.1.3 Description en langage Model

Le processeur facette a été entièrement décrit à l'aide du langage Model propre à SOLO 1400.Trois types de PE1 ont été défini : PE1 pour les normales, PE1 pour le Z et PE1 pour le contour.

```
Include "generate/ecpd15/ram24b.inc"
Include "generate/ecpd15/ramext.inc"

Part add12gd [a(0:11),b(0:11),cin] -> resbar(0:11), coutbar
Serial
Uninterrupted
Signal c(1:12)
sumcar [cin, a(0), b(0)] -> resbar(0), c(1)
Integer i
For i = 1:11 Cycle
sumcar [not[c(i)],a(i),b(i)] -> resbar(i), c(i+1)
Repeat
wire[c(12)] -> coutbar
End

Part registre12gd [clk, d(0:11), clrbar] -> q(0:11),qbar(0:11)
```

```
 Serial
 Uninterrupted

 Integer i
 For i= 0:11 Cycle
 bdffs [clk, d(i), clrbar] -> q(i),qbar(i)
 Repeat
End

Part registre12[clk, d(0:11)] -> q(0:11), qbar(0:11)
 Serial
 Uninterrupted
 Integer i
 For i=0:11 Cycle
 bdff [clk, d(i)] -> q(i), qbar(i)
 Repeat
End

Part ligne1gd [e(0:11),s(0:11),cin,clk,clrbar] -> sbar(0:11),coutbar,clock,clearbar
{ LIGNE 1 DE LA BOUCLE : ADDITIONNEUR + BUFFERS CLR ET CLK }
 Serial
 Uninterrupted
 add12gd [e(0:11),s(0:11),cin] -> sbar(0:11), coutbar
 buffer6 [clk] -> clock
 buffer6 [clrbar] -> clearbar
End

Part ligne2gd [sbar(0:11),coutbar,clock,clearbar] -> s(0:11), cout
{ LIGNE 2 DE LA BOUCLE : REGISTRE + REGISTRE RETENUE }
 Serial
 Uninterrupted
 registre12gd [clock, sbar(0:11), clearbar] -> --(0:11) , s(0:11)
 bdffs [clock, coutbar, clearbar] -> --, cout
End

Part add12_rebouclegd [e(0:11),clk,clrbar,cin] -> s(0:11),cout
{ BOUCLE ADDITIONNEUR / REGISTRE }
 Serial
 Signal sbar(0:11), coutbar, clock, clearbar
 Critical clock, clearbar
 ligne1gd [e(0:11),s(0:11),cin,clk,clrbar] -> sbar(0:11),coutbar,clock,clearbar:ligne1
 ligne2gd [sbar(0:11),coutbar,clock,clearbar] -> s(0:11),cout:ligne2
End

Part reg12syncgd [e(0:11),clk] -> s(0:11)
{ REGISTRE DE SYNCHRONISATION 12 BITS }
 Serial
 Uninterrupted
 Signal clock
 registre12[clock,e(0:11)] -> s(0:11),--(0:11)
 buffer6[clk] -> clock
End

Part mux24gd [a1(0:23),a2(0:23),select] -> s(0:23)
 Uninterrupted
 Serial
```

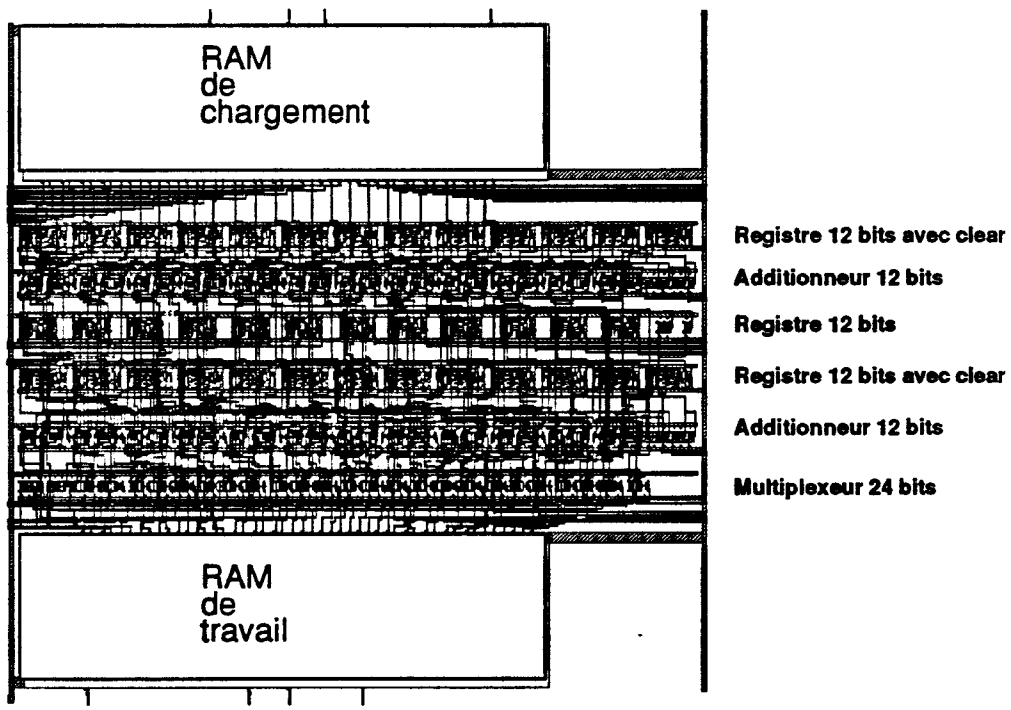
```
 Signal choice,choicebar,dummy
 buffer6 [select] -> choice
 not[select] -> dummy
 buffer6[dummy] -> choicebar
 Integer i
 For i = 0:23 Cycle
 not [andnor (2,2) [a2(i),choice,a1(i),choicebar]] -> s(i)
 Repeat
End

Part masquexorgd [e(0:23), control] -> s(0:23)
 Uninterrupted
 Serial
 Integer i
 For i = 0:22 Cycle
 eqv [e(i), control] -> s(i)
 Repeat
 wire [e(23)] -> s(23)
End

Part buffernormale [e(0:23)] -> s(0:23)
 Uninterrupted
 Serial
 Integer i
 For i = 0:23 Cycle
 buffer4 [e(i)] -> s(i)
 Repeat
End

Partpe1
decal[clock1,clock2,clearbar1,clearbar2,we,me,ad(0:1),weext,meext,adext(0:1),entext(0:23),select]      ->
out(0:23)

Signal cout1,a(0:23),sram(0:23)
Signal smux(0:23), emux(0:23), sadd1(0:11), sadd2(0:11), sadd2bar(0:11)
Signal lsb(0:11), lsbbar(0:11)
Signal control, notsignbit
Signal sxor(0:23)

Serial

Integer i
For i = 0:11 Cycle
 wire [sadd1(i)] -> emux(2*i)
 wire [sadd2(i)] -> emux(i+i+1)
Repeat

ram24b [we,me,ad(0:1),smux(23:0 By -1)] -> a(23:0 By -1) : ramtravail
mux24gd [sram(0:23),emux(0:23),select] -> smux(0:23):mux24
buffer3 [sadd2(11)] -> notsignbit : bufferxor1
buffer6 [notsignbit] -> control : bufferxor2
add12_rebouclegd [a(0:22 By 2),clock1,clearbar1,GND] -> sadd1(0:11), cout1 : boucle1
reg12syncgd [sadd1(0:11),clock1] -> lsb(0:11):sync
add12_rebouclegd [a(1:23 By 2),clock2,clearbar2,cout1] -> sadd2(0:11), – : boucle2
masquexorgd [lsb(0:11),sadd2(0:10), notsignbit, control] -> sxor(0:23): xor
buffernormale [sxor(0:23)] -> out(0:23) : buffersortie
```

```
ramext [weext,meext,adext(0:1),entext(0:23)] -> sram(0:23) : ramext

End


Part
pe1contour[clock1,clock2,clearbar1,clearbar2,we,me,ad(0:1),weext,meext,adext(0:1),entext(0:23),select] -
> sign

Signal cout1,a(0:23),sram(0:23)
Signal smux(0:23), emux(0:23), sadd1(0:11), sadd2(0:11)

Serial

Integer i
For i = 0:11 Cycle
 wire [sadd1(i)] -> emux(2*i)
 wire [sadd2(i)] -> emux(i+i+1)
Repeat

ram24b [we,me,ad(0:1),smux(23:0 By -1)] -> a(23:0 By -1) : ramtravail
mux24gd [sram(0:23),emux(0:23),select] -> smux(0:23):mux24
add12_rebouclegd [a(0:22 By 2),clock1,clearbar1,GND] -> sadd1(0:11), cout1 : boucle1
add12_rebouclegd [a(1:23 By 2),clock2,clearbar2,cout1] -> sadd2(0:11), -- : boucle2
ramext [weext,meext,adext(0:1),entext(0:23)] -> sram(0:23) : ramext
wire [sadd2(11)] -> sign

End

Part pe1z[clock1,clock2,clearbar1,clearbar2,we,me,ad(0:1),weext,meext,adext(0:1),entext(0:23),select] ->
s(0:23)
Signal cout1,a(0:23),sram(0:23)
Signal smux(0:23), emux(0:23), sadd1(0:11), sadd2(0:11)

Serial

Integer i
For i = 0:11 Cycle
 wire [sadd1(i)] -> emux(2*i)
 wire [sadd2(i)] -> emux(i+i+1)
Repeat

ram24b [we,me,ad(0:1),smux(23:0 By -1)] -> a(23:0 By -1) : ramtravail
mux24gd [sram(0:23),emux(0:23),select] -> smux(0:23):mux24
add12_rebouclegd [a(0:22 By 2),clock1,clearbar1,GND] -> sadd1(0:11), cout1 : boucle1
add12_rebouclegd [a(1:23 By 2),clock2,clearbar2,cout1] -> sadd2(0:11), -- : boucle2
ramext [weext,meext,adext(0:1),entext(0:23)] -> sram(0:23) : ramext
For i = 0:11 Cycle
 wire [sadd1(i)] -> s(i)
 wire [sadd2(i)] -> s(i+12)
Repeat
End

End Of File
```

A.1.1.4 Placement

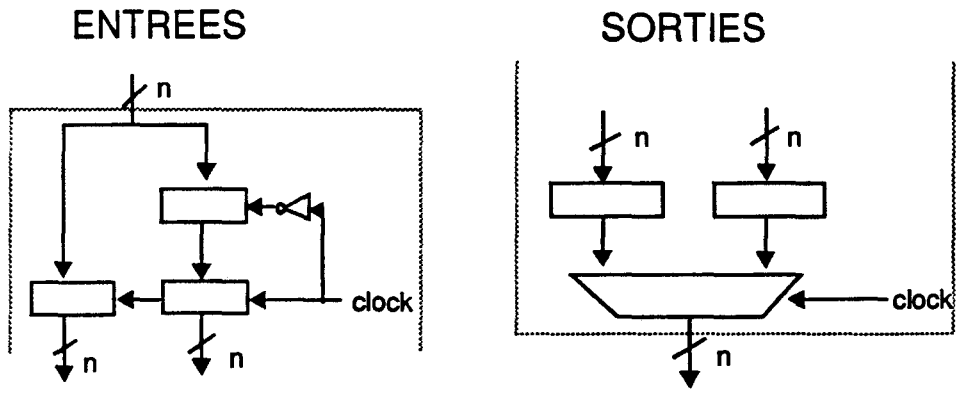
/FACETTE/PE1Z/RAMTRAVAIL
NEWROW L->R
/FACETTE/PE1Z/MUX24
NEWROW L->R
/FACETTE/PE1Z/BOUCLE1/LIGNE1
NEWROW L-> R
/FACETTE/PE1Z/BOUCLE1/LIGNE2
NEWROW L->R
/FACETTE/PE1Z/BOUCLE2/LIGNE1
NEWROW L-> R
/FACETTE/PE1Z/BOUCLE2/LIGNE2
NEWROW L->R
/FACETTE/PE1Z/RAMEXT
NEWROW L->R

A.1.1.5 réalisation VLSI du PE1



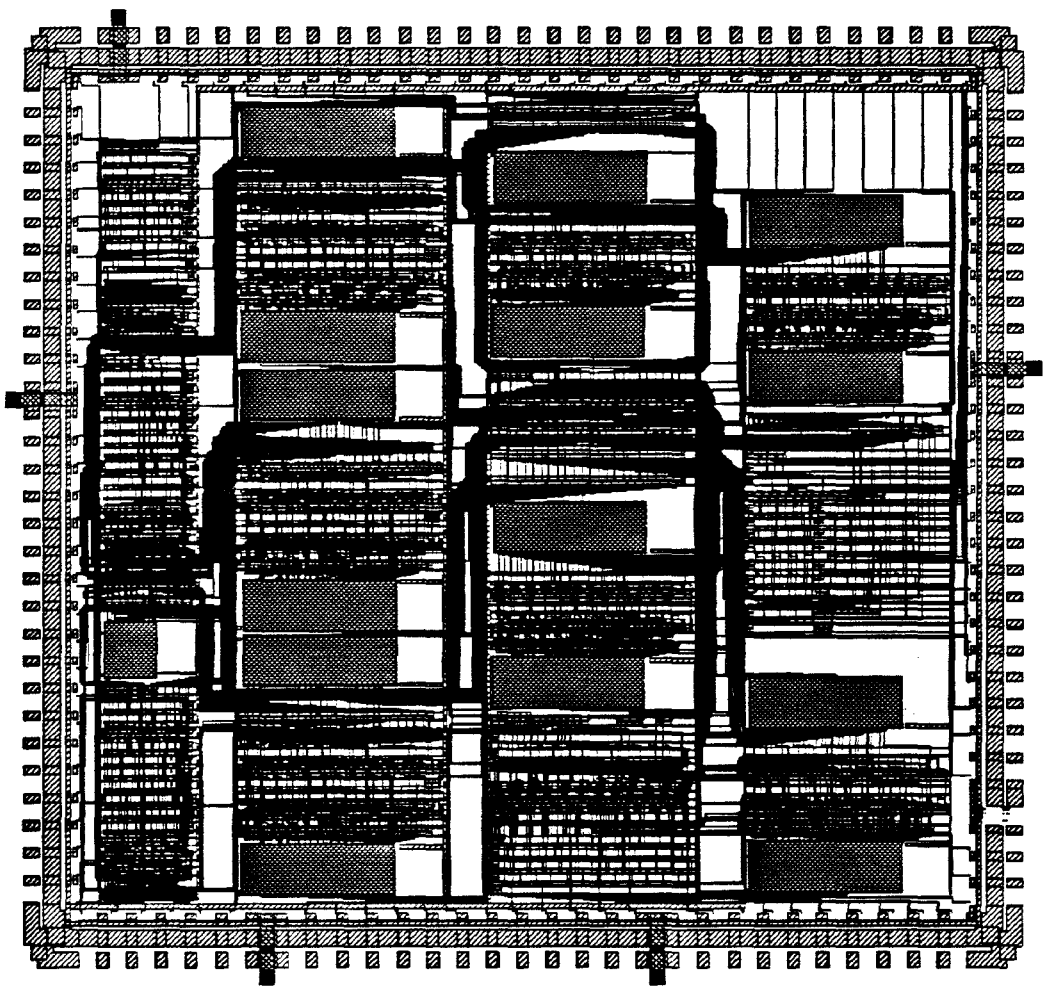
A.1.2 Les Entrées/Sorties

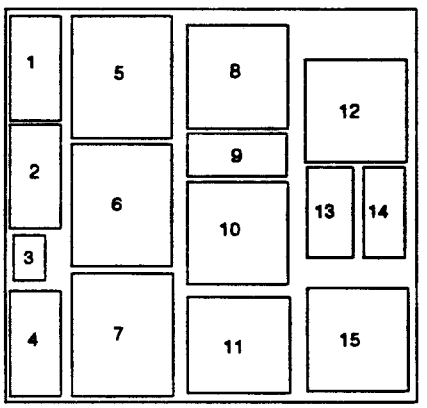
Afin de diviser par deux le nombre de broches du processeur (et donc la largeur du pipeline), les données (profondeur, normale, couleur, mot de contrôle) sont transmises d'un processeur à son voisin en deux temps. Les poids faibles sont transmis sur front bas du signal d'horloge, les poids forts sur front haut.



A.1.3 Le circuit VLSI

A.1.3.1 Masque VLSI





1,2,4: pre-normalisateurs
3: contrôle pre-normalisateurs
8, 10, 12: PE1s (contour)
9: buffers
5,6,7: PE1s (vecteur normal)
15: PE1 (Z)
11: Zbuffer
13: couleur de base
14: unité de commande

A.1.3.2 Charactéristiques techniques

- CMOS 1,5 µ
- 8,4 × 7,86 mm
- ~40.000 transistors
- boîtiers PGA 120 broches

Bibliographie

[ATAM89] A. Atamenia
Architectures cellulaires pour la synthèse d'images
Thèse de Doctorat, Université de Lille, juin 1989

[AKEL89] K. Akeley
The Silicon Graphics 4D/240GTX Superworkstation.
IEEE Computer Graphics and Applications, vol. 9 num. 4, july 89, pp 71-83

[AKEL88] K. Akeley et T. Jermoluk
High-Performance Polygon Rendering
ACM Computer Graphics, vol. 22 num. 4, august 1988, pp 239-246

[APGA88] B. Apgar, B. Bersack et A.Mammen
A Display System for the Stellar Graphics Supercomputer Model GS1000.
ACM Computer Graphics, vol. 22 num. 4, august 1988, pp 255-262

[BISH86] G. Bishop et D. Weiner
Fast Phong Shading
ACM Computer Graphics, vol. 20 num. 4, august 1986, pp 103-106

[BLIN78] J.F. Blinn
Computer Display of Curved Surfaces
Ph.D. thesis, University of Utah, Department of Computer Science, December 1978

[CARP84] L. Carpenter
The A-buffer, an Antialiased Hidden Surface Method
ACM Computer Graphics, vol. 18 num. 3, august 84, pp 103-108

[CHAI91] C. Chaillou
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.
Thèse de Doctorat, Université de Lille, Janvier 1991

[CHAI92a] C. Chaillou, S. Karpf et M. Meriaux
I.M.O.G.E.N.E: A Solution to the Real Time Animation Problem
Advances in Computer Graphics Hardware V, Springer Verlag,

[CHAI92b] C.Chaillou
Architectures des systèmes pour la synthèse d'images
Dunod Informatique, 1992

[CLAR82] J. Clark
The Geometry Engine: a VLSI Geometry System for Graphics
ACM Computer Graphics, vol. 16 num. 3, july 1982, pp 127-133

[CLAU91] U. Claussen
Real Time Phong Shading
Advances in Computer Graphics Hardware V, Springer Verlag, to appear in 1991

[COHE80] D. Cohen et S. Demetrescu
Presentation at SIGGRAPH'80 Panel on Trends on High Performance Graphic System, 1980.

[COHE85] M. Cohen et D. Greenberg
The Hemi-Cube, a Radiosity Solution for Complex Environments
ACM Computer Graphics vol. 19 num. 3, july 1985, pp 31-40

[COX92] M. Cox et P. Hanrahan
Depth Complexity in Object-Parallel Graphics Architectures
Proceedings seventh Eurographics Workshop on Graphics Hardware, september 1992, pp 204-222 et Research Report CS-TR-382-92, september 1992

[CROW77] F. Crow
Shadow Algorithms for Computer Graphics.
ACM Computer Graphics, vol. 11 num. 3, july 1977, pp 242-248

[DEER88] M. Deering, S. Winner, B. Schediwy et 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

[DEGR93] S. Degrande
Définition d'une machine cellulaire pour la mise en oeuvre d'un lancer de rayon massivement parallèle
Thèse de Doctorat, Université de Lille I, à soutenir en mars 1993.

[DENA88] D. Denault, E. Ryherd, J. Torborg et al.
VLSI Drawing Processor Utilizing Multiple Parallel Scan-Line Processors.
Advances in Computer Graphics Hardware II, Springer Verlag, 1988, pp 167-182

[DIED88] T. Diede, C.F. Hagenmaier, G.S. Miranker et al.
The Titan Graphics Supercomputer Architecture.
IEEE Computer, September 1988, pp. 13-30.

[ELLI91] J.L. Ellis, G. Kedem et al.
The RayCasting Engine and Ray Representations
Proceedings ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, june 1991, pp 255-267

[ELLS90] D.E. Ellsworth, H. Good and B. Tebbs
Distributing Display List s on a Multicomputer
Proceedings 1990 Symposium on Interactive 3D Graphics, in Computer Graphics vol. 24 num. 2, march 1990, pp 147-154

[ELLS91] D.E. Ellsworth
Parallel Architectures and Algorithms for Real-Time Synthesis of High-Quality Images Using Deferred Shading
Proceedings Workshop on Algorithms and Parallel VLSI Architectures, 1990

[EVA92] Evans & Sutherland Computer
Freedom Technical Report
septembre 1992

[EYLE88] J. Eyles, J. Austin, H. Fuchs et al.
Pixel-Planes 4: A Summary
Advances in Computer Graphics Hardware II, Springer Verlag, 1988, pp 183-208

[FOLE90] J. Foley, A. Van Dam, S. Feiner et J. Hughes
Computer Graphics: Principles and Practice (second edition)
The systems Programming Series, Addison Wesley 12110, 1990

[FUCH85] H. Fuchs, J. Goldfeather, J. Hultquist et al.
Fast Spheres, Shadows, Textures, Transparencies and Image Enhancement in Pixel-Planes
ACM Computer Graphics, vol.19 num.3, july 1985, pp 111-120

[FUCH89] H. Fuchs, J. Poulton, J. Eyles et al.
Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories.
ACM Computer Graphics, vol. 23 num. 3, july 1989, pp 79-88

[FUSS82] F. Fussell et B.D. Rathi
A VLSI-Oriented Architecture for Real-Time Raster Display of Shaded Polygons
Proceedings Graphics Interface 82, pp 373-380

[GHAR85] N. Gharachorloo et C. Pottle
SUPER BUFFER: A Systolic VLSI Graphics Engine for Real Time Raster Image Generation
Proceedings Chapel Hill Conference on VLSI, 1985

[GHAR88] N. Gharachorloo, S. Gupta, E. Hokenek et al.
Subnanosecond Pixel Rendering with Million Transistor Chips.
ACM Computer Graphics, vol. 22 num.4, august 1988, pp 41-49

[GHAR89] N. Gharachorloo, S. Gupta, R. Sproull et al.
A Characterization of Ten Rasterization Techniques
ACM Computer Graphics, vol. 23 num. 3, july 89, pp 355-368

[GOLD86] J. Goldfeather, J. Hultquist, H. Fuchs
Fast Constructive Geometry Display in the Pixel-Powers Graphics System
ACM Computer Graphics, vol. 20 num. 4, august 1986, pp 107-116

[GORA84] C. Goral, K. Torrance, D. Greenberg, B. Battaile
Modeling the Interaction of Light Between Diffuse Surfaces
ACM Computer Graphics, vol. 18 num. 3, july 1984, pp 213-222

[GROS91] P. Gros
Etude et mise en oeuvre d'une architecture multiprocesseurs pour la synthèse d'images: vers une algorithmie adaptée à une implantation VLSI
Thèse de Doctorat, Université de Technologie de Compiègne, février 1991

[GOUR71] H. Gouraud
Continuous Shading of Curved Surfaces
IEEE Transaction on Computers, vol. C-20 num. 6, june 1971, pp 623-629

[HAEB90] P. Haeberli et K. Akeley
The Accumulation Buffer: Hardware Support for High-Quality Rendering
ACM Computer Graphics, vol. 24, num. 4, august 1990, pp. 309-318

[HASH90] R. Hashemian
Square Rooting Algorithms for integer and Floating Point Numbers
IEEE Transactions on Computer, vol. 39 num. 8, august 1990, pp 1025-1029

[KARP89] S. Karpf
Element du projet IMOGENE: Etude et Réalisation du Processeur Elémentaire
Mémoire de DEA, Université de Lille, Septembre 1989

[KARP91] S. Karpf, C. Chaillou, E. Nyiri et M. Meriaux
Real-Time Display of Quadric Objects in the I.M.O.G.E.N.E Machine.
Proceedings ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, june 1991, pp 269-277

[KARP92] S. Karpf et C.Chaillou
An Efficient Massively Parallel Rasterization Scheme For A High Performance Graphics System.
Proceedings Seventh Eurographics Workshop on Graphics Hardware, septembre 92, pp 158-169

[KAUF91] A. Kaufman
Volume Visualization
IEEE Computer Society Press. 1991

[KEDE89] G. Kedem et J.L. Ellis
The Ray Casting Machine.
Parallel Processing for Computer Vision and Display, Addison Wesley, pp 378-401

[KELL92] M. Kelley, S. Winner et K. Gould
A Scalable Hardware Render Accelerator using a Modified Scanline Algorithm
ACM Computer Graphics, vol. 26 num. 2, july 92, pp 241-248

[KIRK90] D. Kirk et D. Voorhies
The Rendering Architecture of the DN10000VS.
ACM Computer Graphics, vol. 24 num. 4, august 90, pp 299-307

[LAPO91] H. Laporte
Etude et conception d'un composant VLSI dans le cadre du projet IMOGENE: l'extracteur de racine carrée.
Mémoire de DEA, Université de Lille, juin 1991.

[LEFE91] V. Lefevere, S. Karpf, C. Chaillou et M. Meriaux
The I.M.O.G.E.N.E Machine: Some Hardware Elements.
Sixth Eurographics Workshop on Graphics Hardware. To be published in Advances in Computer Graphics Hardware VI, Springer Verlag.

[LEFE92] V. Lefévère et C. Chaillou
Low Cost Hardware for Real Time Phong Lighting
Proceedings Graphics Interface 92 Workshop on Local Illumination, may 1992, pp 23-30.

[LEPR89] E. Leprêtre
Algorithmes Parallèles et Architectures Cellulaires pour la Synthèse d'Images
Thèse de Doctorat, Université de Lille, juin 1989

[LUCA91] M. Lucas
Parallélisme et synthèse d'image
TSI, vol. 10 num. 3, 1991, pp 171-202

[McLE88] J. McLeod
HP delivers photo realism on an interactive system
Electronics, March, 17, 1988, pp. 95-97

[MERI84] M. Mériaux
Contribution à l'Imagerie Informatique : Aspects Algorithmiques et Architecturaux
Thèse d'Etat, Université de Lille I, 1984

[MOLN88] S. Molnar
Combining Z-buffer Engines for Higher-Speed Rendering
Advances in Computer Graphics Hardware III, Springer Verlag, 1988, pp 171-182

[MOLN91] S. Molnar
Image-Composition Architecture for Real-Time Image Generation
PhD Thesis, University of North Carolina, October 1991.

[MOLN92] S. Molnar, J. Eyles et J. Poulton
PixelFlow: High-Speed Rendering Using Image Composition
ACM Computer Graphics, vol. 26 num. 2, july 92, pp 231-240

[NYIR90] E. Nyiri
Modélisation et Simulation d'Objets 3D a l'Aide d'Expressions du Second Degré
Mémoire de DEA, Université de Lille, septembre 1990

[NYIR92a] E. Nyiri et C. Chaillou
Aspect Logiciel du Projet IMOGENE
Actes de MICAD 92, Editions Hermès, pp 201-217

[NYIR92b] E. Nyiri, C. Chaillou et M. Froumentin
Le Polyèdre et la Quadrique comme primitives d'affichage
Actes des Journées GROPLAN 92, novembre 1992

[PHON75] B.T. Phong
llumination for Computer Generated Pictures.
Communications ACM, vol. 18 num. 18, june 1975, pp 311-317

[PINE88] J. Pineda
A Parallel Algorithm for Polygon Rasterization
ACM Computer Graphics, vol. 22, num. 4, august 1988, pp 17-20

[POTM89] M. Potmesil et E. Hoffert
The Pixel Machine: A Parallel Image Computer.
ACM Computer Graphics, vol. 23 num. 3, july 89, pp 69-78

[POUL92] J. Poulton, J. Eyles, S. Molnar and H. Fuchs
Breaking the Frame-Buffer Bottleneck with Logic-Enhanced Memories
IEEE Computer Graphics and Applications, vol. 12 num. 6, novembre 1992, pp 65-74

[PREU91] A. Preux
Le CSG et le Temps Réel
Mémoire de DEA, Université de Lille, septembre 1991

[PREU92] A. Preux et C. Chaillou
Un Algorithme Economique pour l'Antialiassage des Bords de Polygones
Actes des Journées GROPLAN 92, novembre 1992

[SCHN88a] B.O. Schneider
A Processor for an Object-Oriented Rendering System.
Computer Graphics Forum, num. 7, 1988, pp301-310

[SCHN88b] B.O. Schneider et U. Claussen
PROOF: An Architecture for Rendering in Object Space.
Advances in Computer Graphics Hardware III, Springer Verlag, 1988, pp 121-140

[SCHN91] B.O. Schneider
Towards a Taxonomy for Display Processors
Advances in Computer Graphics IV, 1991, Springer-Verlag, pp 3-36

[SEGA92] M. Segal, C. Korobkin, R. van Widenfelt et al.
Fast Shadows and Lighting Effects Using Texture Mapping
ACM Computer Graphics, vol. 26 num. 2, july 1992, pp 249-252

[SILI90] Silicon Graphics.
Power Series Technical Report. 1990.

[SILI92] Silicon Graphics.
RealityEngine in Visual Simulation
Technical Report, 1992.

[SILL89] F. Sillion et C. Puech
A General Two-Pass Method Integrating Specular and Diffuse Reflection
ACM Computer Graphics vol. 23 num. 3, july 1989, pp 335-344

[SHAW88] C.D. Shaw , M. Green et J. Schaeffer
A VLSI Architecture for Image Composition
Advances in Computer Graphics Hardware III, 1988, Springer Verlag, pp 183-199

[SWAN86] R. Swanson et L. Thayer
A Fast Shaded-Polygon Renderer.
ACM Computer Graphics, vol. 20 num. 4, august 1986, pp 95-101

[TORB87] J. Torborg
A Parallel Processor for Graphics Arithmetic Operations.
ACM Computer Graphics, vol. 21 num. 4, july 87, pp 197-204

[WEIN81] R. Weinberg
Parallel Processing Image Synthesis and Anti-Aliasing.
ACM Computer Graphics, vol. 15, num. 3, august 1981, pp. 55-62

[WHIT80] T. Whitted
An Improved Illumination Model for Shaded Display
Communication ACM, vol. 23, 1980, pp 343-349

[WILL78] L. Williams
Casting Curved Shadows on Curved Surfaces
ACM Computer Graphics, vol. 12 num. 3, july 1978, pp. 270-274