NUMERO D'ORDRE : 1267

ANNEE : 1994

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

Eric NYIRI

Etude de la Quadrique comme Primitive d'Affichage en Synthèse d'Images

Thèse soutenue le 8 février 1994, devant la commission d'examen :

Président :	M. MERIAUX	LIFL
Directeur de Thèse :	M. MERIAUX	LIFL
Rapporteurs :	D. ARQUES	Laboratoire d'Informatique de Besançon
	G. HEGRON	Ecole des Mines de Nantes
Examinateurs :	R. CAUBET	IRIT
	C. CHAILLOU	LIFL
	S. KARPF	LIFL

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. BLAYS 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. 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
Mi. 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. 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. 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

Je remercie Monsieur Michel Mériaux, Professeur à l'EUDIL, qui m'a proposé ce sujet de recherche et je suis très heureux qu'il préside ce jury.

Je remercie également Messieurs Didier Arquès, Professeur au Laboratoire d'Informatique de Besançon, et Gérard Hégron, Professeur à l'Ecole de Mines de Nantes, de m'avoir fait l'honneur d'être rapporteurs de cette thèse.

Je remercie chaleureusement Monsieur René Caubet, Professeur à l' I.R.I.T., pour avoir accepté de juger ce travail.

Je travaille en étroite collaboration avec Christophe Chaillou depuis quatre ans. Il a été l'initiateur de nombreux projets, dont celui-ci, et a permis que ce travail se déroule dans les meilleurs conditions.

Sylvain Karpf est une des personnes qui m'ont conduit à l'infographie. Je suis donc très heureux qu'il fasse partie de ce jury.

Je voudrais encourager deux personnes pour la poursuite de leurs travaux :

- Hervé Laporte, qui continue le travail sur la visualisation des quadriques. Nos discutions ont parfois été bruyantes, mais elles nous ont permis d'éclaircir certains problèmes sur le sujet.

- Max Froumentin, qui étudie la modélisation à l'aide de quadriques. Son travail amènera des réponses à certains problèmes posés dans cette thèse.

L'équipe graphique du L.I.F.L. est composée d'une faune éclectique et accueillante. Je tiens à remercier tout particulièrement Samuel Degrande (Sam) pour sa compétence, qui est indéniable, et sa gentillesse, qui est toujours égale.

Je remercie toutes les personnes du L.I.F.L. qui, de jour comme de nuit, y entretiennent une ambiance chaleureuse et décontractée, propice au travail.

Merci enfin à Henri Glanc pour avoir assuré la reproduction de cette thèse avec la compétence que tout le monde lui reconnaît.

Il serait ingrat d'omettre ici tous les membres de ma famille, qui m'ont apporté la patience et la compréhension nécessaires à l'aboutissement de ce travail.

Sommaire

Introduction

Le domaine de la synthèse d'image peut être divisé en deux grandes catégories d'applications suivant que l'on s'attache au réalisme ou à la vitesse d'affichage des images.

• La production d'images très réalistes est utilisée pour la réalisation de films, spots publicitaires et génériques d'émissions télévisées, applications qui touchent le grand public. Elle nécessite l'emploi de méthodes coûteuses en temps. Le lancer de rayons par exemple est basé sur une simulation des rayons lumineux parvenant à un observateur. La radiosité par contre ne prend pas en compte l'observateur, mais les interactions lumineuses entre les différents objets d'une scène. Les calculs effectués par ces deux méthodes sont considérables, puisque le nombre de rayons lumineux traités pour un lancer de rayons est grand et que le calcul des interactions lumineuses employées en radiosité est complexe. Les recherches dans ce domaine s'orientent donc vers une parallélisation des algorithmes employés, afin de diminuer les temps de calcul, ce qui permet d'obtenir des images plus rapidement et/ou d'augmenter la complexité des calculs dans les traitements pour produire des images encore plus réalistes.

• Les principales applications visées par la production rapide d'images sont les simulateurs de vol et de conduite, la simulation scientifique (comme la simulation moléculaire) et les mondes virtuels. De nombreuses recherches ont été menées afin de concevoir des machines parallèles dédiées pouvant produire des images en temps-réel. Les machines de rendu géométrique[1] affichent des facettes planes pour garantir le temps réel, les objets à afficher sont alors approchés par un ensemble de facettes (facettisation). Les machines de rendu volumique permettent de visualiser des scènes définies directement par un ensemble de volumes élémentaires (appelés voxels), ces données étant assez simples pour être traitées rapidement. L'approximation des objets par un ensemble de facettes ou de points, oblige à utiliser et à afficher un grand nombre de ces objets primaires, afin que les images produites soient de bonne qualité.

Dans le cadre du rendu géométrique, la qualité des images est généralement augmentée en utilisant une facettisation de plus en plus fine des objets : les machines sont alors contraintes à traiter de plus en plus de facettes. Les études actuelles en ce domaine cherchent donc à définir des machines toujours plus performantes, afin d'accroître le nombre de facettes affichées par images. Actuellement, les plus puissantes de ces machines annoncent des performances de l'ordre d'un million de facettes affichées par seconde.

Dans cette thèse, nous tentons d'apporter une autre réponse à l'amélioration de la qualité des images en changeant de primitive d'affichage : l'emploi d'une primitive plus complexe permet de mieux approcher les objets à afficher. Le nombre de primitives à convertir pour le rendu est ainsi moins important.

Notre but est d'étudier la quadrique comme primitive de visualisation pour les machines de rendu géométrique et de la comparer à la primitive facette.

L'organisation de ce document est la suivante :

1. Le rendu géométrique est présenté dans le premier chapitre

• Le premier chapitre présente l'utilisation des primitives simples pour l'affichage. Nous étudions tout d'abord la relation entre la modélisation et le rendu en facettes, puis nous exposons les différents algorithmes utilisés pour l'affichage de celles-ci. Nous détaillons ensuite différents types de parallélismes employés pour augmenter la vitesse de visualisation d'une scène. Nous terminons ce chapitre par la description de deux méthodes utilisant des approximations pour la conversion des sphères et des cylindres en pixels.

• Le second chapitre est consacré à la définition d'unités de conversion de quadriques en pixels. Nous décrivons la quadrique comme figure géométrique, puis nous étudions les notions liées à l'affichage de quadriques. Nous introduisons les éléments ayant servi pour la définition d'unités matérielles de conversion, en commençant par deux études précédentes sur l'affichage d'objets définis à partir de quadriques. Le cadre d'utilisation de ces unités de conversion est enfin étudié.

• Le troisième chapitre est une étude sur la création des images de synthèse lors de l'utilisation de la primitive d'affichage quadrique. Nous commençons par la modélisation des objets pour le rendu en quadrique. Nous étudions ensuite les techniques d'amélioration de la qualité du rendu qui permettent d'obtenir des images plus réalistes et leurs utilisations avec la nouvelle primitive.

• Le quatrième chapitre présente un émulateur logiciel d'une machine utilisant les processeurs quadriques définis dans le second chapitre. Cet émulateur intègre un programme d'animation qui nous a permis d'effectuer diverses expériences.

• Le cinquième chapitre est une évaluation des coûts du processus d'affichage en fonction de la primitive utilisée. Pour cette étude, nous avons séparé les coûts des opérations effectuées avant la conversion, des coûts de conversion des objets. Nous démontrons ainsi que les machines d'affichage intégrant la primitive quadrique peuvent avoir un avenir.

Chapitre 1 :
Le Rendu Géométrique

Dans ce chapitre, nous allons décrire un processus d'affichage de scène, qui à partir de la description d'une scène sous la forme d'un ensemble d'objets primaires (primitives d'affichage) abouti à un ensemble de pixels. Ce processus emploie la projection orthographique des objets sur l'écran et est appelé rendu géométrique.

Le rendu géométrique utilise actuellement la facette plane triangulaire (ou quadrilatère) comme primitive d'affichage. Cette primitive est employée par la plupart des stations graphiques commerciales, du fait de sa simplicité de conversion en pixels.

Dans un premier temps, nous montrons comment les objets d'une scène sont définis puis transformés en un ensemble de facettes utilisées pour l'affichage. Dans un second temps, nous détaillons les différentes méthodes de visualisation de facettes représentant des objets tridimensionnels. Nous étudions ensuite les différentes méthodes de parallélisation employées lors du rendu de facette, en vue de l'implantation matérielle sur machines multi-processeurs. Enfin nous décrivons le processus d'affichage des sphères et des cylindres proposé par l'équipe de recherche de Chapel Hill (University of North Carolina), qui a voulu éviter les problèmes liés à l'utilisation de la facette comme primitive de rendu.

1.1 Les objets en rendu géométrique

Lors de la modélisation d'une scène, il est important de pouvoir visualiser et modifier rapidement les objets afin de parvenir rapidement à la représentation finale que l'on désire. On recherche l'interactivité entre l'utilisateur et le programme.

Cette rapidité d'affichage est également nécessaire dès que l'on veut animer des images en faisant intervenir un participant extérieur comme dans un simulateur de vol. Dans ce cas, la rapidité de l'affichage est un facteur crucial, puisque la perception humaine nécessite une vitesse d'au moins 25 images par seconde, afin qu'un observateur ne perçoive pas le saut entre deux images successives.

1.1.1 Les primitives d'affichage

Pour obtenir une grande rapidité lors de l'affichage, les machines de rendu emploient des primitives simples de conversion. Le choix de cette primitive détermine alors en grande partie la qualité des images générées.

Les premières machines de rendu géométrique de scène tridimensionnelle affichaient des segments et par conséquent, ne permettaient de visualiser que le contour des objets sous la forme de segments (wire frame). Cet affichage est rapide et permet l'interaction, mais la visualisation des contours ne permet pas

d'appréhender convenablement la position et la forme des objets tridimensionnels (comme dans le cas du cube de Necker de la Figure 1.1).

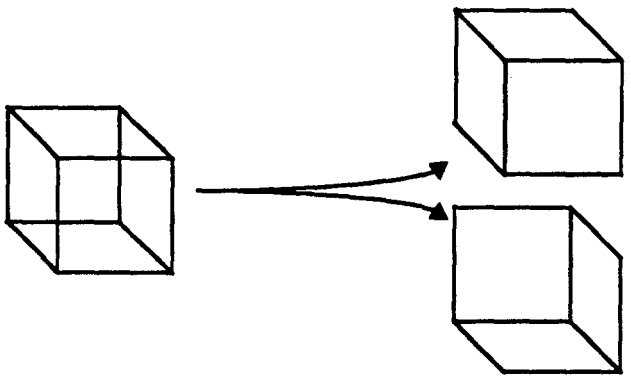


Figure 1.1 : Le cube de Necker

L'évolution de la vitesse et de la complexité matérielle des machines permit par la suite d'employer une autre primitive d'affichage, qui répondait en partie aux problèmes de la perception tridimensionnelle des objets, tout en restant assez simple pour que l'affichage soit rapide : la facette.

La primitive facette employée par les machines de conversion est définie par la donnée de trois ou quatre points, qui sont ses sommets; pour la suite de cet exposé, nous considérons les facettes triangulaires, les facettes quadrilatérales n'étant que peu employées. La surface d'une facette est plane, ce qui facilite sa conversion.

1.1.2 La modélisation et la facettisation

La visualisation d'un objet complexe nécessite que cet objet soit construit à l'aide de la primitive d'affichage. Or les primitives de modélisation sont différentes des primitives de rendu. Actuellement, les machines de rendu géométrique n'utilisent que la primitive facette, il est donc nécessaire de facettiser les objets.

Le cadre général de la modélisation

Le premier domaine où la modélisation fit son apparition, fut la CFAO (Conception et Fabrication Assistées par Ordinateur). La fabrication ayant été automatisée, la modélisation consistait alors à déterminer les opérations à effectuer sur un bloc de matière initial, afin d'obtenir un objet décrit sous la forme de côtes et autres valeurs nécessaires à l'usinage. La principale action étant le retrait de matière par des fraiseuses et autres machines outils, une opération fort employée était alors la soustraction dans un bloc de matière initial.

Cette utilisation amena à développer une modélisation à base de volumes initiaux simples, composés par les opérations d'union, de soustraction et d'intersection. Cette méthode de modélisation prit le nom de Constructive Solid Geometry (C.S.G).

Le modèle volumique (C.S.G)

- La définition d'un objet

Un objet est défini en C.S.G par un ensemble de volumes de bases et d'opérations sur ces volumes (Figure 1.2).

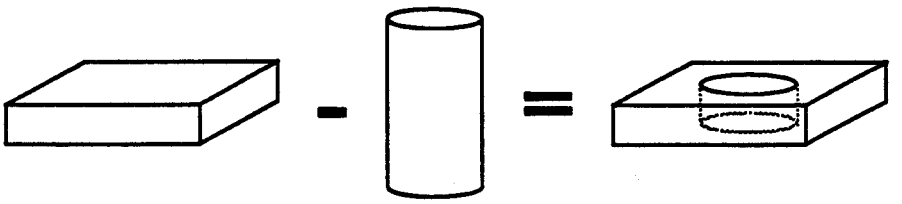


Figure 1.2 : Un objet en C.S.G.

Deux représentations équivalentes sont possibles pour un objet donné (Figure 1.3):

- L'arbre binaire (a) : les volumes initiaux sont les feuilles de l'arbre et les opérations sont les nœuds. La résolution (la construction de l'objet) s'effectue par le parcours de l'arbre du bas vers le haut. Des algorithmes ont été élaborés afin d'obtenir des arbres ayant certaines particularités (être sous forme d'union de groupes d'intersections, ne pas avoir de branches redondantes, etc) et de simplifier ainsi les traitements ultérieurs des objets.

- L'expression booléenne (b) : les volumes initiaux sont les opérandes et les actions sont les opérateurs. L'évaluation de l'expression donne l'objet ainsi décrit. En appliquant des règles de réécriture, il est possible d'obtenir des expressions normales disjonctives, ce qui correspond à la forme union de groupes d'intersections pour un arbre.

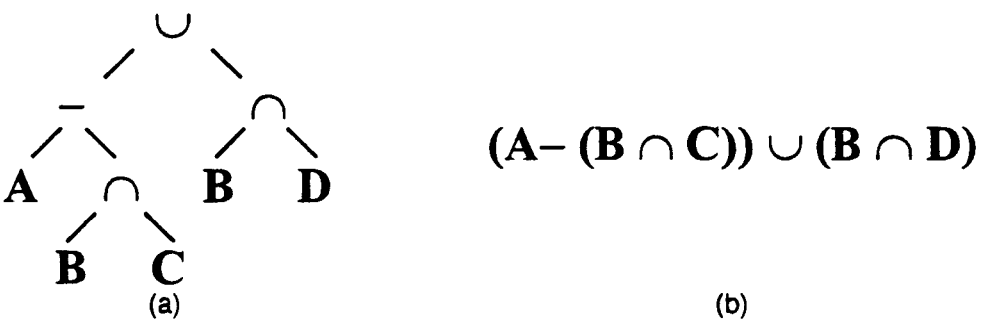


Figure 1.3 : Deux formes de représentation pour un objet

Lors de la modélisation en C.S.G., la difficulté n'est pas la topologie de l'objet que l'on cherche à construire, mais plutôt le positionnement dans l'espace des différents volumes initiaux. En effet les deux formes de représentation utilisées indiquent explicitement la topologie, mais ne donnent pas d'indication quant à la position des volumes les uns par rapport aux autres. En outre, les volumes initiaux sont bien souvent infinis dans plusieurs directions. De ce fait, certaines simplifications, comme la suppression de certaines opérations lorsque deux objets sont disjoints, obligent à l'emploi de boîtes englobantes comme informations supplémentaires.

- La facettisation des objets C.S.G.

Pour le rendu géométrique, un objet défini en C.S.G. doit être converti en un ensemble de facettes, qui approche la surface de l'objet.

La première méthode (Figure 1.4 a) consiste à calculer les intersections entre les différentes surfaces des volumes de base, afin d'obtenir les bords de l'objets. On facettise ensuite l'objet en fonction de ses bords et des surfaces des volumes de base : la facettisation est fonction de l'objet construit. Cette méthode n'est possible que si les volumes initiaux sont simples, les calculs d'intersection amenant très

rapidement à résoudre des systèmes complexes (par exemple, l'intersection de deux surfaces décrites par des équations du second degré, est une courbe donnée par une équation de degré quatre).

La seconde méthode (Figure 1.4 b) utilise une facettisation des volumes de base, afin d'éviter le calcul d'intersection complexe. La facettisation de l'objet complet consiste alors à supprimer les facettes ou les morceaux de facettes approchant les volumes de base et n'appartenant pas à la surface du volume final.

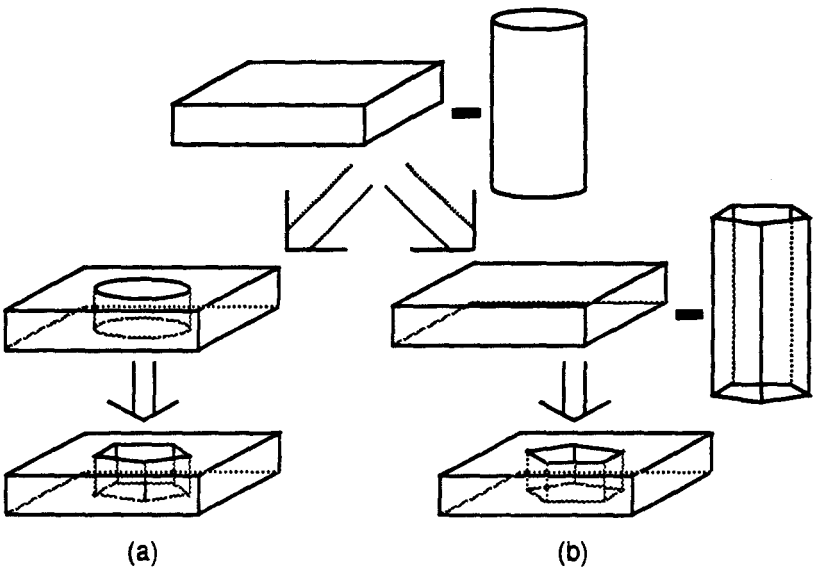


Figure 1.4 : Deux approches de la facettisation d'objet C.S.G.

Chaque objet C.S.G. est donc représenté pour l'affichage par un ensemble de facettes. Les facettes étant attachées par leurs bords puisqu'elles définissent un volume, l'objet est défini suivant un modèle de représentation par les bords (B-rep).

L'évolution des techniques de fabrication montre les limites du C.S.G. :

- La première de ces limites est atteinte par la découpe libre du bloc initial, car le bloc de matière enlevé n'est plus alors défini par un ensemble de volumes simples. Le volume doit être remplacé par une description de son contour à l'aide de surfaces permettant de décrire des volumes complexes.
- La seconde limite est que toutes les opérations ne sont pas réalisables : la déformation d'une pièce lors de l'emboutissage n'est pas représentable en C.S.G. L'utilisation des surfaces amène également une solution puisque l'on peut contrôler la déformation des surfaces décrivant les objets.

Le modèle surfacique

Contrairement au C.S.G., un objet est représenté lors de la modélisation surfacique, par l'ensemble des surfaces délimitant son volume [Mill86]. La définition de surfaces et leurs emplois permettent d'obtenir des objets complexes en limitant la complexité topologique. De plus ces surfaces sont déjà des "objets" limités dans l'espace, ce qui facilite la connaissance de leurs positions.

- La définition d'un objet

Différents types de surfaces paramétriques sont employés en modélisation suivant les propriétés que l'on veut obtenir, la possibilité de raccorder deux surfaces en tangence, de déformer une surface localement ou globalement,...

Une primitive très utilisée en modélisation est le patch de Bézier triangulaire [Fari92] défini par la donnée de points de contrôle que l'on peut déplacer afin de modifier la surface. Cette primitive est une généralisation des courbes de Bézier à l'espace.

Un courbe de Bézier de degré n est une courbe paramétrique définie par la donnée de $n+1$ points de contrôle $b_0, b_1, ..., b_n$. Un point de la courbe $b(t)$ est calculé à partir de son paramètre t par :

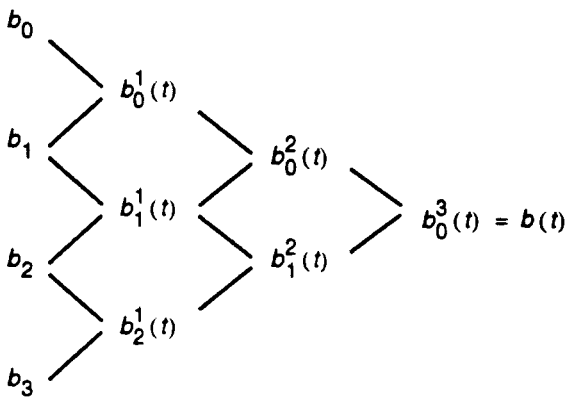
$$b(t) = \sum_{i=0}^{n} B_i^n(t)\, b_i \text{ avec } B_i^n(t) = \frac{n!}{i!\,(n-i)!} t^i (1-t)^{n-i} \text{ et } t \in [0,1]$$

Une méthode récursive de calcul des points en fonction du paramètre a été définie par De Casteljau:

$$b_i^r(t) = (1-t)\, b_i^{r-1}(t) + t\, b_{i+1}^{r-1}(t) \quad \begin{cases} r = 1, ..., n \\ i = 1, ..., n-r \end{cases}$$

avec $b_i^0(t) = b_i$. Nous obtenons alors $b_0^n(t)$, le point de la courbe de Bézier dont le paramètre est t.

Par exemple, pour une courbe de Bézier cubique, on calcule le point $b(t)$ en définissant successivement les points $b_i^r(t)$:



Dans le cas tridimensionnel des patchs triangulaires, une surface de degré n est construite à partir de $\frac{1}{2}(n+1)(n+2)$ points de contrôle notés $b_{i,j,k}$ avec $i+j+k = n$ et $i, j, k \geq 0$. La méthode de De Casteljau du calcul récursif d'un point dans le cas monodimensionnel, s'étend alors facilement au cas tridimensionnel : un point $b(t) = b_{0,0,0}^n(t)$ est construit par la définition successive des points $b_I^r(t)$ définis par :

$$b_I^r(t) = u\, b_{I+e1}^{r-1}(t) + v\, b_{I+e2}^{r-1}(t) + w\, b_{I+e3}^{r-1}(t)$$

avec $t = (u, v, w)$, $I = i, j, k$ telque $i+j+k = n-r$, et $e1 = 1,0,0$, $e2 = 0,1,0$, $e3 = 0,0,1$. Les points $b_I^0(t)$ sont les points de contrôles.

Si l'on prend l'exemple d'un patch de Bézier quadratique, défini par six points de contrôle, le point $b\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ est construit à partir de :

$$b_{0,0,1}^1 = \frac{1}{3} b_{1,0,1}^0 + \frac{1}{3} b_{0,1,1}^0 + \frac{1}{3} b_{0,0,2}^0$$

$$b_{0,1,0}^1 = \frac{1}{3} b_{1,1,0}^0 + \frac{1}{3} b_{0,2,0}^0 + \frac{1}{3} b_{0,1,1}^0$$

$$b_{1,0,0}^1 = \frac{1}{3} b_{2,0,0}^0 + \frac{1}{3} b_{1,1,0}^0 + \frac{1}{3} b_{1,0,1}^0$$

par

$$b^2_{0,0,0} = \frac{1}{3}b^1_{1,0,0} + \frac{1}{3}b^1_{0,1,0} + \frac{1}{3}b^1_{0,0,1} = b\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$$

- La facettisation des surfaces

Les primitives utilisées en modélisation sont généralement simples à facettiser parce qu'elles sont paramétriques : il suffit de choisir des points définis par la division de l'espace des paramètres pour obtenir un réseau de points et donc de triangles. Des méthodes spécifiques suivant le type des surfaces, ont par ailleurs été développées afin d'obtenir une facettisation dont on peut contrôler l'erreur d'approximation.

Si l'on reprend l'exemple d'un patch de Bézier triangulaire, les points de contrôle de la surface forment un réseau de triangles, qui approchent la surface. Si l'on désire augmenter le nombre de triangles pour l'affichage, afin de diminuer l'erreur de l'approximation, on subdivise la surface en sous-surfaces ayant elles aussi des points de contrôle. Les nouveaux points de contrôle des sous-surfaces forment alors un réseau de triangles plus fin qui approche mieux la surface initiale. Cette méthode est itérative (les sous-surfaces peuvent à leur tour être subdivisées) et le réseau formé par les nouveaux points de contrôle défini à chaque itération, converge rapidement vers la surface.

Pour la subdivision d'un patch de Bézier triangulaire, on emploie un point $b(t_1)$ (en général $t_1 = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$) défini par la méthode de De Casteljau, en calculant successivement des points $b^r_i(t_1)$. Le point $b(t_1)$ permet de diviser la surface initiale en trois sous-surfaces, dont les nouveaux points de contrôle sont les points initiaux et les autres points calculés dans la méthode de De Casteljau. L'espace des paramètres des points d'une sous-surface peut être projeté dans l'espace de paramétrage initial ($t \rightarrow t'$) et le processus de division peut alors être réitéré en calculant le point $b(t'_1)$.

L'exemple ci-dessus permet ainsi de déterminer "rapidement" un réseau de facettes approchant un patch de Bézier triangulaire, tout en contrôlant le niveau de facettisation.

Une objet est alors approché par une facettisation successive de toutes les surfaces utilisées pour le construire.

La modélisation surfacique permet d'obtenir des objets complexes, mais elle reste d'une manipulation difficile et semble inadaptée à la représentation de certains objets, qui peuvent être décrits simplement à partir de volumes primaires.

1.2 Le rendu des facettes

Les opérations nécessaires pour afficher une facette sont les suivantes :

- le positionnement définit l'objet dans le repère d'affichage.

- la conversion en pixel détermine les points de l'écran où la facette est présente.

- l'élimination des parties cachées permet de sélectionner l'objet ou la partie d'objet visible lorsque plusieurs facettes se recouvrent

- l'éclairement détermine la couleur d'un objet en fonction de différents paramètres, comme la position des sources de lumière, les propriétés lumineuses de l'objet (réfraction, diffraction, ...).

1.2.1 Le positionnement des facettes

Un objet est généralement défini dans un repère global (le repère du monde), qui représente l'espace dans lequel une scène est construite. Ce n'est que lors de la visualisation que l'on place le point d'observation (qui est la représentation du système de visualisation composé de l'écran d'affichage et de l'observateur), définissant ainsi ce qui est perçu dans l'image que l'on veut afficher.

Pour le rendu, la position de l'objet dans le repère d'affichage est définie par un changement de repère du repère global vers le repère du système de visualisation. En général, afin d'obtenir une image finale plus réaliste, on applique une perspective sur les objets visualisés (Figure 1.5).

Le changement de repère

Cette étape est définie par une matrice M_r de dimension 4x4, qui permet le changement de repère du monde vers le repère du système de visualisation dans le système de coordonnées homogènes[1].

Un point $P(x, y, z)$ de l'espace global est transformé par la matrice de changement de repère M en un point $P'(x', y', z')$ par $P'_m = M_r P_m$ avec P_m et P'_m la représentation en coordonnées homogènes des points P et P'.

Si l'on effectue ce changement de repère sur les trois sommets d'une facette, on définit la facette dans le repère du système de visualisation.

La transformation perspective

Lors de la phase de conversion, que nous détaillons par la suite, les objets sont définis en chaque point par une projection orthographique, ce qui revient à dire que l'observateur est placé à l'infini par rapport à l'écran de visualisation. Un point $P'(x', y', z')$ est alors projeté au pixel $p(x', y')$.

Si l'on veut augmenter le réalisme des images, il faut utiliser une projection perspective, qui définit la déformation dûe au système de visualisation. Un point $P'(x', y', z')$ est alors vu au pixel $p_p(x_p, y_p)$ défini par:

$$x_p = \frac{Dx'}{D+z'} \text{ et } y_p = \frac{Dy'}{D+z'}$$

avec D le distance entre l'observateur et l'écran.

Pour la conversion, un point $P'(x', y', z')$ est transformé en un point $P_p(x_p, y_p, z_p)$ défini par une transformation perspective, la définition de la valeur $z_p = \frac{Dz'}{D+z'}$ est nécessaire pour la phase d'élimination des parties cachées.

La composition de la transformation perspective et de la projection orthographique définit une projection perspective pour l'affichage.

1. Les coordonnées homogènes définissent la projection d'un point dans un espace vectoriel à quatre dimensions, afin d'effectuer les mouvements de translation et de rotation sur un point par une opération matricielle.

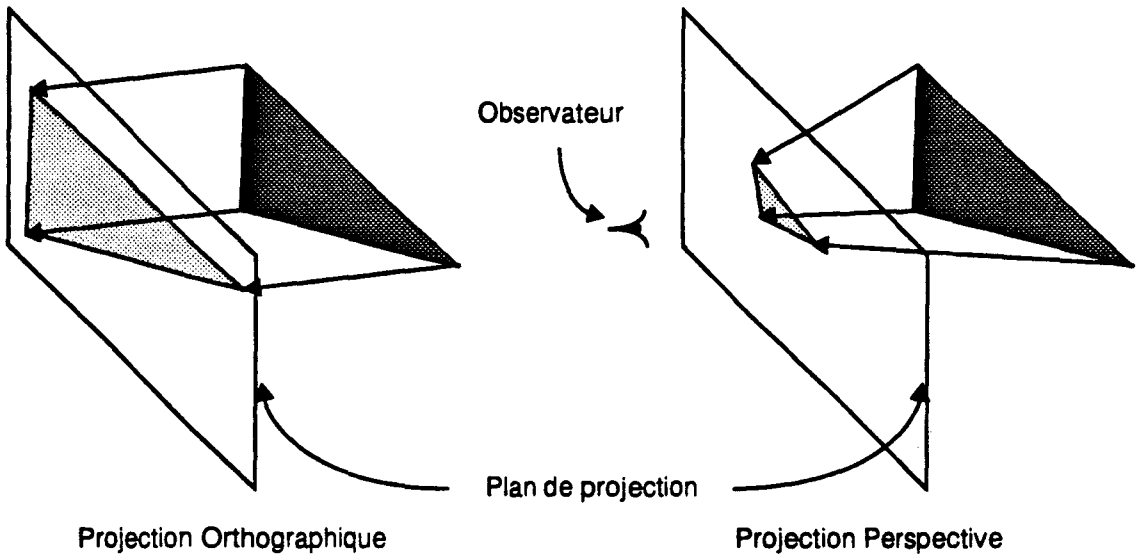


Figure 1.5 : Les deux types de projection

La transformation perspective peut également être appliquée par un produit matriciel : le point P est transformé en P' par $P'_m = M_p P_m$.

avec $M_p = \begin{bmatrix} D & 0 & 0 & 0 \\ 0 & D & 0 & 0 \\ 0 & 0 & D & 0 \\ 0 & 0 & 1 & D \end{bmatrix}$ et P'_m, P_m, les coordonnées homogènes des deux points.

Le changement de repère de l'espace global vers le repère du système de visualisation avec la transformation perspective peut alors être défini par une seule matrice $M_{pr} = M_p M_r$; un point P de l'espace global est transformé directement en un point P' par $P'_m = M_{pr} P_m$.

En appliquant la matrice M_{pr} sur chacun des sommets d'une facette, on obtient la position de la facette dans le repère de visualisation.

1.2.2 La conversion

La facette étant placée dans le repère du système de visualisation, il faut déterminer les pixels où elle est présente.

La première approche est de partir de l'objet pour définir les pixels : c'est la méthode du remplissage par suivi de contour.

La seconde approche est de tester en chaque pixel si l'objet est présent : c'est la méthode par expressions.

Le remplissage par suivi de contour

Le principe de cette méthode est d'utiliser la géométrie de l'objet pour définir les pixels où il est présent [Hegr85].

Sur une ligne y_i, les pixels appartenant à une facette sont les points situés sur un segment entre le point

du bord gauche $P_{ig}(x_{ig}, y_i)$ et le point du bord droit $P_{id}(x_{id}, y_i)$ de la facette, ces deux points étant les intersections des côtés gauche et droit de la facette avec la ligne y_i. Les côtés gauche et droit d'une facette étant deux segments de droites, les points gauche et droit sont alors déterminés par :

$$x_{ig} = -\frac{b_g}{a_g}y_i - \frac{c_g}{a_g} \text{ et } x_{id} = -\frac{b_d}{a_d}y_i - \frac{c_d}{a_d}$$

avec $a_g x + b_g y + c_g = 0$ et $a_d x + b_d y + c_d = 0$, les équations des droites des côtés.

Si l'on se trouve en une ligne y_i et que l'on connaît $P_{ig}(x_{ig}, y_i)$, alors le point $P_{(i+1)g}(x_{(i+1)g}, y_{i+1})$ d'extrémité gauche de la ligne y_{i+1} suivante peut être calculé par :

$$x_{(i+1)g} = -\frac{b_g}{a_g}y_{i+1} - \frac{c_g}{a_g} = -\frac{b_g}{a_g}y_i - \frac{b_g}{a_g} - \frac{c_g}{a_g} = x_{ig} - \frac{b_g}{a_g}$$

Par le même raisonnement, on obtient le point d'extrémité droit $P_{(i+1)d}(x_{(i+1)d}, y_{i+1})$ de la facette pour la ligne suivante, connaissant le point P_{id} par

$$x_{(i+1)d} = x_{id} - \frac{b_d}{a_d}$$

L'algorithme utilise comme point de départ le sommet $s1(x_{s1}, y_{s1})$ de la facette ayant la plus petite ordonnée (donc les points extrémités sont $P_{1g} = P_{1d} = s_1$), puis définit des segments de pixels successivement en chaque ligne, par le calcul incrémental des extrémités gauches et droites (Figure 1.6 a). Le conversion se termine lorsque le sommet $s3(x_{s3}, y_{s3})$ de la facette ayant la plus grande valeur d'ordonnées est atteint.

Lors de la conversion, il faut changer la valeur de l'incrément pour le calcul de l'extrémité gauche ou droite lorsque cette extrémité est le sommet $s2(x_{s2}, y_{s2})$ de la facette, puisque le côté concerné change de droite support.

La méthode par expressions

Dans la méthode par expressions, connaissant les coordonnées d'un pixel $pi(x_{pi}, y_{pi})$ on cherche à déterminer si la facette est présente en celui-ci.

La projection des bords d'une facette sur le plan de l'écran de visualisation sont trois segments de droite. Une droite définissant deux demi-plans, un pixel est dans la projection du contour de la facette, s'il est dans l'intersection des trois demi-plans contenant la projection de la facette.

Pour déterminer si un point $P(x_p, y_p)$ est dans l'un ou l'autre des demi-plans définis par une droite d'équation $f(x, y) = ax + by + c = 0$, il suffit d'évaluer l'expression $f(x_p, y_p)$: le résultat de cette évaluation donne un résultat positif pour un demi-espace et négatif pour l'autre.

Dans la méthode par expressions, tous les pixels de l'écran testent leur appartenance à une facette. On utilise les équations $f_1(x, y) = 0$, $f_2(x, y) = 0$ et $f_3(x, y) = 0$ des trois droites des côtés de la facette, pour déterminer trois expressions $f_1(x, y)$, $f_2(x, y)$, $f_3(x, y)$, qui déterminent six demi-plans. Ensuite, chaque pixel $pi(x_{pi}, y_{pi})$ évalue ces expressions pour ses coordonnées et teste les résultats, afin de savoir s'il est dans les trois demi-plans contenant la projection de la facette.

Par convention, on oriente les trois expressions pour obtenir un résultat positif lors de l'évaluations des expressions aux points intérieurs à la projection du contour de la facette. La facette est présente en un pixel $pi(x_{pi}, y_{pi})$, si $f_1(x_{pi}, y_{pi}) \geq 0$, $f_2(x_{pi}, y_{pi}) \geq 0$ et $f_3(x_{pi}, y_{pi}) \geq 0$ (Figure 1.6 b).

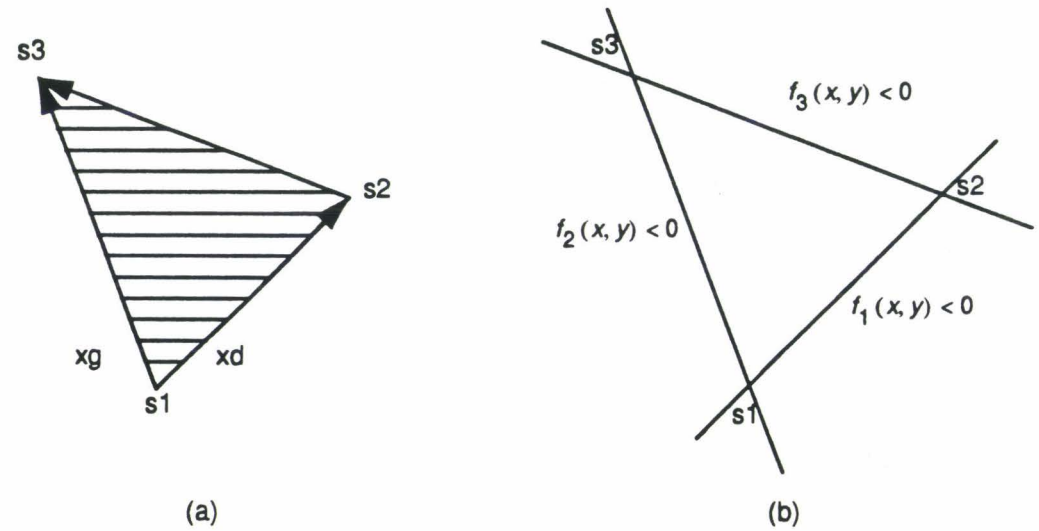


Figure 1.6 : Deux méthodes de conversion des facettes

1.2.3 L'élimination des parties cachées

Si plusieurs objets d'une scène sont présents en un même pixel, il faut savoir quel objet est réellement vu dans l'image finale ; on effectue une recherche des faces visibles par élimination des parties cachées. L'objet vu en un point donné d'une image est l'objet le plus près de l'observateur, donc ayant la plus petite valeur de profondeur au point.

Deux approches peuvent être employées :

- l'élimination des parties cachées est traitée avant la conversion des objets en pixels. Si un objet O_1 en recouvre un second O_2 alors les points de O_1 seront forcément devant ceux de O_2. Les pixels où est présent l'objet O_1 n'ont pas à être traités pour l'objet O_2.
- L'élimination des parties cachées est effectuée après la conversion en pixels. En tout pixel, on détermine l'objet le plus proche de l'observateur par un test sur les profondeurs des objets présents.

Le traitement avant la conversion

Pour effectuer l'élimination des parties cachées avant la conversion, il existe différentes méthodes, qui utilisent la géométrie de la facette. Les facettes étant planes, elles sont contenues dans des plans tridimensionnels, qui peuvent être utilisés pour déterminer la position des objets les uns par rapports aux autres.

Voici un exemple d'algorithme utilisant un découpage des facettes en segments, pour effectuer l'élimination des parties cachées. L'algorithme de subdivision "scanline", découpe l'espace du repère d'affichage en tranches horizontales. Chaque tranche étant un "plan de profondeur" contenant une ligne de pixels, on se ramène alors à un problème bidimensionnel, l'intersection d'un plan de profondeur et des facettes définissant un ensemble de segments (Figure 1.7 a).

L'algorithme se déroule ainsi :

- pour une ligne, on détermine pour chaque facette les extrémités gauche et droite des segments intersection de la face et du plan. On trie alors la liste des facettes dans le sens croissant des profondeurs

des extrémités gauches, afin de créer une liste de bords gauches. On effectue ensuite le même traitement sur les bords droits.

- si on obtient la même liste à gauche et à droite, alors les segments des facettes sont affichés dans l'ordre inverse d'une des deux listes, puisqu'il ne peut pas y avoir intersection entre deux facettes (propriété des segments de droites) (Figure 1.7 b).

- sinon on détermine un point d'intersection entre deux segments ou un point d'extrémité d'un segment, afin de découper la ligne de pixels en deux sous-lignes (Figure 1.7 c). On recommence alors le même traitement sur les deux sous-lignes. Le traitement est récursif et s'arrête pour une sous-ligne soit quand on obtient les mêmes listes à gauche et à droite, soit quand la sous-ligne est de la taille d'un pixel.

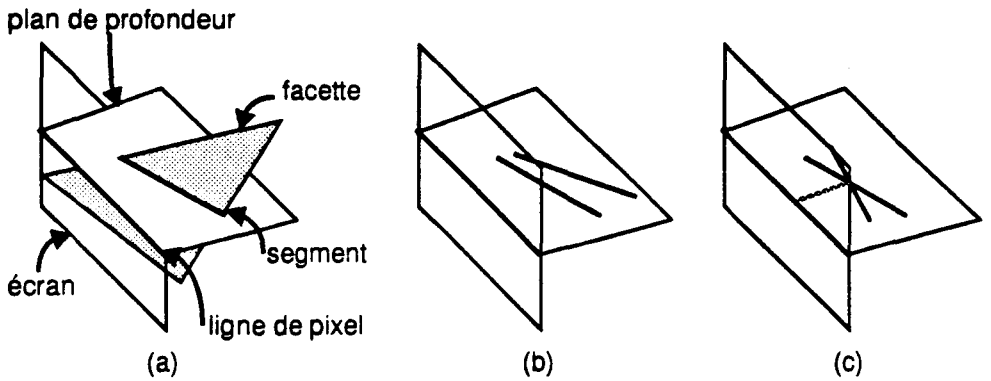


Figure 1.7 : L'algorithme de subdivision scan-line

Le traitement après la conversion en pixel

Le principe du tampon de profondeur (Z-buffer) est de déterminer en chaque pixel où une facette est présente, la profondeur du plan support de celle-ci. La facette vue en un pixel ayant la plus petite profondeur, si les facettes sont converties en pixels les unes après les autres, il suffit de mémoriser la profondeur de la facette visible en chaque pixel après la conversion d'un certain nombre de primitives, et d'effectuer la comparaison des profondeurs et éventuellement la mémorisation de la nouvelle profondeur, lorsqu'une nouvelle facette est convertie.

Algorithme du Z-buffer

```
Pour tous les pixels (i,j) de l'écran
    Z(i,j) =Zmax                    /*Zmax est la valeur de fond d'écran*/
FinPour
Pour chaque objet
    Pour chaque pixel (i,j)
        calculer Z0(i,j)            /*Profondeur de l'objet au pixel*/
        Si Z0(i,j) < Z(i,j)
            Z(i,j) = Z0(i,j)
            A(i,j) = A0(i,j)        /*Attributs de l'objet au pixel*/
        FinSi
    FinPour
FinPour
```

Une facette appartenant à un plan d'équation $ax+by+cz+d=0$, la profondeur de la facette en un pixel (x_{pi}, y_{pi}) est $Z(x_{pi}, y_{pi}) = -\frac{a}{c}x_{pi} - \frac{b}{c}y_{pi} - \frac{d}{c}$, qui est un expression du premier degré en (x, y). Cette expression est facile à évaluer en chaque pixel de manière incrémentale, puisque $Z(x+1, y) = Z(x, y) - \frac{a}{c}$ et $Z(x, y+1) = Z(x, y) - \frac{b}{c}$.

1.2.4 Les calculs d'éclairement

Les premières méthodes d'affichage de facette utilisaient un éclairage constant (flat shading), qui définissait une couleur unique sur toute la surface de la primitive (Figure 1.10 a). Or la perception d'un objet dépend surtout de la luminance qu'un observateur reçoit.

Des modèles d'éclairement simples ont alors été développés afin qu'un observateur perçoive la forme des objets. Parallèlement à la définition des modèles de calcul de l'éclairement, des méthodes ont été définies afin que les facettes représentant un objet courbe puissent simuler les variations de couleur sur sa surface, donnant ainsi l'impression à un observateur que ce ne sont pas des facettes qui sont affichées mais l'objet réel.

La méthode de Gouraud

Le premier modèle d'éclairement fut celui de Lambert, qui définit la couleur en un point d'après la normale $\vec{N}$ à la surface d'un objet et la direction des sources d'éclairement $\vec{L}_j$ en ce point (Figure 1.9). Il définit pour chaque couleur C, une intensité donnée par :

$$I = k_a I_a + \sum_{j=0}^{n} k_d I_j (\vec{N} \cdot \vec{L}_j)$$

avec i_a la luminosité ambiante, i_j l'intensité de la source j et k_a, k_d les caractéristiques de l'objet pour la couleur C.

Gouraud [Gour71] lui associa une méthode de calcul de l'intensité des couleurs des points d'une facette par interpolation bilinéaire des intensités I_{s1} I_{s2} I_{s3} aux sommets S_1 S_2 S_3 de la facette (Figure 1.8). L'interpolation utilisée permet de garantir la continuité C1 des intensités sur les côtés communs des facettes, il suffit donc de l'utiliser sur chaque facette pour avoir les intensités (r,v,b) en tout point.

Pour calculer l'intensité I_p d'une couleur C en un point $P(x, y)$, on utilise deux points Q et R, qui sont les extrémités gauche et droite de la facette pour la ligne y. pour lesquels on définit les intensités :

$$I_q = (1-t) I_{s1} + t I_{s2} \text{ avec } t = \frac{S_1 Q}{S_1 S_2}$$

$$I_r = (1-u) I_{s1} + u I_{s3} \text{ avec } u = \frac{S_1 R}{S_1 S_3}$$

L'intensité au point P est alors

$$I_p = (1-w) I_q + w I_r \text{ avec } w = \frac{QP}{QR}$$

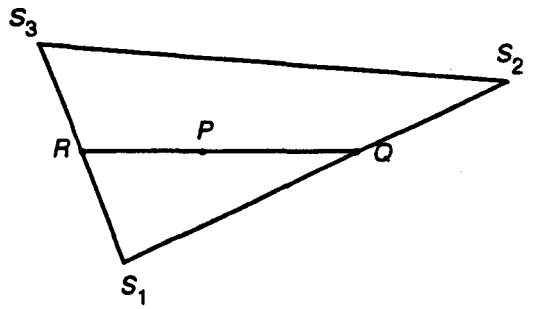


Figure 1.8 : L'interpolation bilinéaire des couleurs

Lorsqu'un réseau de facettes approche une surface, les normales aux sommets des facettes ayant été définies lors de la facettisation, on calcule l'intensité des couleurs rouge, verte et bleue (qui sont les couleurs primaires de l'afficheur) aux sommets et on utilise ensuite l'interpolation pour créer l'illusion d'afficher une surface courbe (Figure 1.10 b).

Remarque : Il est possible d'exprimer l'interpolation à l'aide d'une expression du premier degré en (x, y), qui définit alors un plan d'intensité pour chaque couleur. L'intensité en un pixel est alors définie directement par l'évaluation de cette expression.

La méthode de Phong

Le modèle de Lambert ne permet de visualiser que l'aspect diffus de lumière, les objets ont alors un aspect mat. Phong[Phon75] a développé un modèle intégrant la réflexion des surfaces, afin de visualiser les effets spéculaires sur les objets, leur rendant un aspect ponctuel plus brillant. Le modèle de Phong a repris les composantes du modèle de Lambert en lui associant une valeur d'intensité spéculaire dépendant de la position de l'observateur par rapport à l'objet.

L'intensité de la couleur est alors définie par :

$$I = k_a I_a + \sum_{j=0}^{n} \left(k_d I_j (\vec{N} \cdot \vec{L}_j) + k_s I_j (\vec{O} \cdot \vec{R}_j)^n \right)$$

avec $\vec{R}_j$ le vecteur symétrique de $\vec{L}_j$ par rapport à la normale $\vec{N}$, et $\vec{O}$ la direction de l'observateur (Figure 1.9).

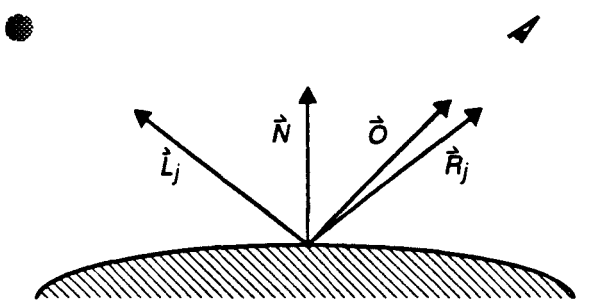


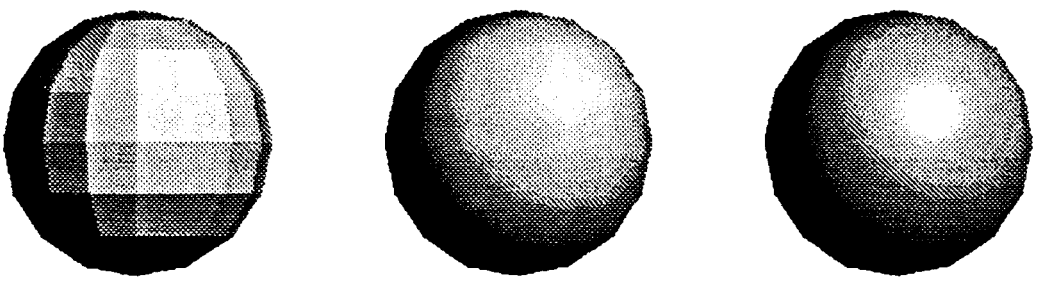
Figure 1.9 : Les vecteurs de l'éclairement

L'utilisation du modèle d'éclairement de Phong avec la méthode de rendu de facettes de Gouraud pose

des problèmes, puisque l'effet spéculaire n'est pas toujours bien perçu :

- si les trois sommets d'une facette ont la même intensité pour les couleurs r,v,b, alors la couleur est constante en tout point de la surface. Lorsque plusieurs facettes représentent un objet courbe, si l'une d'elles a cette caractéristique, elle est perçu comme étant un triangle plat, ce qui nuit à la représentation de l'objet.
- l'interpolation des intensités fait apparaître dans certains cas, un effet spéculaire différent de celui normalement perçu (plus large ou plus petit).

Phong a également développé une méthode de rendu des facettes associée à son modèle d'éclairement. Il propose d'interpoler les normales au sommet au lieu des intensités des couleurs, les calculs d'éclairement étant effectués en chaque point. L'effet spéculaire est alors mieux rendu lorsque l'on cherche à représenter l'aspect courbe d'une surface approchée par un réseau de facettes (Figure 1.10 c).



(a) Eclairement Constant (b) Interpolation de Gouraud (c) Interpolation de Phong

Figure 1.10 : Les méthodes d'éclairement des facettes

1.3 Organisations des machines de rendu géométrique de facettes

La définition de matériel pour le rendu des facettes oblige à effectuer des choix au niveau des algorithmes employés et au niveau du parallélisme employé dans la définition de la machine. Le lecteur intéressé pourra trouver une description détaillée des machines de rendu de facettes dans [Fole90][Luca91][Chai92]

1.3.1 Les pipelines de rendu géométrique de facettes

Le positionnement des facettes est le traitement qu'il faut effectuer en premier, afin de pouvoir les afficher. L'ordre dans lequel les autres traitements sont effectués dépend des algorithmes choisis.

Lors de la conversion, les machines effectuent également le calcul de la profondeur en chaque pixel et l'interpolation bilinéaire de trois valeurs (intensités des couleurs r,v,b ou composantes de la normale). Suivant que la conversion utilise le remplissage par suivi de contour ou la méthode par expressions, la profondeur et les valeurs interpolées sont définies à l'aide d'une méthode incrémentale ou par une expression du premier degré.

Actuellement, les machines de rendu géométrique utilisent pour la plupart l'algorithme d'élimination des parties cachées du Z-buffer, parce qu'il est très simple à implanter matériellement et que les deux méthodes d'éclairement obligent le traitement de tous les pixels d'une facette pour le calcul de la couleur. Cet algorithme devant s'effectuer après la conversion, sa position dans le déroulement de l'affichage est fixé.

Suivant qu'une machine utilise la méthode d'éclairement des facettes de Gouraud ou de Phong, on obtient deux déroulements possibles pour la conversion d'une facette par une machine de rendu. La séparation des traitements lors de l'affichage permet alors d'obtenir un effet de pipeline.

Le pipeline de type Gouraud

L'organisation de l'affichage utilisant le rendu de facette par la méthode de Gouraud, est définie par les quatre étapes suivantes (Figure 1.11) :

- la première étape est le positionnement de la facette dans le repère de visualisation.
- la seconde étape consiste à calculer la couleur en chaque sommet des facettes.
- dans la troisième étape, la facette est convertie en pixels, pour lesquels on calcule la couleur et la profondeur du plan support de la facette par interpolation.
- la quatrième étape permet alors d'effectuer l'élimination des parties cachées par Z-buffer avant la visualisation sur l'écran.

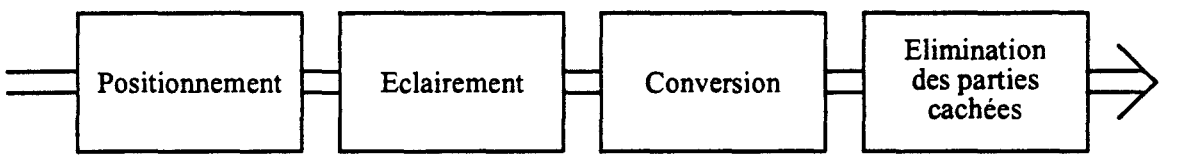


Figure 1.11 : le pipeline de Gouraud

Le pipeline de type Phong

L'utilisation de la méthode de Phong pour le rendu des facettes, impose un ordre d'exécution des traitements différents (Figure 1.12) :

- comme pour le pipeline de Gouraud, la première étape est le positionnement de la facette dans le repère de visualisation.
- la seconde étape est la conversion de la facette en pixels, qui comprend dans ce cas le calcul de la profondeur du plan support et l'interpolation des normales aux sommets
- vient ensuite l'élimination des parties cachées dans la troisième étape
- les calculs d'éclairement sont effectués dans la quatrième étape, avant la visualisation sur l'écran.

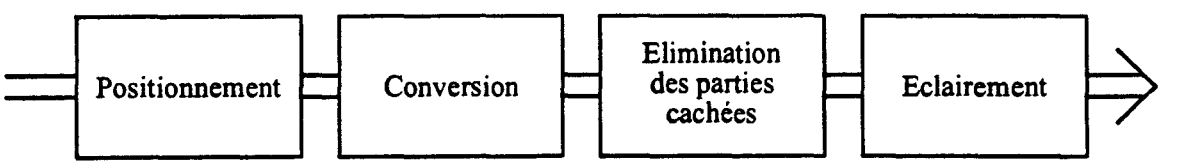


Figure 1.12 : Le pipeline de Phong

Les machines actuelles utilisent généralement le pipeline de Gouraud, puisque les calculs d'éclairement ne sont effectués que sur les trois sommets des facettes avant la conversion, et non en tous pixels comme lors de l'utilisation de la méthode de rendu de Phong.

Cependant certaines recherches s'orientent vers l'utilisation de la méthode d'interpolation de Phong lors du rendu des facettes [Deer88] [Schn88]. En effet, l'interpolation des normales aux sommets des facettes permet d'obtenir des images de meilleure qualité que celles affichées avec la méthode de rendu de Gouraud.

L'éclairement des objets après la conversion (post-éclairement) augmente la somme de calculs à effectuer. Il faut en effet évaluer l'intensité en chaque pixel d'après la formule d'éclairement de Phong [Phon75] avec des vecteurs normalisés :

$$I = k_a I_a + \sum_{j=0}^{n} \left(k_d I_j (\vec{N} \cdot \vec{L}_j) + k_s I_j (\vec{O} \cdot \vec{R}_j)^n \right)$$

avec I_j et $\vec{L}_j$ l'intensité et la direction de la source j, $\vec{N}$ la normale à la surface $\vec{R}_j$ le vecteur réfléchi de la source j par rapport à la surface et $\vec{O}$ la direction de l'observateur (les termes k_a, k_d, k_s et I_a sont constants).

En fait, le coût du calcul se situe principalement à deux niveaux :

- la normalisation des vecteurs $\vec{L}_j$, $\vec{N}$, $\vec{O}$ utilisés dans les calculs de produits scalaires (si $\vec{L}_j$ et $\vec{N}$ sont normés alors $\vec{R}_j$ l'est aussi).
- l'élévation à la puissance pour l'évaluation du terme spéculaire.

Afin de permettre l'utilisation du modèle de Phong, des processeurs spécifiques ont été définis pour permettre d'effectuer la normalisation de vecteurs [Knit93], ou plus globalement l'éclairement complet des pixels [Deer88] [Clau91]. La réalisation d'un module de post-éclairement est envisageable [Lefe92][Lefe94], surtout si l'on considère la progression rapide de la technologie employée dans les processeurs actuels.

La réalisation d'un module de post-éclairement des objets est chère, mais l'éclairement des pixels étant effectué après élimination des parties cachées, les calculs inutiles sont supprimés. De plus, le principe du post-éclairement peut s'appliquer à toute primitive d'affichage, dont on connaît la normale en chaque pixel. Il est donc possible d'utiliser différentes primitives en conversion, tout en gardant le même module d'éclairement. Par opposition, le modèle de Gouraud est spécifique au rendu des facettes.

1.3.2 La parallélisation de l'affichage

Pour accroître le nombre d'objets affichés tout en gardant une vitesse de rendu rapide, plusieurs types de parallélisme sont employés [Luca91][Karp93]. Nous allons voir les exemples des deux principaux types de parallélisme employés par les machines de rendu géométrique.

Le premier est employé pour la conversion des objets en pixels, car ce traitement est coûteux puisqu'en chaque pixel il faut déterminer si la facette est présente, déterminer une profondeur et évaluer trois valeurs obtenues par interpolation bilinéaire.

Le second est utilisé à un niveau plus global, celui de l'image. Il consiste à diviser l'écran en zones et à générer en parallèle les différentes zones. Une unité de conversion, constituée de plusieurs processeurs de conversion, s'occupe uniquement des objets situés dans sa zone.

Le parallélisme de conversion

Lorsque l'on emploie un grand nombre de processeurs pour la conversion, on peut choisir de les attacher à un pixel ou à un objet [Meri84][Chai91].

- Le parallélisme Pixel

En associant un processeur par pixel, on obtient une grille fixe mais régulière de processeurs, ce qui facilite l'organisation architecturale de la machine.

Les pixels étant fixes, ce sont les objets qui traversent la grille formant ainsi un flot. Le temps de conversion est alors une fonction linéaire du nombre d'objets.

L'utilisation de ce parallélisme impose que tous les processeurs effectuent les mêmes traitements sur les pixels, les processeurs ont tous la même complexité matérielle. Dans le cas de la conversion de primitives facettes uniquement, les processeurs sont relativement simples; l'emploi d'un grand nombre de processeurs est donc possible.

- Le parallélisme Objet

Un processeur étant associé à un objet, l'organisation des processeurs est modulaire : la structure et la complexité matérielle d'une machine sont fonction du nombre et du type de processeurs utilisés.

Un processeur traite tous les pixels d'un objet et crée alors un flot de pixels traversant le système. Le temps de conversion dépend du ratio $\frac{\text{nombre d'objets}}{\text{nombre de processeurs}}$, qui définit une fonction constante par paliers lorsque le nombre d'objets varie.

Une machine utilisant ce type de parallélisme peut très facilement convertir différents types d'objets.

La parallélisation du traitement de l'image

La parallélisation du traitement de l'image nécessite de découper l'écran en zones. L'allocation des zones pouvant être statique ou dynamique, différents types d'organisation des unités de conversion sont possibles. Un écran étant un ensemble rectangulaire de pixels, on peut choisir de le découper en lignes (horizontales ou verticales) ou en zones rectangulaires plus petites. Afin d'estimer le nombre d'objets réellement traités par l'ensemble des unités de conversion, on utilise le facteur de duplication [Karp93], qui représente le nombre moyen théorique de zones recouvertes par un objet lors de l'affichage.

L'efficacité des unités de conversion est accrue, mais les facettes de la base de données de la scène doivent être triées en fonction du découpage choisi. Pour chaque primitive, il faut en effet déterminer les zones où elle est présente, afin de limiter le nombre de pixels inutilement traités.

- Organisation du parallélisme

Dans le cas d'une allocation statique, le nombre d'unités de conversion doit être égal au nombre de zones obtenues après découpage, une unité de conversion s'occupant d'une seule zone : le parallélisme (ou la réalisation) est total(e) (Figure 1.13 a).

Lors d'une allocation dynamique, une unité de conversion peut s'occuper de plusieurs zones : le parallélisme est partiel (ou encore la réalisation est partielle), le nombre d'unités de conversion pouvant être inférieur au nombre de zones. Tout au long de la création de l'image, dès qu'une unité de conversion a fini de convertir les objets de sa zone, on lui attribue une nouvelle zone. Lorsqu'une machine possède une seule unité de conversion, celle-ci doit traiter les zones les unes après les autres, de manière séquentielle ; on obtient alors une réalisation partielle séquentielle (Figure 1.13 b). Si plusieurs unités de conversion travaillent en parallèle, on obtient une réalisation partielle parallèle (Figure 1.13 c).

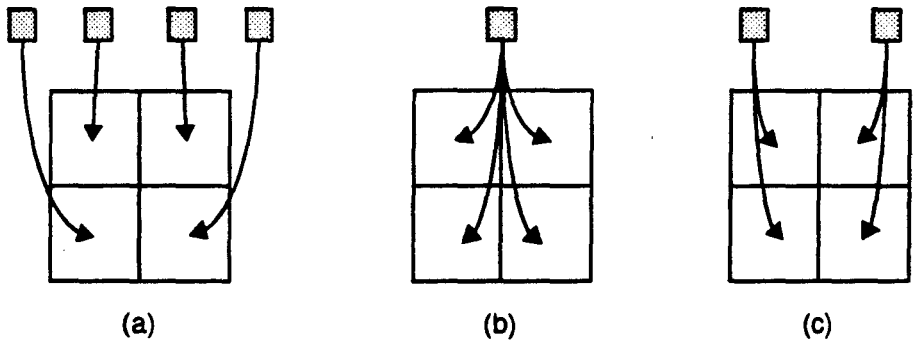


Figure 1.13 : L'organisation des unités de conversion

• Les lignes écran

Lorsque l'on découpe un écran en lignes, on peut choisir deux modes de fonctionnement des unités de conversion :

- Une unité de conversion traite une seule ligne à la fois. La conversion d'une facette sur une ligne devient alors le traitement d'un segment de droite horizontal (horizontal span). L'unité de conversion utilise des processeurs spécialisés pour la conversion de ces segments. Le facteur de duplication obtenu est la taille moyenne des facettes par rapport au sens de découpage.

- Une unité de conversion traite plusieurs lignes. La répartition des lignes étant régulière, les lignes des différentes unités de conversion sont entrelacées. Les lignes allouées à une unité de conversion forment aussi un rectangle, à un changement d'échelle près : les objets convertis sont également des facettes. Le facteur de duplication est la taille moyenne des facettes dans le sens du découpage après le changement d'échelle.

Pour déterminer les lignes où une facette est présente, on trie les sommets et on détermine les extremums par rapport au sens du découpage. Les coordonnées des deux sommets obtenus donnent alors l'intervalle de lignes à traiter.

Le remplissage par suivi de contour est une méthode de conversion très bien adaptée au découpage de l'écran en ligne, puisqu'il permet de définir directement les segments des lignes à partir des facettes.

• Les zones écran

Le traitement des objets sur une zone écran rectangulaire est effectué de la même manière que sur l'écran complet. L'efficacité d'une unité de conversion augmente, puisque le nombre de pixels traités inutilement est plus petit : les objets n'étant pas présents dans une zone ne sont pas traités.

Le facteur de duplication pour un découpage en zones rectangulaire est calculé à partir de la définition de la boîte écran d'une facette. La boîte écran étant la boîte englobante rectangulaire de la facette après projection sur l'écran, elle est définie par ses quatre sommets $C_1(x_{min}, y_{min})$, $C_2(x_{max}, y_{min})$, $C_3(x_{max}, y_{max})$ et $C_4(x_{min}, y_{max})$ avec x_{min}, x_{max}, Y_{min}, y_{max} les valeurs minimales et maximales des abscisses et de ordonnées des trois sommets de la facette.

Le facteur de duplication est alors [Moln91] : $FD = \frac{(l+L)(h+H)}{LH}$

avec $l \times h$ la taille moyenne des boîtes écran des facettes et $L \times H$ la taille des zones.

Les zones sont définies par un découpage régulier suivant les deux axes x et y, avec un pas K_x et K_y. Pour

déterminer les zones où une facette est probablement présente (Figure 1.14), on effectue un pré-tri, connaissant la boîte écran de la primitive.

On calcule en premier :

$$zone_{xmin} = trunc\left(\frac{x_{min}}{K_x}\right) \qquad zone_{ymin} = trunc\left(\frac{y_{min}}{K_y}\right)$$

$$zone_{xmax} = trunc\left(\frac{X_{max}}{K_x}\right) \qquad zone_{ymax} = trunc\left(\frac{Y_{max}}{K_y}\right)$$

Les zones où la facette est probablement présente sont ensuite définies par :

$$zone_{xmin} \leq X_{depart} \leq zone_{xmax}$$
$$zone_{ymin} \leq Y_{depart} \leq zone_{ymax}$$

avec X_{depart} et Y_{depart} les coordonnées du point d'entrée de la zone.

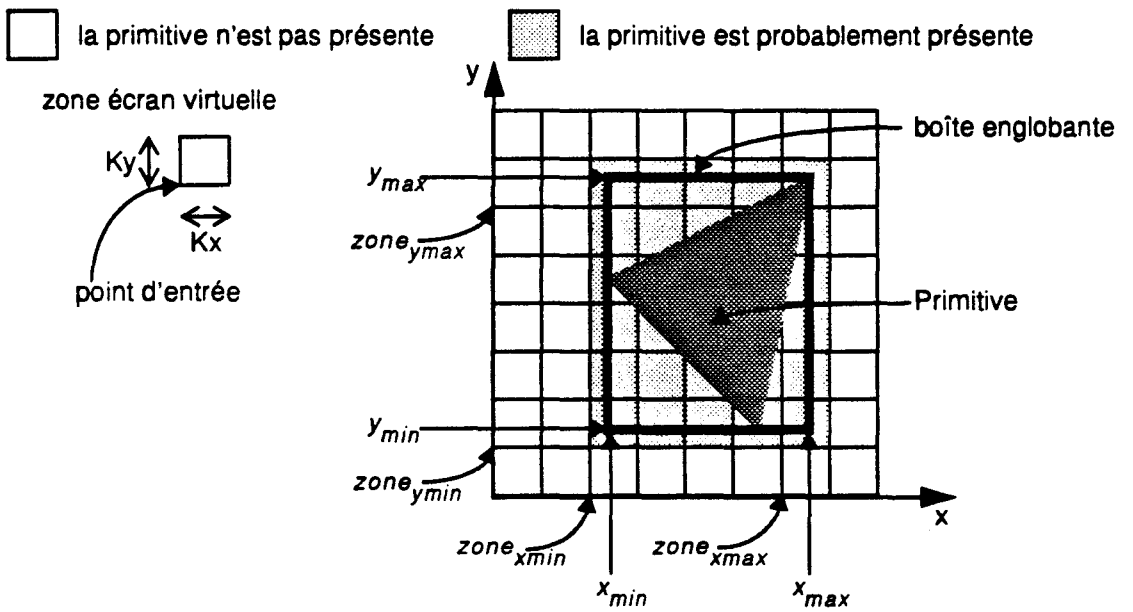


Figure 1.14 : Définition des zones écran occupées par une primitive

La méthode du remplissage par suivi de contour n'est pas adaptée au découpage d'un écran en zone pour deux raisons (Figure 1.15) :

- le clipping d'une facette triangulaire par les cotés d'une zone rectangulaire, génère une facette à plus de trois côtés dans la plupart des cas.

- le passage d'une zone à la suivante oblige, toujours dans le cadre du remplissage par suivi de contour, à recalculer les points de départ et d'arrivée de la facette pour la nouvelle zone.

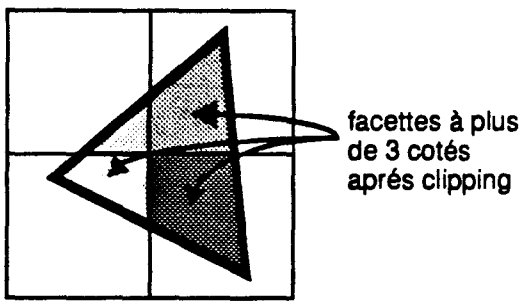


Figure 1.15 : Clipping d'une facette pour un découpage par zones

Sylvain Karpf [Karp93] a montré dans sa thèse que le meilleur découpage d'un écran pour la parallélisation au niveau de l'image est la définition de zones rectangulaires, puisque cette méthode permet de réduire le taux de duplication des primitives à afficher. Le nombre de primitives effectivement traitées étant plus petit, les performances réelles de la machine d'affichage sont augmentées.

Le découpage de l'écran est utilisé pour paralléliser les machines de rendu, mais il nécessite de trier les primitives et de positionner les primitives dans chaque zone où elles sont présentes. Nous étudions dans le chapitre 5, le coût non négligeable de ces opérations.

1.4 Vers des primitives de plus haut niveau

Afin d'améliorer la qualité des images générées, certaines équipes de recherche ont essayé d'utiliser des primitives de plus haut niveau que la facette plane lors de la phase du rendu. L'affichage de facettes utilisant des expressions du premier degré, les recherches se sont portées vers l'utilisation d'expressions du second degré. Ces expressions ont été employées au niveau du rendu des facettes [Kirk90] et de la conversion des objets [Eyle88].

Dans le cadre de la conversion des objets à l'aide d'expressions du second degré, nous allons étudier les approximations de sphères et de cylindres développées par l'équipe de Chapel Hill. Nous présentons les algorithmes et leur implantation sur les machines Pixel-Planes 4 et Pixel-Planes 5.

1.4.1 L'origine des primitives sphères et cylindres

La plupart des modeleurs actuels intègrent dans leurs primitives de modélisation les sphères et les cylindres, afin de simplifier la construction d'une scène puisque ces éléments apparaissent souvent. Par exemple, lors de la description d'une molécule, on utilise des sphères qui représentent des atomes.

Les machines actuelles n'utilisent que la facette et le rendu de Gouraud pour l'affichage, les sphères et les cylindres définis lors de la phase de modélisation sont donc facettisés pour le rendu (Figure 1.16).

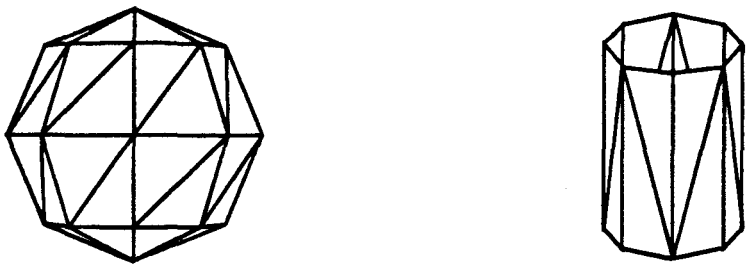


Figure 1.16 : La sphère et le cylindre facettisés

L'équipe de Pixel-Planes 4 a alors tout naturellement pensé à utiliser la sphère comme primitive d'affichage. Afin de réduire la complexité des calculs lors du rendu, la profondeur d'une sphère est approchée par une fonction du second degré en x et en y, $Z(x,y) = ax^2 + by^2 + cxy + dx + ey + f$. Ayant défini une machine évaluant directement une expression du second degré, l'équipe de Chapel Hill a conçu une méthode permettant d'approcher la surface d'un cylindre par une méthode similaire à l'approximation des sphères [Gold86] [Eyle88].

1.4.2 Les approximations sphère et cylindre

La base des deux méthodes est d'utiliser des surfaces dont la profondeur s'exprime à l'aide d'une expression du second degré pour approcher la profondeur des sphères et des cylindres. Un demi-cercle pouvant être approché par une parabole (Figure 1.17), les surfaces d'approximation sont construites à partir de paraboles.

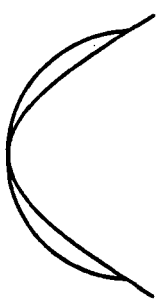


Figure 1.17 : Approximation d'un demi-cercle par une parabole

La sphère

La projection orthographique du contour d'une sphère de centre $O(x_o, y_o, z_o)$ et de rayon R sur l'écran est un cercle C de centre $O_p(x_o, y_o)$ et de rayon R.

L'évaluation de l'expression $f_2(x,y) = x^2 + y^2 - 2x_o x - 2y_o y + x_o^2 + y_o^2 - R^2$ de ce cercle, en un point quelconque permet de déterminer la position de ce point par rapport à C. Pour un pixel $P_i(x_i, y_i)$, on peut alors déterminer si la sphère est présente ou non par un test sur le signe de $f_2(x_i, y_i)$. L'algorithme de conversion utilisé est donc similaire à celui de la conversion des facettes par expressions, l'expression évaluée pour la sphère étant du second degré.

Afin d'effectuer l'élimination des parties cachées, il est nécessaire de calculer la profondeur avant de la

sphère. Or cette profondeur est définie en un pixel $P_i(x_i, y_i)$ par :

$$z = z_o - \sqrt{R^2 - (x_i - x_o)^2 - (y_i - y_o)^2} \qquad \text{(Exp.1)}$$

Pour éviter le calcul d'une racine carrée, on utilise une profondeur approchée définie par :

$$z = z_o - \frac{\left(R^2 - (x_i - x_o)^2 - (y_i - y_o)^2\right)}{R} \qquad \text{(Exp.2)}$$

La profondeur approchée s'exprimant à l'aide d'une expression du second degré, définissant un paraboloïde (objet construit par la rotation d'une parabole autour de son axe (Figure 1.18)), elle peut être calculée en tout pixel pour effectuer l'élimination des parties cachées par la méthode du "Z-buffer".

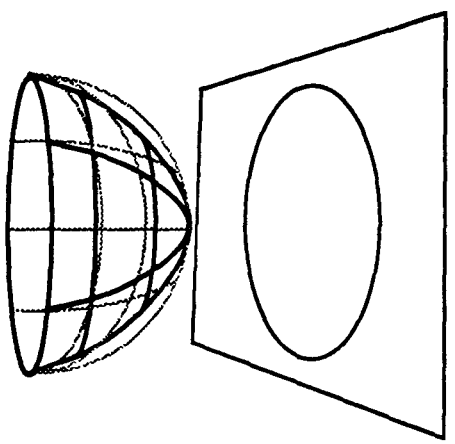


Figure 1.18 : Approximation d'une sphère par un paraboloïde

Les calculs d'éclairement nécessitent de connaître la normale à la surface en chaque pixels, l'interpolation de couleur ne pouvant s'effectuer que sur des facettes. La normale unitaire à une sphère en un point $P(x_p, y_p, z_p)$ de sa surface est

$$\vec{N} = \left(\frac{x_p - x_o}{R}, \frac{y_p - y_o}{R}, \frac{z_p - z_o}{R}\right) \qquad \text{(Exp.3)}$$

L'utilisation de l'approximation définie pour la profondeur permet d'obtenir une normale approchée :

$$\vec{N} = \left(\frac{x_p - x_o}{R}, \frac{y_p - y_o}{R}, \frac{\left(R^2 - (x_i - x_o)^2 - (y_i - y_o)^2\right)}{R^2}\right) \qquad \text{(Exp.4)}$$

dont les composantes s'expriment à l'aide de deux fonctions du premier degré et d'une fonction du second degré.

Si l'on désire ne calculer que l'intensité diffuse pour une source directionnelle ($\hat{L}(l_x, l_y, l_z)$ est constant en tout point), on peut alors évaluer directement le produit scalaire $\hat{L} \bullet \vec{N}$ en chaque pixel par :

$$\hat{L} \bullet \vec{N} = \frac{l_x(x_p - x_o) + l_y(y_p - y_o)}{R} + \frac{l_z\left(R^2 - (x_i - x_o)^2 - (y_i - y_o)^2\right)}{R^2} \qquad \text{(Exp.5)}$$

l'expression obtenue étant du second degré.

Le Cylindre

Le cylindre étant infini dans deux directions, la primitive utilisée par l'équipe de Chapel Hill est définie

par un volume cylindrique coupé par deux plans parallèles qui ferment le volume.

Le contour d'un cylindre ainsi défini est construit à l'aide de deux ellipses (e_1 et e_2) et quatre droites (d_1, d_2, d_3 et d_4) (Figure 1.19).

Les deux ellipses sont définies par deux équations du second degré $c_1(x, y) = 0$ et $c_2(x, y) = 0$; un point $P(x_p, y_p)$ appartient au disque elliptique défini par e_i, si $c_i(x_p, y_p) \geq 0$.

Comme dans le cas de la conversion des facettes par expressions, les quatre équations $f_i(x, y) = 0$ $i \in \{1, 2, 3, 4\}$ des droites d_i sont utilisées afin de déterminer des demi-plans. Pour les calculs, l'équipe de Chapel Hill préfère utiliser deux expressions du second degré, qui sont définies par les produits $f_1(x, y) \times f_2(x, y)$ et $f_3(x, y) \times f_4(x, y)$ et qui remplacent les quatre expressions de demi-plans utilisés.

Il suffit d'évaluer les quatre expressions du second degré en un pixel pour connaître la position de celui-ci par rapport au contour du cylindre et déterminer si le cylindre est présent au pixel.

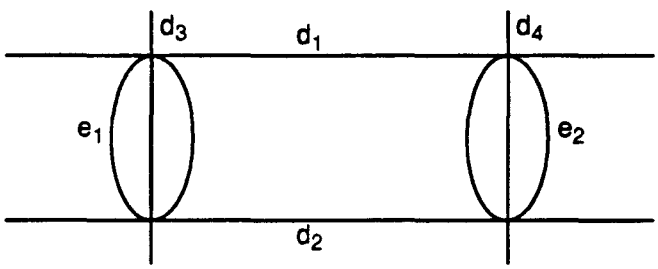


Figure 1.19 : Le contour d'un cylindre

La profondeur avant d'un cylindre est calculée par une expression $Z(x, y) = f_1(x, y) - \sqrt{f_2(x, y)}$ avec $f_1(x, y)$ une expression linéaire et $f_2(x, y)$ une expression quadratique. Une approximation de cette profondeur est donnée en tout point par $Z'(x, y) = f_1(x, y) - s - tf_2(x, y)$ avec s et t constants [Gold86]. Si s et t sont choisis convenablement, l'expression du second degré obtenue décrit un cylindre parabolique, qui approche la face avant du cylindre sur deux bande longitudinales.

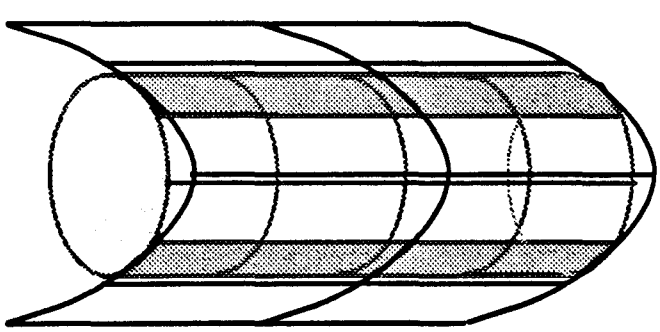


Figure 1.20 : Approximation d'un cylindre par un cylindre parabolique

L'erreur sur les profondeurs dûe à l'approximation dépend de la taille du cylindre ; l'équipe de Chapel Hill emploie donc plusieurs cylindres paraboliques pour approcher la profondeur avant d'un cylindre, chaque cylindre parabolique approchant la surface cylindrique sur des bandes longitudinales différentes (Figure 1.21). La valeur de profondeur choisi en un pixel est alors la plus grande des profondeurs des différents cylindres paraboliques.

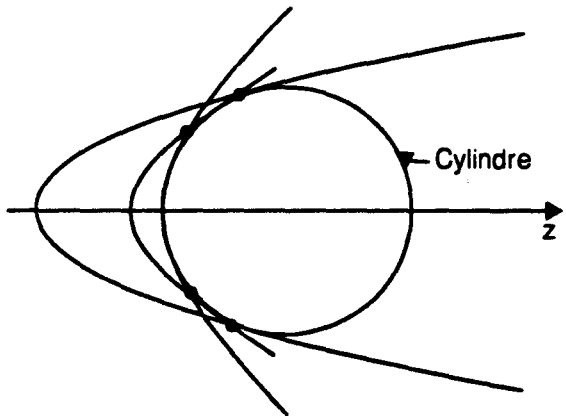


Figure 1.21 : Approximation d'un cylindre par plusieurs cylindres paraboliques

Les coordonnées de la normale à la surface du cylindre sont données par trois expressions de la forme $N_i = f_{1i}(x,y) - \sqrt{f_{2i}(x,y)}$. Pour les calculs d'éclairement, on emploie le même type d'approximation pour obtenir $N'_i = f'_{2i}(x,y)$ définissant une normale approchée définie par des expressions du second degré. Une seule forme d'approximation de normale est utilisée afin de ne pas obtenir de discontinuité lors de l'éclairement de l'objet.

1.4.3 Pixel-Planes 4 et Pixel-Planes 5

Les machines développées par l'équipe Chapel Hill utilisent un parallélisme massif pixel et ont été réalisées à l'Université de Caroline du Nord.

Pixel-Planes4

La base de la machine [Eyle88] est un tableau de processeurs pixels de taille 512×512, alimentés par un arbre permettant de calculer incrémentalement une expression du premier degré (Figure 1.22).

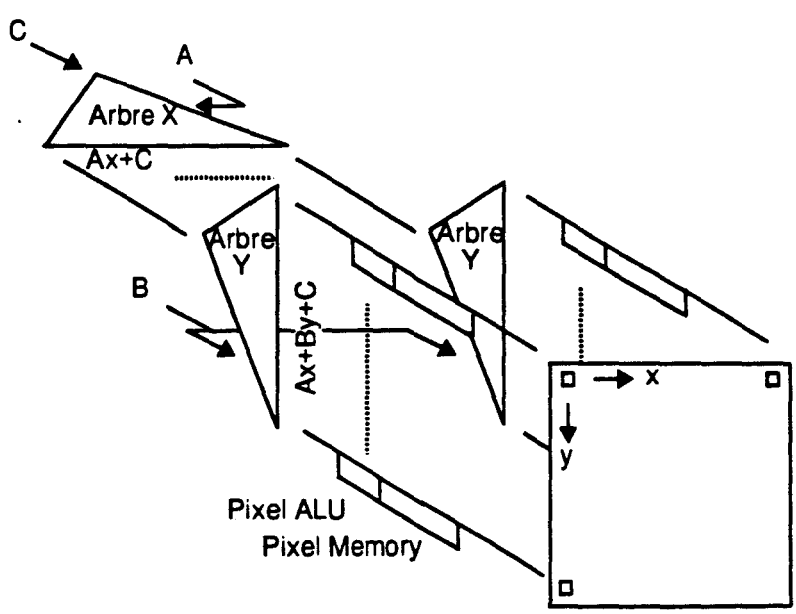


Figure 1.22 : Principe de fonctionnement de Pixel-Planes 4 (d'après [Chai91])

Cette machine SIMD a été conçue pour afficher des facettes avec la méthode par expressions.

L'arbre ne permettant de calculer que des expressions du premier degré, l'évaluation d'une expression du second degré est effectuée en séparant les termes du premier et du second degré. Les termes du premier degré sont calculés dans l'arbre et les termes du second degré sont pré-calculés en chaque pixel. Les processeurs pixels effectuent la somme des valeurs obtenues pour définir la valeur de l'expression du second degré.

L'algorithme utilisé se décompose en trois étapes :

- L'expression définissant le contour de la sphère est calculée, afin que chaque processeur puisse déterminer si l'objet est présent au pixel qui lui est associé

- La valeur de profondeur est ensuite calculée sur tous les processeurs, mais elle n'est utilisée qu'aux pixels où l'objet est présent.

- Dans le cadre d'un simple Z-Buffer, chaque processeur pixel compare la valeur de profondeur obtenue avec celle qu'il a mémorisé, pour déterminer si la sphère est effectivement visible.

- Enfin, le calcul de la couleur est effectué sur tous les processeurs, mais les valeurs R,V, B ne sont prises en compte en un pixel que si la sphère est présente et visible.

Pixel-Planes 5

L'équipe de Chapel Hill a développé ensuite la machine Pixel-Planes 5 permettant de calculer directement des expressions du second degré, en modifiant l'arbre d'alimentation des processeurs pixels [Fuch89]. Cette machine permet alors de convertir directement des approximations de cylindre et de sphères, sans pré-calculs des termes du second degré. La taille du tableau de processeurs est cependant réduite à 128×128 pixels, pour diminuer le coût de réalisation et augmenter l'efficacité de la machine. Toutes les opérations effectuées au sein des processeurs pixels sont programmées, afin d'obtenir plus de souplesse quant au traitement désiré.

L'algorithme utilisé pour afficher un cylindre est :

- Les quatre expressions du second degré définissant le contour du cylindre sont calculées en chaque pixel, afin de déterminer si l'objet est présent.

- Les valeurs des profondeurs étant calculées les unes à la suite des autres, les processeurs utilisent une mémoire annexe pour déterminer la bonne valeur de l'approximation en effectuant un Z-buffer inverse (la plus grande valeur est mémorisée)

- Lorsque toutes les valeurs de profondeurs ont été traitées, on compare la valeur de profondeur obtenue dans la mémoire annexe avec celle mémorisée dans le Z-buffer de la scène pour déterminer si le cylindre est effectivement visible.

- Le calcul de la couleur est effectué sur tous les processeurs, mais les valeurs R,V, B ne sont prises en compte en un pixel que si l'objet est présent et visible.

1.4.4 Les limites des approximations

• Le principe des méthodes d'approximation décrites ci-dessus n'est applicable que pour les objets sphériques ou cylindriques. Par exemple, il n'est pas possible d'approcher convenablement la surface d'un cône par un petit nombre de cylindres paraboliques. Les machines employant ces primitives doivent donc se limiter à la conversion de sphères et de cylindres.

• L'application d'une transformation perspective sur une sphère ou un cylindre produit un objet qui

n'est ni sphérique, ni cylindrique. Si on utilise en rendu l'approximation des sphères et des cylindres, il n'est alors plus possible d'employer la perspective. Les scènes affichées sont donc nettement moins réalistes que lors de l'utilisation d'un rendu en facettes avec application de la perspective.

• D'autres problèmes liés à l'utilisation d'approximation de sphères et de cylindres se posent également et remettent en cause l'utilisation de ces primitives lors de l'affichage de scènes.

Si deux sphères approchées de même rayon ont une intersection alors, quelles que soient leurs positions relatives, l'intersection apparaît à l'écran comme étant un segment de droite. En effet, deux sphères de même rayon sont approchées par deux paraboloïdes identiques à une translation près. Ces deux paraboloïdes sont définis par deux équations du second degré :

$$Q_i(x, y, z) = a_i x^2 + b_i y^2 + d_i x + e_i y + f_i - z = 0 \text{ avec } i \in \{1, 2\} \qquad \text{(Eq.1)}$$

Comme les deux paraboloïdes sont identiques à une translation prèLQs $a_1 = a_2$ et $b_1 = b_2$. L'intersection de ces deux surfaces peut alors être définie par :

$$(d_1 - d_2)\, x + (e_1 - e_2)\, y + (f_1 - f_2) = 0 \qquad \text{(Eq.2)}$$

et l'une des deux équations de paraboloïdes.

L'équation (Eq.2), qui représente la projection de l'intersection sur l'écran, démontre que les pixels appartenant simultanément aux deux paraboloïdes sont alignés.

L'approximation de la profondeur d'un cylindre par plusieurs cylindres paraboliques implique une discontinuité de la courbe d'intersection entre le cylindre approché et un autre objet. Cette discontinuité de la courbe d'intersection oblige à employer beaucoup de cylindres paraboliques pour l'approximation de la profondeur, afin de réduire le défaut visuel.

Les approximations des objets sphériques et cylindriques n'améliorent donc pas beaucoup les intersections entre les objets d'une scène.

Si l'on veut utiliser une autre primitive d'affichage que la facette, il faut donc se tourner vers d'autres surfaces que celles uniquement à base de paraboles. Le plan étant décrit par une équation du premier degré, il semble alors naturel d'envisager d'utiliser une primitive dont la surface est définie à l'aide d'une équation du second degré : la surface quadrique.

Chapitre 2 :
La Quadrique

Dans le chapitre précédent, nous avons remarqué qu'une voie a été ouverte vers l'utilisation de primitives de rendu de plus haut niveau. La primitive que nous proposons d'étudier ici est la quadrique, représentée à l'aide d'une équation du second degré en x, y et z.

Nous introduisons tout d'abord les notions liées aux surfaces définies par ce type d'équation et à leur utilisation par une machine de rendu géométrique. Nous présentons dans une seconde partie les différentes études menées par d'autres équipes pour la conversion des quadriques en pixels. Nous décrivons ensuite les éléments ayant conduit à la définition de processeurs de conversion de quadriques. Les différentes versions de processeurs étudiées en vue d'une implémentation matérielle dans une machine objet utilisant le pipeline de rendu de Phong sont alors détaillées. Nous présentons les données fournies aux processeurs de conversion, ceux-ci nécessitant l'emploi de valeurs entières et nous terminons par l'utilisation des processeurs définis lors d'un découpage de l'écran en zones.

2.1 Mathématiques pour la Quadrique

La quadrique regroupe différentes classes de surfaces. Ces surfaces sont les extensions à l'espace des coniques, qui sont des courbes définies par des équations du second degré à deux variables. Pour mettre en évidence les différences entre les classes de surfaces quadriques, intéressons nous d'abord aux courbes coniques.

Les coniques non dégénérées sont de trois types (Figure 2.1): elliptique, hyperbolique et parabolique. Les ellipses, dont le cercle, regroupent l'ensemble des courbes fermées, les hyperboles sont des courbes non fermées ayant deux parties disjointes, et les paraboles des courbes non fermées n'ayant qu'une partie.

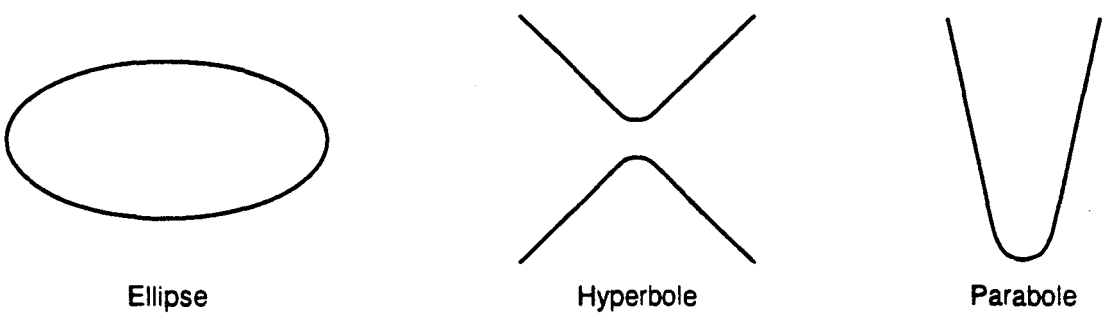


Figure 2.1: Les Coniques

Les coniques dégénérées du premier ordre sont des courbes définies par le produit de deux droites, qui peuvent être sécantes, parallèles ou confondues.

Les quadriques sont des surfaces construites par l'extension des coniques à une troisième dimension :

• Les surfaces de révolution sont décrites par une courbe conique pivotant autour d'un des axes de son repère canonique. Les quadriques obtenues sont les ellipsoïdes, dont la sphère est un cas particulier, les hyperboloïdes et les paraboloïdes. Une autre classe de quadriques regroupe les paraboloïdes hyperboliques, dont la base est une hyperbole ayant des paramètres qui décrivent une trajectoire parabolique. L'ensemble de ces surfaces forme les quadriques non dégénérées (Figure 2.2).

• Les surfaces étant construites par composition d'une courbe conique et d'une droite sont des quadriques dites "dégénérées du premier ordre" (Figure 2.3). Ces surfaces sont les cônes qui sont construits par la rotation d'une droite autour d'un axe, et les surfaces cylindriques, qui sont définies à partir d'une courbe translatée suivant un axe et qui ont pour noms : cylindre elliptique (que l'on appelle communément cylindre), cylindre hyperbolique et cylindre parabolique.

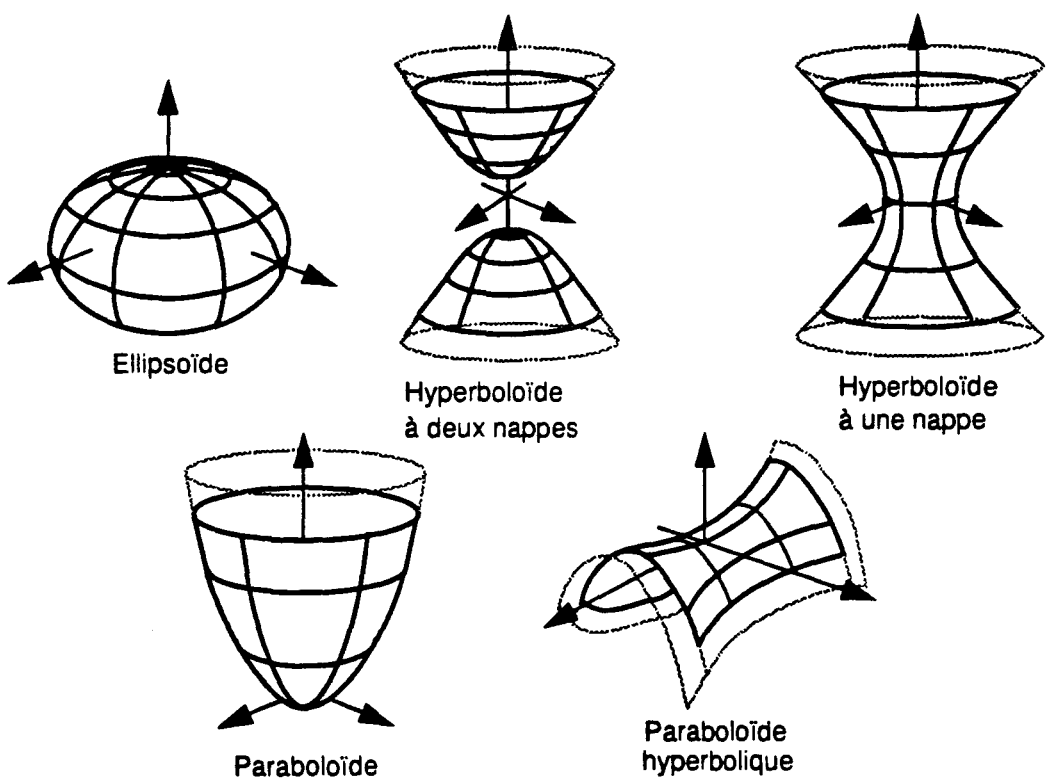


Figure 2.2 : Les Quadriques non dégénérées

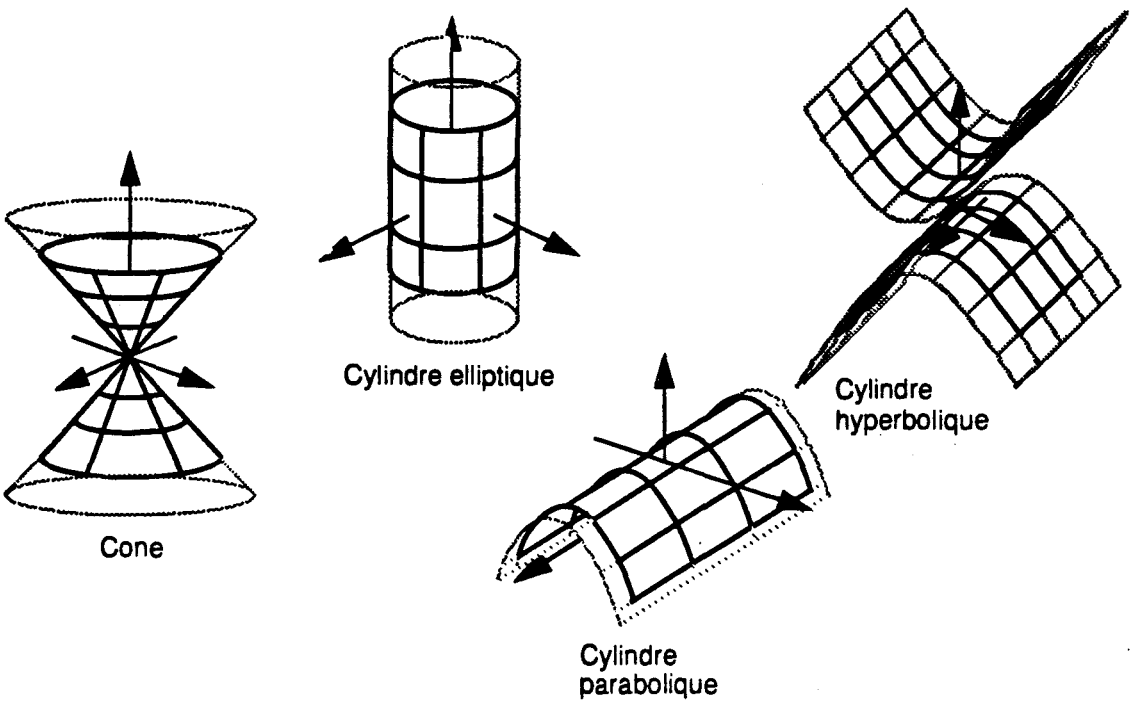


Figure 2.3 : Les Quadriques dégénérées du 1er ordre

• Les quadriques dites "dégénérées du second ordre" sont des surfaces construites par la composition d'une courbe conique dégénérée et d'une droite. Les surfaces obtenues sont par exemple, deux plans qui peuvent être en intersection, parallèles ou même confondus (Figure 2.4).

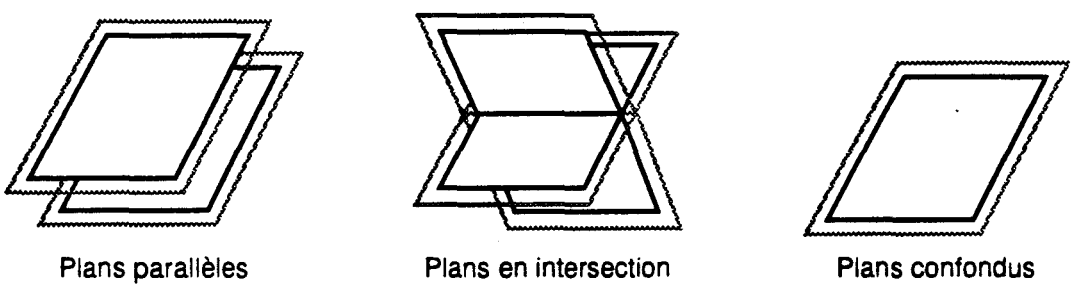


Figure 2.4 : Exemples de Quadriques dégénérées du 2nd ordre

2.1.1 Définition Mathématique

Les paramètres des expressions et des équations des surfaces quadriques définissent directement la forme de celles-ci, comme dans le cas des courbes coniques.

Les coniques et les quadriques

Dans le repère canonique $(O, \vec{x}, \vec{y})$, les trois types de coniques non dégénérées sont :

Les ellipses, qui sont des courbes fermées définies par les expressions paramétriques $\begin{cases} x = a \times \cos\theta \\ y = b \times \sin\theta \end{cases}$ et

par les équations implicites de la forme $\frac{x^2}{a^2}+\frac{y^2}{b^2}-1=0$, les paramètres a et b étant définis par l'intersection de la courbe avec les axes du repère.

(Si a=b=R, on retrouve alors la définition d'un cercle de rayon R).

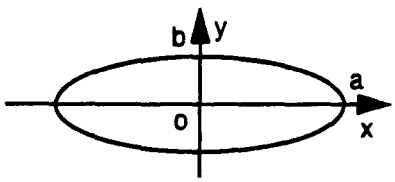


Figure 2.5 : l'Ellipse

- Les hyperboles dont la définition passe par le choix de 2 droites symétriques par rapport aux axes du repère canonique, que l'on appelle asymptotes de l'hyperbole. L'équation de ces droites dans le repère fixé est $a\times x+b\times y=0$ pour l'une et $a\times x-b\times y=0$ pour l'autre.

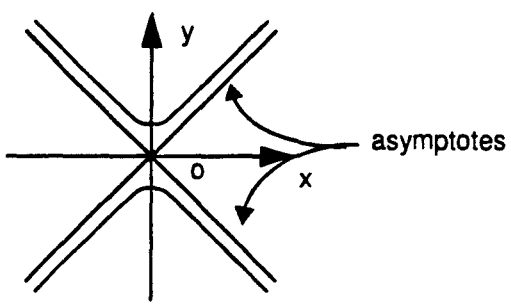


Figure 2.6 : l'Hyperbole

Les expressions paramétriques obtenues sont $\begin{cases} x=a\times\cosh\theta \\ y=b\times\sinh\theta \end{cases}$ et

les équations implicites sont de la forme $\frac{x^2}{a^2}-\frac{y^2}{b^2}+1=0$.

- Les paraboles sont des courbes plus particulières puisque, contrairement aux courbes précédentes, elles ne possèdent qu'un axe de symétrie : l'axe y.

La forme de leur expression paramétrique $\begin{cases} x=t \\ y=\frac{t^2}{a} \end{cases}$ et de leur équation implicite $x^2+a\times y=0$

indiquent par ailleurs cette différence.

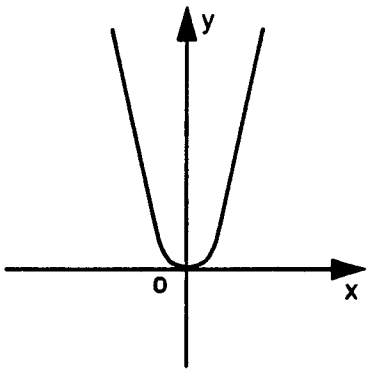


Figure 2.7 : la Parabole

Les expressions paramétriques des différentes formes de surfaces quadriques, comme leurs équations implicites peuvent facilement se déduire de celles des coniques. Ainsi, il est plus aisé de déterminer le type d'une quadrique non dégénérée en connaissant l'une ou l'autre de ses deux formes d'expression dans le repère canonique (Tableau 2.1).

Type de Quadrique	Expression paramétrique	Equation implicite
Ellipsoïde	$x = a \times \cos\theta \times \cos\varphi$ $y = b \times \cos\theta \times \sin\varphi$ $z = c \times \sin\theta$	$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} - 1 = 0$
Hyperboloïde à 1 nappe	$x = a \times \cos\theta \times \cosh\varphi$ $y = b \times \sinh\varphi$ $z = c \times \sin\theta \times \cosh\varphi$	$\frac{x^2}{a^2} - \frac{y^2}{b^2} + \frac{z^2}{c^2} - 1 = 0$
Hyperboloïde à 2 nappes	$x = a \times \cos\theta \times \sinh\varphi$ $y = b \times \cosh\varphi$ $z = c \times \sin\theta \times \sinh\varphi$	$\frac{x^2}{a^2} - \frac{y^2}{b^2} + \frac{z^2}{c^2} + 1 = 0$
Paraboloïde	$x = a \times t \times \cos\theta$ $y = t^2$ $z = c \times t \times \sin\theta$	$\frac{x^2}{a^2} + \frac{z^2}{c^2} - y = 0$
Paraboloïde Hyperbolique	$x = a \times t \times \cosh\theta$ $y = t^2$ $z = c \times t \times \sinh\theta$	$\frac{x^2}{a^2} - \frac{z^2}{c^2} - y = 0$

Tableau 2.1 : Expressions des Quadriques non dégénérées

Hormis le cône, les quadriques dégénérées du 1er ordre sont des cylindres et donc leurs équations dans le repère canonique ne comporte aucun terme en z (ou x ou y). Ceci revient à dire que l'équation implicite d'un cylindre, dans le repère canonique, à la même forme que l'équation d'une conique (si $C(x,y) = ax^2 + by^2 + cxy + dx + ey + f = 0$ est l'équation d'une ellipse alors l'équation d'un cylindre elliptique est $Q(x,y,z) = ax^2 + by^2 + cxy + dx + ey + f = 0$). On peut alors remarquer que les expressions paramétriques des cylindres sont identiques à celles des coniques (Tableau 2.2). Les cônes sont eux aussi facilement reconnaissables, puisqu'ils représentent des cercles dont les rayons varient en fonction de la profondeur, comme le montre la forme de leurs expressions.

Type de Quadrique	Expression paramétrique	Equation implicite
Cylindre Elliptique	$x = a \times \cos\theta$ $y = b \times \sin\theta$	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$
Cylindre Hyperbolique	$x = a \times \cosh\varphi$ $y = b \times \sinh\varphi$	$\frac{x^2}{a^2} - \frac{y^2}{b^2} + 1 = 0$
Cylindre Parabolique	$x = a \times t$ $y = t^2/b$	$\frac{x^2}{a^2} - b \times y = 0$
Cône	$x = a \times t \times \cos\theta$ $y = b \times t \times \sin\theta$ $z = c \times t$	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 0$

Tableau 2.2 : Expressions des Quadriques dégénérées du 1er ordre

Nous ne détaillons pas les expressions de quadrique dégénérées du 2nd ordre, celles-ci n'offrant pas d'intérêt particulier pour comprendre les différentes formes utilisables.

Une remarque importante pour la suite de ce travail : l'intersection d'un plan défini par $P(x,y,z) = 0$ et d'une quadrique $Q(x,y,z) = 0$ est une courbe conique, puisque si l'on place la quadrique dans un repère associé au plan (tel que le plan soit défini par $P'(x',y',z') = z' = 0$), alors l'équation de l'intersection est $I(x',y',z') = ax'^2 + by'^2 + cx'y' + dx' + ey' + f = 0$

Equation de quadrique

Dans le cadre de la visualisation, nous utilisons les quadriques sous la forme d'équations implicites. En effet, celles-ci sont suffisamment générales pour représenter toutes les familles de quadriques sous une forme unique :

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

Ceci permet donc une manipulation uniforme de toutes les quadriques, que ce soit lors des opérations

de transformations ou d'affichage.

Cette représentation a en outre l'avantage de définir directement les quadriques dans un espace tridimensionnel, alors que les expressions paramétriques les définissent dans l'espace bidimensionnel des paramètres; le passage de l'espace bidimensionnel des paramètres à l'espace tridimensionnel de visualisation pose des problèmes lors de l'affichage [Levi76][Mill88].

Les problèmes de l'affichage d'une quadrique à partir de son expression paramétrique dépendent d'abord de la représentation utilisées :

- Si la quadrique est représentée à l'aide de fonctions trigonométriques, il faut évaluer celles-ci pour déterminer les coordonnées d'un point : l'évaluation rapide d'une fonction trigonométrique est difficilement réalisable.

- Si l'on effectue un changement de l'espace des paramètres afin d'obtenir des fonctions de paramétrisation moins complexes, le nouvel espace des paramètres pose alors des problèmes. Par exemple si on fixe $u = \tan\theta$ et $v = \tan\varphi$, les ellipsoïdes peuvent être définis par $x = f_x(u, v)$, $y = f_y(u, v)$ et $z = f_z(u, v)$, mais tous les points $P(x, y, z)$ des surfaces ne peuvent plus être atteints puisque les paramètres u et v à utiliser varient de $-\infty$ à $+\infty$.

De plus avec la représentation paramétrique l'échantillonnage de l'espace des paramètres doit être contrôlé, afin que les points calculés se projettent sur les pixels de l'écran lors de l'affichage.

La représentation des quadriques sous la forme d'équations implicites s'impose donc pour le rendu.

• Les opérations de transformation sur les quadriques

Les premières transformations utilisées pour la visualisation des quadriques sont les rotations et les translations, qui permettent de placer les objets dans le repère observateur. La composition de toutes ces actions s'exprime par une matrice M 4×4, qui transforme un point $P(x, y, z)$ d'un repère lié à l'objet, en un point $P'(x', y', z')$ du repère observateur, déterminée par $P' = MP$.

Or un point $P(x, y, z)$ d'une surface quadrique $Q(x, y, z) = 0$, peut être définie sous forme matricielle par $P^t QP = 0$ avec

$$Q = \begin{bmatrix} a & d & e & g \\ d & b & f & h \\ e & f & c & i \\ g & h & i & j \end{bmatrix} \quad (Q \text{ est la représentation matricielle de } Q(x, y, z))$$

Pour déterminer l'équation $Q'(x', y', z') = 0$ représentant la quadrique dans le repère observateur, il suffit alors de connaître M puisque $P = M^{-1t}P'$ et donc $Q' = M^{-1}QM^{-1t}$ [Potm82].

Puisque $Q'(x', y', z') = 0$ et $KQ'(x', y', z') = 0$ (K constant) représente la même surface quadrique, on peut utiliser M^* la matrice des cofacteurs de M lors des calculs de changement de repère [Klei93], afin d'éviter le calcul du déterminant de M.

Comme $M^{-1} = \frac{M^*}{Det}$ (avec $Det \neq 0$), nous obtenons $M^* QM^{*1} = Det^2(M^{-1}QM^{-1t}) = Det^2 Q'$. Cette méthode de changement de repère nous évite le calcul du déterminant et la division.

Lors de la visualisation, il faut représenter une quadrique en perspective. Dans le chapitre 1, nous avons indiqué que la transformation perspective d'un objet permet d'utiliser ensuite une projection orthographique lors de l'affichage. On considère alors que les points $P(x, y, z)$ de l'espace de départ, sont vus comme étant les points $P_p(x_p, y_p, z_p)$ avec:

$$x_p = \frac{D \times x}{D+z},\ y_p = \frac{D \times y}{D+z} \text{ et } z_p = \frac{D \times z}{D+z}.$$

Rechercher l'objet résultat de la transformation perspective d'une quadrique définie par $Q(x, y, z) = 0$, consiste à déterminer les points P_p dont les antécédents vérifient cette équation. Or la transformation considérée est bijective dans son domaine de définition, donc à chaque point $P_p(x_p, y_p, z_p)$ correspond un point unique $P(x, y, z)$ de l'espace initial, qui est défini par :

$$x = \frac{D \times x_p}{D - z_p},\ y = \frac{D \times y_p}{D - z_p} \text{ et } z = \frac{D \times z_p}{D - z_p}.$$

Si P vérifie

$$Q(x, y, z) = ax^2 + by^2 + cz^2 + 2dxy + 2exz + 2fyz + 2gx + 2hy + 2iz + j = 0 \qquad \text{(Eq.3)}$$

alors P' vérifie

$$Q(x_p, y_p, z_p) =$$
$$a_t x_p^2 + b_t y_p^2 + c_t z_p^2 + 2d_t x_p y_p + 2e_t x_p z_p + 2f_t y_p z_p + 2g_t x_p + 2h_t y_p + 2i_t z_p + j_t \qquad \text{(Eq.4)}$$
$$= 0$$

avec

$$a_t = a \qquad b_t = b \qquad c_t = c - 2\frac{i}{D} + \frac{j}{D^2}$$

$$d_t = d \qquad e_t = e - \frac{g}{D} \qquad f_t = f - \frac{h}{D}$$

$$g_t = g \qquad h_t = h \qquad i_t = i - \frac{j}{D} \qquad j_t = j$$

En effet si l'on cherche à exprimer l'équation $Q(x, y, z)$ pour les paramètres x_p, y_p et z_p, alors nous obtenons

$$\frac{D^2}{(D - z_p)^2} \times Q_p(x_p, y_p, z_p) = 0$$

or $D \neq 0$ et $z_p > D$ dans l'espace final, donc l'ensemble des points P_p doit vérifier (Eq.4).

L'objet résultant de la transformation est également une quadrique, mais son type peut être différent de celui de la quadrique initiale.

Puisque la transformation perspective peut être exprimée sous forme matricielle, on emploie en général la matrice M_p représentant la transformation pour calculer $M_{qp} = M_p^{-1} \times M_q \times M_p^{-1t}$ la matrice définissant la quadrique en perspective.

L'affichage

Lors de la phase de rendu, nous devons déterminer pour chaque pixel (x, y) de l'écran, si l'objet est présent et quelle est sa couleur.

En outre, si l'on veut effectuer l'opération d'élimination des parties cachées, nous devons connaître la

position de cet objet par rapport aux autres objets de la scène. Pour cette dernière opération et pour utiliser l'algorithme du Z-buffer, il faut déterminer une valeur de profondeur pour chaque pixel où la quadrique est présente.

L'équation (Eq.3) peut se récrire en

$$Rz^2 + Sz + T = 0 \qquad \text{(Eq.5)}$$

avec

$$R = c \qquad S = 2ex + 2fy + 2i \qquad T = ax^2 + by^2 + 2dxy + 2gx + 2hy + j.$$

Si l'on pose x et y comme étant les coordonnées d'un pixel, rechercher la profondeur équivaut à résoudre cette équation du second degré en z.

C'est lors du calcul de la profondeur d'une quadrique que l'on détermine automatiquement si la surface est présente en un pixel, puisque dans le cas où la quadrique est absente, l'équation (Eq.5) n'a pas de solution réelle. La résolution de l'équation (Eq.5) pouvant donner deux solutions réelles, nous devons connaître laquelle des deux valeurs obtenues en un pixel doit être utilisée pour l'élimination des parties cachées (nous répondrons plus loin à cette question).

Le calcul de la couleur d'un pixel, par le modèle d'éclairement de Phong [Phon75], est effectué par la donnée de 4 valeurs en un pixel (x, y), qui sont la profondeur et les trois composantes de la normale à la surface. Connaissant l'équation $Q(x, y, z) = 0$ représentant la surface, la normale $\vec{N}$ non normalisée est définie par la dérivée de $Q(x, y, z)$ suivant les trois variables :

$$\vec{N} = \left(\frac{\partial Q}{\partial x}, \frac{\partial Q}{\partial y}, \frac{\partial Q}{\partial z}\right) .$$

Algorithme de conversion

L'algorithme comporte deux parties distinctes. La première, qui regroupe la détermination du contour et de la profondeur, consiste à résoudre l'équation $Q(x, y, z) = 0$, connaissant (x, y) les coordonnées d'un pixel. La seconde partie recouvre les calculs des normales pour l'éclairement ultérieur des pixels par le modèle de Phong.

- Le contour et la profondeur d'une quadrique s'obtiennent en résolvant l'équation (Eq.5).

  On obtient alors deux valeurs de profondeur en un pixel (x, y)

$$z1 = \frac{-S + \sqrt{S^2 - 4 \times R \times T}}{2R} \text{ et } z2 = \frac{-S - \sqrt{S^2 - 4 \times R \times T}}{2R} \text{ (avec } R \neq 0) \qquad \text{(Exp.6)}$$

  Comme R est une valeur constante, S est une fonction du premier degré et T une fonction du second degré, nous pouvons résoudre l'équation par:

$$z'1 = f1(x, y) + \sqrt{f2(x, y)} \text{ et } z'2 = f1(x, y) - \sqrt{f2(x, y)} \qquad \text{(Exp.7)}$$

avec $f1(x, y) = -\frac{2e}{c}x - \frac{2f}{c}y - \frac{2i}{c}$

et $f2(x, y) = \frac{4(e^2 - ac)}{c^2}x^2 + \frac{4(f^2 - bc)}{c^2}y^2 + \frac{8(ef - dc)}{c^2}xy + \frac{8(ei - gc)}{c^2}x + \frac{8(fi - hc)}{c^2}y + \frac{4(i^2 - jc)}{c^2}$.

  Nous obtenons une indication de présence par $f2(x, y) \geq 0$ (qui définit la silhouette de la quadrique), ainsi que deux valeurs de profondeur ordonnées ($z'1 \leq z'2$, en fixant R positif). L'équation $f1(x, y) - z = 0$ détermine un plan que l'on appelle plan polaire de la quadrique.

- La normale $\vec{N}(Nx, Ny, Nz)$ en un point de la surface est définie à partir de (Eq.3) par

$$\begin{aligned} Nx &= 2 \times a \times x + 2d \times y + 2e \times z + 2g \\ Ny &= 2 \times b \times y + 2d \times x + 2f \times z + 2h \\ Nz &= 2 \times c \times z + 2e \times x + 2f \times y + 2i \end{aligned} \qquad \text{(Exp.8)}$$

Or, comme nous l'avons vu précédemment, l'objet que nous visualisons n'est pas l'objet réel, puisqu'il a subi une transformation perspective. Il nous faut retrouver la normale réelle à la surface en chaque point P_p transformé, c'est à dire la normale au point P antécédent de P_p.

Exprimons les composantes de la normale à la surface non transformée par rapport au point P', et aux nouveaux coefficients:

$$Nx = (2 \times a_t \times x_p + 2d_t \times y_p + 2e_t \times z_p + 2g_t) \times \frac{D}{D - z_p}$$

$$Ny = (2 \times b_t \times y_p + 2d_t \times x_p + 2f_t \times z_p + 2h_t) \times \frac{D}{D - z_p}$$

$$Nz = \left(\left(2 \times c_t + \frac{2i_t}{D}\right) \times z_p + \left(2e_t + \frac{2g_t}{D}\right) \times x_p + \left(2f_t + \frac{2h_t}{D}\right) \times y_p + 2i_t + \frac{2 \times j_t}{D}\right) \times \frac{D}{D - z_p}$$

Une normale à une surface doit être normalisée pour les calculs d'éclairement.

Nous pouvons calculer

$$\vec{N'} = \left(D - \frac{z_p}{D}\right)\vec{N} \qquad (z_p \neq D)$$

à la place de N, et l'utiliser après sa normalisation pour l'éclairement (l'opération de normalisation devant être effectuée aussi lors de l'utilisation directe de $\vec{N}$).

Chaque coefficient de la normale est défini par une expression du 1er degré en x, y et z, or la profondeur d'une quadrique s'exprime, en chaque pixel, à l'aide d'une expression du premier degré, d'une expression du second degré et de l'évaluation d'une racine carrée. Nous pouvons alors transformer l'expression de chaque composante de la normale pour les exprimer sous la forme :

$$N_x = ((2 \times a_t \times x_p + 2d_t \times y_p + 2g_t) + 2e_t \times f1(x_p, y_p)) \pm \sqrt{4e_t^2 \times f2(x_p, y_p)} \qquad \text{(Exp.9)}$$

$$N_y = ((2 \times b_t \times y_p + 2d_t \times x_p + 2h_t) + 2f_t \times f1(x_p, y_p)) \pm \sqrt{4f_t^2 \times f2(x_p, y_p)} \qquad \text{(Exp.10)}$$

$$N_y = \left(\left(\left(2e_t + \frac{2g_t}{D}\right) \times x_p + \left(2f_t + \frac{2h_t}{D}\right) \times y_p + 2i_t + \frac{2 \times j_t}{D}\right) + \left(2 \times c_t + \frac{2i_t}{D}\right) \times f1(x_p, y_p)\right) \qquad \text{(Exp.11)}$$

$$\pm \sqrt{\left(2 \times c_t + \frac{2i_t}{D}\right)^2 \times f2(x_p, y_p)}$$

Nous obtenons ainsi des expressions permettant de calculer directement une normale, sans utiliser la valeur de profondeur.

2.2 Etudes antérieures de conversion de quadriques

Certaines études ont déjà été menées concernant la quadrique en tant que primitive d'affichage. Pour la plupart de celles-ci, le but a été la conversion d'objet définis en C.S.G., la surface quadrique servant à délimiter un volume.

Nous allons présenté deux approches, une implantation d'algorithmes d'affichage sur processeurs généraux [Klei93] et la définition d'une machine spécialisée [Kede89].

2.2.1 La conversion logicielle

Bien que l'étude de van Kleij [Klei93] ait été menée dans le cadre d'une approche C.S.G. Les algorithmes peuvent être exploités sans modification notable pour un rendu géométrique.

Nous allons centrer notre présentation sur la partie conversion, nous ne prenons pas en compte des informations nécessaires au traitement de l'arbre C.S.G.

Algorithme de subdivision "scan-line"

La démarche adoptée est basée sur un algorithme C.S.G. de conversion de polygones utilisant le balayage de ligne.

L'algorithme initial emploie une liste de faces actives (Active Facet List) et il est construit sur le principe suivant :

- Pour une ligne (p_i, p_j) écran donnée (puis un segment (p_i, p_j) de ligne écran donnée), la liste est triée suivant le sens croissant de profondeur sur le bord gauche des faces et le tri s'arrête lorsque l'on a trouvé la face visible. On génére ainsi une liste de bord gauche. Le même tri est appliqué au bord droit pour créer la liste de bord droit.

- Si les listes de bord gauche et de bord droit obtenues sont identiques, il ne reste plus qu'à afficher la face visible.

- Si les deux listes diffèrent, alors il existe des faces qui s'intersectent sur la ligne en cours d'affichage. On détermine un point p_k d'intersection entre deux faces, puis le traitement est recommencé sur les deux segments (p_i, p_k) et (p_k, p_j).

Malheureusement, même si les listes de bord gauche et droit sont identiques, avec les quadriques, il est quand même possible qu'une intersection se produise sur un segment de ligne (p_i, p_j) (Figure 2.8.a).

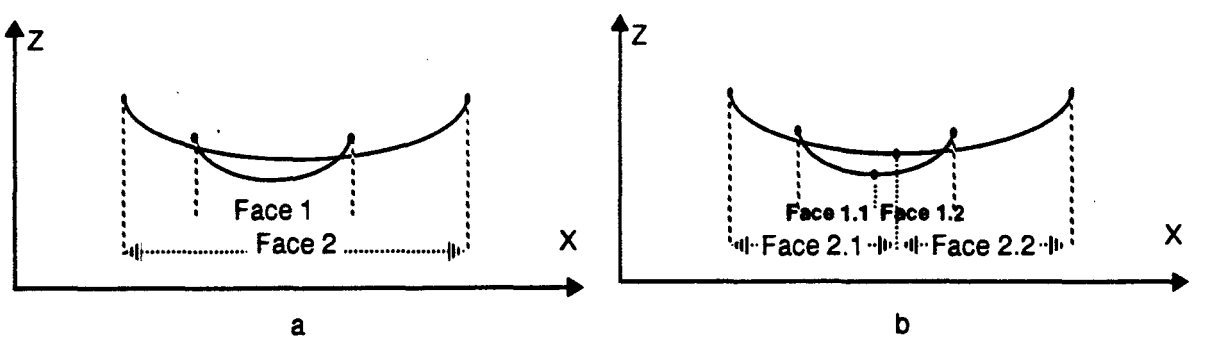


Figure 2.8 : Cas d'intersection sur un segment

Pour que cet algorithme soit utilisable, il est nécessaire de découper les quadriques en segments monotones dans le plan d'affichage des différentes lignes (Figure 2.8.b), afin de n'obtenir que trois cas simples à traiter.

Ces trois cas sont :

- le cas de non-intersection : deux segments de quadriques n'ont pas d'intersection si leurs rectangles

englobants respectifs non pas de parties communes (Figure 2.9.a) ou si le segment devant est convexe et le segment derrière est concave (Figure 2.9.b). Le segment de quadrique à afficher est le segment de devant.

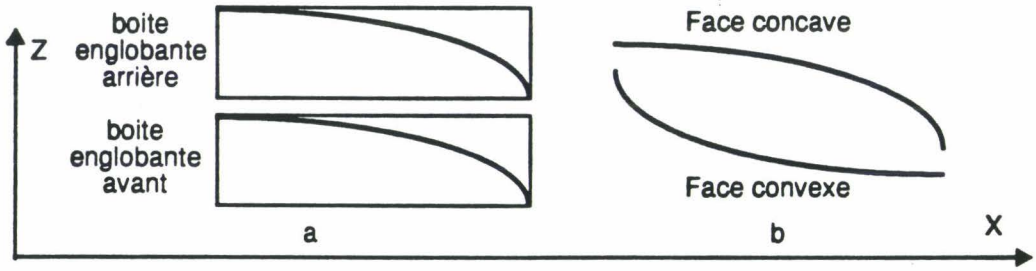


Figure 2.9 : Cas triviaux de non-intersections

- le cas trivial où l'intersection est un point unique : deux segments de quadriques monotones ayant des valeurs de profondeurs dont l'ordre est inversé à droite et à gauche, ne s'intersectent qu'en un seul point. L'intersection de ces deux segments de quadriques peut être définie par différentes méthodes analytiques, mais elles ne sont pas utilisées non pour des raisons de précision des valeurs obtenues (les équations à résoudre sont du quatrième ordre) mais parce que les solutions sont multiples (on considère les deux quadriques entières). Pour résoudre le système formé par les deux équations de quadriques et trouver le point d'intersection sur une ligne, la solution employée est la méthode itérative de Newton-Raphson. Le point initial de l'itération est le centre du rectangle défini par l'intersection des deux boites englobantes des quadriques sur la ligne. En cas de divergence du système, le segment de ligne en cours de traitement est divisé en deux.

- le cas d'intersection en deux points ou en aucun point : dans ce cas, une méthode du type "diviser pour conquérir[1]" est employée. Le segment de ligne est divisé en deux de manière récursive jusqu'a ce que l'on détecte un cas de non intersection ou d'intersection en un point unique, ou que le segment soit plus petit qu'un pixel.

Pour l'éclairage, van Kleij ne détermine la normale à la surface visible qu'en certains points du segment et calcule la couleur de l'objet en tous pixels par interpolation linéaire, réduisant ainsi le coût.

Algorithmes scanline et tampon de profondeur

Sur une ligne, la face visible en un pixel est déterminée à partir des valeurs de profondeurs calculées en ce point. Van kleij a implémenté la méthode de calcul incrémental en entier, par différences finies[2] [Chan89] des expressions du premier et du second degré pour le calcul entier des profondeurs des surfaces quadriques, ainsi qu'une méthode de calcul des profondeurs en flottant. La profondeur de l'objet en un pixel est définie ensuite par le parcours de l'arbre C.S.G., connaissant les profondeurs de ses Nœuds.

Pour une implantation sur un processeur général, l'algorithme de subdivision en scanline et celle du "scan-line" et tampon de profondeur sont relativement simples à mettre en oeuvre.

1. divide and conquer en anglais
2. forward differencing en anglais

Si l'on cherche à définir une unité de conversion spécialisée, le seul algorithme qui peut être utilisé est la méthode scanline et tampon de profondeur ; la méthode de subdivision scan-line étant adaptative et globale pour une scène nécessite de faire des choix qui augmentent le coût de l'unité de conversion, le parallélisme au niveau des objets ou des pixels ne pouvant pas être employé.

2.2.2 La Ray Casting Machine

La Ray-Casting Machine (appelée maintenant Ray Casting Engine) est à notre connaissance le seul système câblé pouvant afficher des quadriques sans approximation [Kede89][Elli91]. Cette machine est spécifiquement conçue pour le rendu d'arbres C.S.G., aussi les quadriques utilisées lors de la conversion ne sont pas délimitées par des plans (la coupe des quadriques est effectuée lors du traitement de l'arbre C.S.G.).

La machine est constituée de deux types de processeurs :

- les processeurs objets appelés P.C. (Primitive Classifier) permettant de calculer en chaque pixel, les valeurs de profondeur avant et arrière de l'objet dont il a la charge.

- les processeurs de combinaisons appelés C.C. (Classification Combine processors), qui réalisent les opérations logiques de l'arbre C.S.G. Les C.C. peuvent accepter directement des valeurs de la machine hôte, afin d'utiliser des primitives de plus haut niveau que les quadriques (par exemple le tore) dans l'arbre C.S.G.

L'intérêt que nous portons à cette machine se situe au niveau des P.C., puisque ceux-ci convertissent des quadriques en pixels.

Comme nous l'avons vu, l'expression des profondeurs d'une quadrique peut s'écrire sous la forme:

$z = f1(x, y) \pm \sqrt{f2(x, y)}$ avec $f1(x, y)$ expression linéaire et $f2(x, y)$ expression quadratique.

Un P.C. (Figure 2.10) contient alors une unité de calcul linéaire (Linear Evaluation Unit) et une unité de calcul quadratique (Quadratic Evaluation Unit) utilisant, toutes deux, la méthode du "forward differencing", et une unité de calcul de racine carrée (SQRT) ainsi qu'un additionneur et un soustracteur. Les opérateurs ne calculent qu'un bit à la fois, afin de réduire la complexité matérielle. La racine carrée d'un nombre sur 48 bits, est calculée par une méthode exacte (voir au paragraphe 2.3.3) adaptée au mode bit série, par l'utilisation d'une unité Bit Reverse Unit (le Q.E.U. fournissant d'abord les bits de poids faibles et le SQRT utilisant en premier les bits de poids forts). Un P.C. est capable d'effectuer le calcul des profondeurs d'une quadrique ou d'un plan (dans ce dernier cas, l'unité de calcul quadratique et l'unité de calcul de la racine carré, ne sont pas exploitées).

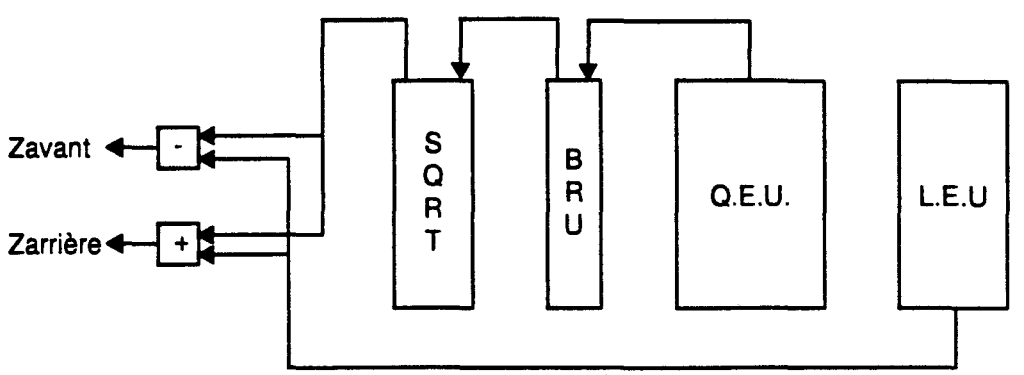


Figure 2.10 : Le P.C. de la Ray Casting Machine

Un prototype de cette machine a été réalisé et forme un outil remarquable pour le domaine de la C.A.O., en autorisant l'utilisation de quadriques dans les arbres C.S.G.

Cependant l'utilisation du mode bit série pour les calculs et l'emploi de quadriques infinies comme primitives (la limitation est effectué dans l'arbre C.S.G.), empêche son exploitation en rendu géométrique.

2.3 Eléments pour la définition d'une unité de conversion de quadriques

Avant de définir une unité de conversion de quadriques, il est nécessaire de définir précisément l'objet à convertir. Ensuite, nous présentons les modules de base utilisés pendant la phase de conversion, pour définir un processeur objet permettant la conversion de quadriques en pixels.

2.3.1 La définition de l'objet affiché

En C.S.G. les quadriques définissent les enveloppes des volumes utilisés pour la construction des objets. La quadrique étant une surface, elle peut être employée pour construire des objets suivant le modèle surfacique. Il faut donc déterminer lors de la conversion, si la quadrique définit une surface ou un volume. Si l'on veut qu'une machine de rendu géométrique puisse utiliser la quadrique en conversion, il est nécessaire de la délimiter puisque les surfaces quadriques sont en général infinies : nous avons choisi de délimiter l'objet quadrique à l'aide de plans.

L'objet quadrique

Deux définitions sont envisageables pour l'affichage d'une quadrique :

- La première est d'utiliser la primitive pour définir un objet surfacique, qui est alors la surface de la quadrique délimitée par les plans de coupe.
- La seconde est de considérer, comme en C.S.G, que la surface quadrique est utilisée pour délimiter un volume : la surface quadrique définit l'enveloppe d'un volume qui est coupé par des plans.

Le choix d'objets surfaciques ne pose pas de problème, par contre la conversion des objets quadriques volumiques nécessite de déterminer le volume que l'on cherche à afficher.

Pour les quadriques fermées telles que la sphère, le volume associé est intuitivement l'intérieur de cette surface (Figure 2.11).

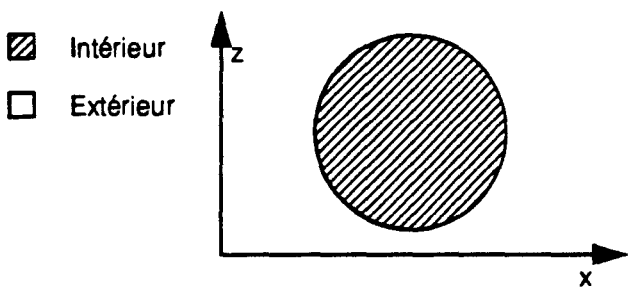


Figure 2.11 : Définition d'un volume sphérique

Pour les autres quadriques, il est difficile de déterminer a priori, le volume associé. La seule solution est de désigner explicitement le volume.

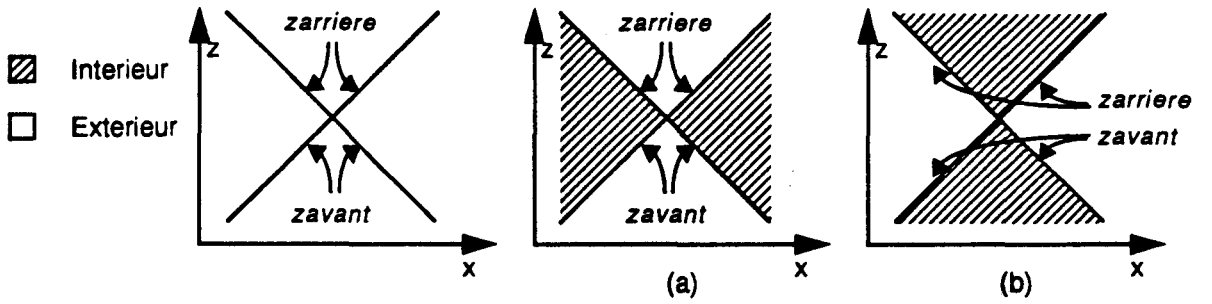


Figure 2.12 : Définition d'un volume par une surface cônique

Prenons le cas d'un cône (Figure 2.12): la vision bidimensionnelle de la surface indique que deux volumes peuvent être considérés. Dans le cas (a), le volume est l'espace compris entre les valeurs *zavant* et *zarriere*, alors que dans le cas (b) il est en deux parties définies par]-∞ , *zavant*] et [*zarriere* , +∞[. Lorsque le volume quadrique est dans la position (a), nous employons le terme quadrique dans la vue de coté et dans la position (b) le terme quadrique dans la vue de face.

Lorsque l'on a défini un volume, il faut savoir repérer les deux cas lors de l'affichage. Pour cela un moyen très simple est d'orienter l'équation de la quadrique lors de la modélisation. Si on pose que le coefficient c de l'équation doit être positif lorsque l'on se trouve dans le cas (a); on obtient alors $zavant = z1$, $zarriere = z2$ puisque $z1 \leq z2$ et le volume est défini par l'ensemble des points $P_V(x, y, z)$ tels que $zavant \leq z \leq zarriere$ (*zavant* et *zarriere* étant définis aux points (x, y)). Si lors des opérations de changement de repère le coefficient c devient négatif, nous obtenons $zavant = z2$ et $zarriere = z1$ puisque $z1 \geq z2$ alors que $zavant \leq zarriere$, donc l'ensemble des point "compris" entre $z1$ et $z2$ devient l'union de]-∞ , *zavant*] et [*zarriere*,+∞[.

Cette méthode d'orientation des quadriques permet aussi de déterminer si un point $P(x_p, y_p, z_p)$ se trouve à l'intérieur du volume choisi, par l'évaluation de $Q(x_p, y_p, z_p) \geq 0$, $Q(x, y, z) = 0$ étant l'équation de la quadrique, qui est positive.

Les Plans de Coupes

Les quadriques étant infinies en général, nous avons décidé de les délimiter par des plans. Ces plans peuvent être simplement considérés comme des limites, dans le cas où une quadrique est utilisée pour former un patch, ou comme des coupes effectuées dans le volume associé à cette quadrique.

Une quadrique étant coupée par un plan, il faut choisir la partie à afficher parmi deux possibilités (Figure 2.13).

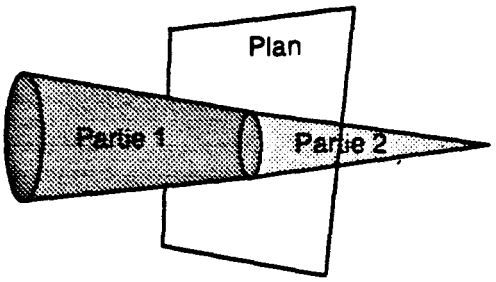


Figure 2.13 : Quadrique coupée par un plan

Une solution simple pour contrôler ce choix lors de l'affichage est d'orienter un plan par sa normale lors de la définition de l'objet, et de supprimer la partie de quadrique se situant du coté de la normale. Après la phase de changement de repère et perspective, l'équation $Pl(x, y, z) = ax + by + cz + d = 0$ d'un plan permet de déduire sa normale $N(a, b, c)$, afin d'effectuer le choix. Nous avons appelé "plans avants", les plans dont les normales sont orientées vers l'écran, "plans arrières", ceux dont les normales sont orientées à l'opposé et "plans perpendiculaires", les plans orthogonaux à l'écran.

Le changement de repère défini par la matrice M d'un plan $P(x, y, z)$ est donné par $P' = M^{-1}P$. Si on effectue le changement de repère avec M^* la matrice des cofacteurs au lieu de M^{-1}, la normale au plan peut être inversée, puisque le déterminant de la matrice peut être négatif : on risque de choisir la mauvaise partie de la quadrique lors de l'affichage. Pour se prémunir contre cette erreur, nous sommes contraints à calculer le déterminant de M.

2.3.2 Les modules de base

Comme on peut le remarquer dans l'étude théorique, le calcul en chaque pixel de la profondeur d'une quadrique et de sa normale nécessite l'évaluation d'expressions du premier degré, d'une expression du second degré et d'une racine carrée. L'affichage s'effectuant en "scan-line", les modules évaluant les expressions du premier et du second degré peuvent effectuer les calculs de manière incrémentale en utilisant la méthode du "forward differencing" [Chan89]. L'évaluation d'une racine carrée se faisant en chaque pixel, il est nécessaire de définir un module d'extraction de racines carrées.

Le Processeur Elémentaire du premier degré

La méthode du Forward Differencing évalue une expression du premier degré en un pixel (x,y), par rapport à la valeur calculée au pixel précédent. Nous pouvons distinguer deux cas pour le pixel précédent: il est soit en (x-1, y) soit en (x, y-1).

Considérons le premier cas.

Nous cherchons à évaluer $f1(x, y)$ en fonction de $f1(x+1, y)$:

$$f1(x-1, y) = a(x-1) + by + c = ax + by + c - a = f1(x, y) - a$$

donc nous avons

$$f1(x, y) = f1(x-1, y) + a.$$

Par le même raisonnement,

$$f1(x, y) = f1(x, y-1) + b.$$

L'affichage s'effectuant "en scan-line", en chaque début de ligne on calcule $f1(0,y) = f1(0,y-1) + b$ (soit une addition), et en chaque pixel $f1(x,y) = f1(x-1,y) + a$ (soit une addition, voir Figure 2.14).

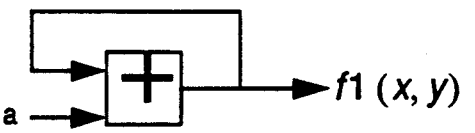


Figure 2.14 : Calcul incrémental de $f1(x,y)$ sur une ligne

Cette méthode permet de définir un processeur évaluant une expression du premier degré en chaque pixel (P.E.1). Pour la réalisation du P.E.1, il est nécessaire de déterminer sur combien de bits doivent être effectuées les additions. En supposant que la taille de l'écran est de $2^{10} \times 2^{10}$ pixels, la droite la plus proche de la verticale a une pente de 2^{10} : le rapport entre les coefficients a et b de l'expression est de 2^{10}. Les valeurs x et y varient sur l'écran entre 0 et 2^{10}, donc la valeur maximale que peut atteindre l'expression du premier degré est de l'ordre 2^{20} : nous utilisons alors 24 bits pour le calcul [Chai91].

Le Processeur Elémentaire du second degré

Comme pour la fonction du 1er degré, l'évaluation de la fonction du second degré $f2(x,y)$ doit être définie par rapport au résultat obtenu en x-1, y ou en x, y-1:

$$\begin{aligned} f2(x-1,y) &= a(x-1)^2 + by^2 + c(x-1)y + d(x-1) + ey + f \\ &= ax^2 + by^2 + cxy + dx + ey + f - 2ax - cy + a - d \\ &= f2(x,y) - 2ax - cy + a - d \end{aligned}$$

Nous pouvons donc calculer incrémentalement la fonction du second degré, puisque f2(x-1,y) est la valeur au point précédent et $2ax + cy - a + d = f_x(x,y)$ est une fonction du premier degré en x et y, pouvant être évaluée incrémentalement.

De façon similaire, nous pouvons définir $f2(x,y)$ comme étant la somme de $f2(x,y-1)$, valeur au point précédent, et de $2by + cx - b + e = f_y(x,y)$, une fonction du premier degré.

Dans le cas particulier de l'affichage en "scan-line", $f_y(x,y)$ est évaluée en début de ligne donc x=0 d'où $f_y(0,y) = 2by - b + e = f_y(0,y-1) + 2b$. De plus comme $f_x(x,y)$ est évaluée sur la même ligne (y est constant), il suffit de calculer en début de ligne $cy - a + d = \alpha(y)$ par rapport à la ligne précédente, pour déterminer la valeur de $f_x(x,y) = f_x(x-1,y) + 2a$.

Il faut donc calculer en chaque début de ligne $f_y(0,y) = f_y(0,y-1) + 2b$, $f2(0,y) = f2(0,y-1) + f_y(0,y)$ et $f_x(0,y) = f_x(0,y-1) + c$ (soit 3 additions/soustraction), puis en chaque pixel $f_x(x,y) = f_x(x-1,y) + 2a$ et $f2(x,y) = f2(x-1,y) + f_x(x,y)$ (donc 2 additions/soustractions).

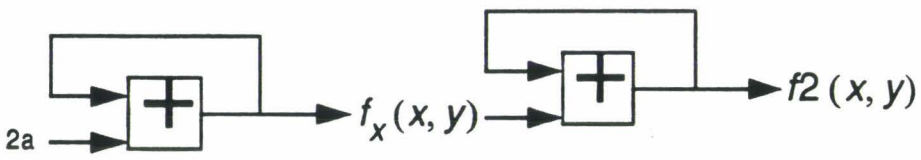


Figure 2.15 : Calcul incrémental de $f_2(x, y)$ sur une ligne

Une étude d'un Processeur Elémentaire calculant un expression quadratique (P.E.2) a été réalisée par S. Karpf [Karp89]. Un P.E.2 a une dynamique sur 48 bits, qui est le double de celle d'un P.E.1 puisqu'une expression du second degré est le produit de deux expressions du premier degré.

2.3.3 L'Extracteur de Racine Carrée

Afin de calculer la profondeur d'une quadrique en un pixel, il est nécessaire d'évaluer une racine carrée. Nous avons donc étudié les méthodes non récursives de calcul d'une racine carrée, les méthodes récursives étant difficilement utilisables dans une machine de rendu, puisque le nombre d'itération à effectuer pour obtenir une racine carrée dépend de la valeur en entrée.

Les premières méthodes étudiées devaient permettre d'obtenir une valeur proche de la racine carrée recherchée tout en ayant un coût inférieur au calcul exact.

- Les méthodes par approximation

L'approximation linéaire par intervalles définies par Hashemian [Hash90] impliquait des discontinuités d'ordre 1 qui étaient incompatibles avec le tri en profondeur des objets. Nous avons alors essayé une version modifiée de cet algorithme, qui permet d'obtenir une fonction C0 approchant la racine carrée. L'erreur maximale obtenue par cette méthode est de 6%.

Méthode d'Hashmian modifiée pour le calcul de la racine carrée R d'un nombre X
Si X est codé sur 2m bits ($2^{2m-1} \le X \le 2^{2m}$) *alors*

$$R(X) = \frac{X}{2^{m+1}} + 2^{m-1}$$

Si X est codé sur 2m-1 bits ($2^{2m-2} \le X \le 2^{2m-1}$) *alors*

$$R(X) = \frac{X}{2^m} + 2^{m-1} - 2^{m-2}$$

L'approximation quadratique par intervalles consiste à approcher la racine carrée par une fonction du second degré

$R(X) = \frac{1}{15} \times \left(\frac{1}{2^{m-2}} \times \left(\frac{X}{2^m} \right)^2 + 15 \times \frac{X}{2^m} + 2^{m+2} \right)$ définie sur l'intervalle $[2^{2m-2}, 2^{2m}]$.

L'erreur maximale est de 1,3%

Nous avons effectué des tests indiquant que l'erreur maximale de la racine carrée telle que nous l'avons utilisée en conversion, doit être inférieure à 1%. Nous avons alors étudié la méthode de correction [Hash90] décrite par :

$$R'(X) = R(X) - \frac{R(X)^2 - X}{2^{m+1}}$$

et nous avons conclu qu'il faut appliquer une fois cette correction sur le résultat obtenu avec l'approximation quadratique et deux fois sur le résultat de l'approximation linéaire.

- Une méthode de calcul exact [Cowg63].

Le principe de cette méthode est d'effectuer une division, dont le diviseur est également le résultat. Le calcul fournit un digit par itération, en commençant par les digits de poids fort.

H. Laporte [Lapo91] a défini un réseau cellulaire (Figure 2.16), qui calcule la racine carrée entière R(r1r2r3...) d'un nombre X(x1x2x3x4x5x6...)

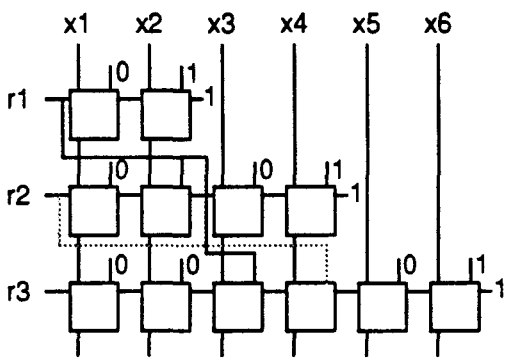


Figure 2.16 : réseau de calcul d'une racine carrée exacte

H. Laporte a montré que le coût d'implantation matérielle des méthodes par approximation, avec les cellules de correction nécessaires, était du même ordre de grandeur que le calcul exact effectué avec la méthode décrite ci-dessus, pour des nombres sur 48 bits.

Nous avons alors retenu la méthode de calcul de la racine carrée exacte pour définir le module d'extraction des racines carrées.

Le processeur de conversion doit pouvoir convertir les objets quadriques délimités par des plans, que ces objets soient surfaciques ou volumiques. Les modules de base ayant été décrits, nous allons maintenant définir l'organisation d'un processeur objet les utilisant, afin de convertir des objets quadriques surfaciques ou volumiques.

2.4 Le Processeur Objet Quadrique

La première approche employée pour définir un processeur de conversion de quadriques est directement inspirée des processeurs de conversion des facettes : l'objet est décrit explicitement par son contour. Par la suite, nous avons amélioré la définition du processeur de conversion en employant une seconde approche dérivée des méthodes d'affichage des arbres C.S.G.: l'objet est construit implicitement

pendant la conversion. Enfin, nous proposons aussi la définition d'un processeur de conversion de patchs quadriques, plus simple à réaliser puisqu'il ne convertit pas les objets volumiques.

2.4.1 La première génération

La première génération de processeur objet quadrique s'inspire de la méthode d'affichage des cylindres développée par l'équipe de Pixel-Planes [Fuch85]. En premier lieu, la quadrique coupée par des plans est décrite par un contour écran défini à l'aide de demi-plans et de coniques (l'intersection d'une quadrique et d'un plan est une conique). Ensuite, lorsque les calculs des différentes profondeurs et des normales ont été effectués en un pixel, il ne reste plus qu'à déterminer les valeurs de profondeur et la normale en fonction de la position du pixel dans des zones prédéfinies [Karp91].

La méthode de conversion

La première étape de cette étude est de définir le contour intrinsèque de la quadrique ($Q(x, y, z) = 0$) appelé silhouette, sans tenir compte des plans de coupe. Comme nous avons pu le remarquer, le calcul de profondeur nécessite l'évaluation de la racine carrée d'une fonction du second degré f2(x,y). Or si le résultat de cette fonction est négatif en un pixel (x,y), la racine carrée n'est pas une valeur réelle et donc la quadrique n'est pas présente.

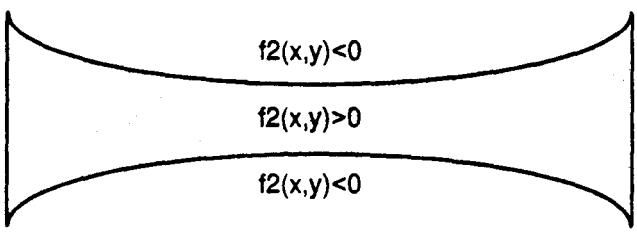


Figure 2.17: Silhouette d'une quadrique

La seconde étape est de déterminer la limite de contour de la quadrique coupée par un plan. La projection de l'intersection d'une quadrique et d'un plan est une conique d'équation $C(x, y) = 0$.

Nous pouvons "orienter" cette équation pour que l'évaluation de $C(x, y)$ soit positive aux points (x,y,z) du plan qui se trouve à l'intérieur de la quadrique: ces points vérifient $Pl(x, y, z) = ax + by + cz + d = 0$, qui est l'équation du plan, et $Q(x, y, z) \geq 0$.

Si l'on exprime la profondeur z du plan en fonction de x et y par

$$z = \frac{-a \times x - b \times y - d}{c} \qquad \text{(Exp.12)}$$

et qu'on la remplace dans l'expression du volume quadrique, nous obtenons l'expression de la conique orientée.

Un fois le contour de coupe déterminé et le contour de la quadrique défini, nous devons supprimer la partie de quadrique se situant du mauvais coté du plan; un demi-plan P est défini par une expression $P(x, y) \geq 0$, avec $P(x, y) = 0$ pour les points (x,y,z) appartenant à l'intersection du plan de coupe et du plan polaire de la quadrique. Ce demi-plan et les contours définis permettent alors de choisir la zone où la quadrique est présente. L'orientation de l'équation $P(x, y)$ est déterminée d'après la normale au plan

de coupe.

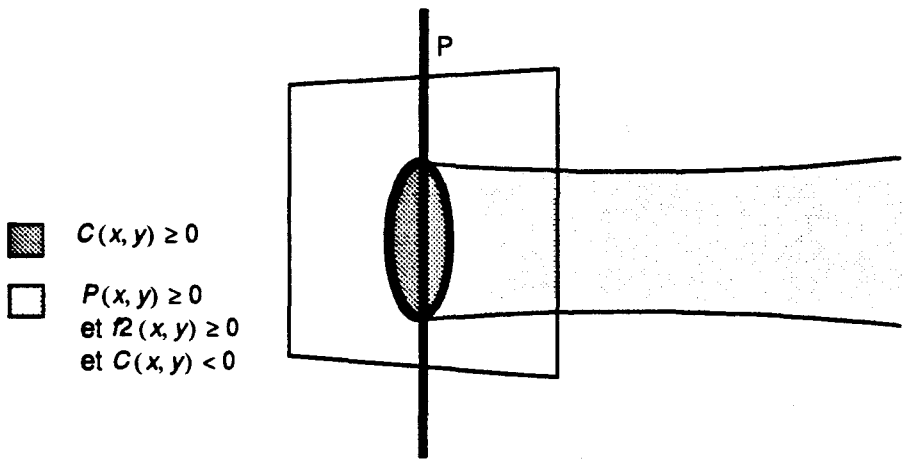


Figure 2.18 : contour dû à la coupe d'une quadrique

Nous pouvons définir ainsi 3 zones pour le contour de la quadrique coupée :

- le plan coupe la quadrique dans la conique si $C(x, y) \geq 0$
- la quadrique est totalement présente si $f2(x, y) \geq 0$, $P(x, y) \geq 0$ et $C(x, y) < 0$
- l'objet n'est pas présent dans le reste de l'écran

On peut remarquer que l'étude précédente ne tient compte que de la coupe par un seul plan, or pour utiliser (ou pour sculpter) une quadrique, nous en utilisons plusieurs. Nous allons étudier maintenant l'influence d'un autre plan de coupe lors de la définition du contour.

a) si il n'y a pas d'intersection entre les deux plans de coupe, ou si celle-ci est extérieure à la quadrique, alors le second plan n'intervient pas pour le contour d'intersection.

b) si un autre plan de coupe intersecte le plan en cours de traitement à l'intérieur du volume de la quadrique, alors celui-ci intervient lors de la définition du contour d'intersection précédent: la conique de contour $C(x, y)$ n'est pas complète, puisqu'elle est limitée par la projection de la droite d'intersection D des deux plans. La droite D, d'équation $D(x, y) = 0$, définit deux demi-plans servant au calcul final du contour de l'objet.

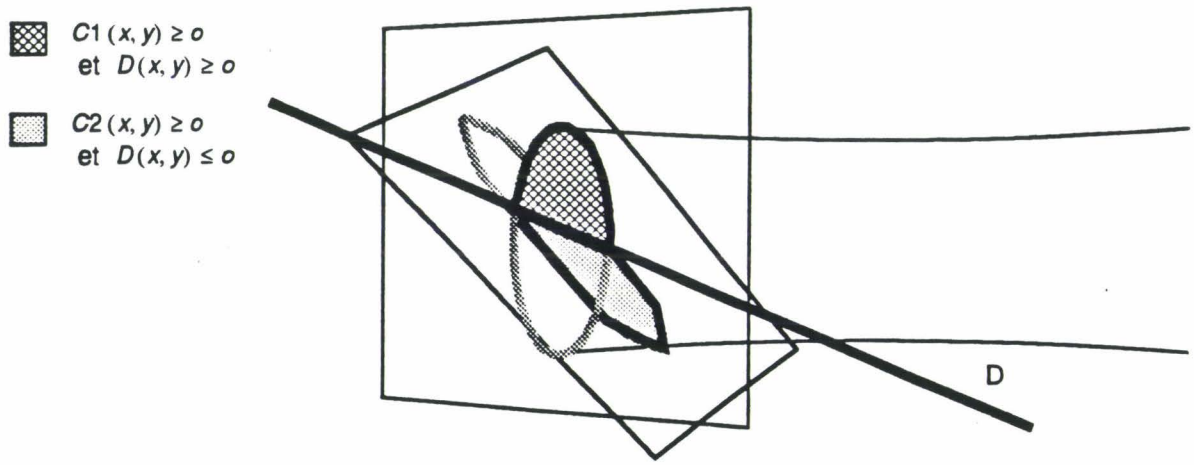


Figure 2.19 : Quadrique coupée par deux plans

L'orientation des plans de coupe et la définition de l'objet à afficher (i.e. objet plein ou patch) devant être pris en compte, l'algorithme final devient complexe :

Pour chaque pixel faire
Zf=ZQ_f et Zb=ZQ_b
Pour chaque plan avant i
Si $C_i \geq 0$ et si pour tout plan j ($j \neq i$), $inter_{ij}=1$ et $d_{ij} \geq 0$
alors si obj=0 Zf=Zp_i
sinon Zf= Zb
Pour chaque plan arrière i
Si $C_i \geq 0$ et si pour tout plan j ($j \neq i$), $inter_{ij}=1$ et $d_{ij} \leq 0$
alors si obj=0 Zb=Zp_i
sinon si Zb=Zf alors Zf=Zb=Zmax
sinon Zb = Zf
Si Zf=ZQ_f et Zb=ZQ_b {le pixel n'est pas sur un contour dû à un plan}
alors Si f2 ≤ 0 ou Si ∃ i telque $P_i \leq 0$
alors Zf=Zb=Zmax {le pixel est en dehors de l'objet}

avec, pour un pixel donné

ZF et *ZB*, les valeurs de profondeurs avant et arrière cherchées

ZQ_f et ZQ_b, les profondeurs avant et arrière de la surface quadrique

C_i, l'équation de la projection de l'intersection du plan i et de la quadrique ($C_i \geq 0$ dans la conique d'intersection)

d_{ij}, équation de la projection de l'intersection des plans i et j ($d_{ij} \geq 0$ quand le plan i est devant le plan j, donc $d_{ij} = -d_{ji}$)

Zp_i, profondeur du plan i

P_i, équation de la projection de l'intersection du plan i et du plan polaire de la quadrique

f2, fonction du second degré servant au calcul de Z_f et Z_b

L'algorithme utilise aussi des valeurs définies avant la conversion :

$inter_{ij}$ = 1 si les plans i et j ont une intersection dans le volume quadrique

obj = 0 pour un objet plein et différente pour un patch

La valeur de fond d'écran *Zmax* est utilisée pour les pixels hors contour.

L'algorithme étant trop complexe pour être implanté matériellement, même lors de l'utilisation de deux plans de coupe quelconques, on se limite aux objets quadriques coupés par deux plans parallèles (P_1 avant et P_2 arrière). L'algorithme utilisé est alors simplifié et peut être employé dans une unité de conversion :

Pour chaque pixel faire
Zf=ZQ_f et Zb=ZQ_b(approximations)
Si $C_1 \geq 0$
alors si obj=0 Zf=Zp_1
sinon Zf= Zb
Si $C_2 \geq 0$
alors si obj=0 Zb=Zp_2
sinon Zb= Zf
Si $C_1 \geq 0$ et $C_2 \geq 0$ et obj=0
alors Zf=Zb=Zmax
Si Zf=ZQ_f et Zb=ZQ_b {le pixel n'est pas sur un contour du à un plan}
alors Si f2 ≤ 0 ou $P_1 \leq 0$ ou $P_2 \leq 0$

La définition du P.O.Q première génération

Nous avons défini le schéma fonctionnel d'un Processeur Objet Quadrique (P.O.Q.), en nous limitant aux objets coupés par deux plans parallèles, mais en autorisant l'utilisation d'objet plein ou de patch. L'objet à afficher est alors délimité par un plan arrière et un plan avant.

L'algorithme utilise 5 expressions pour définir le contour :

C_1 est la conique d'intersection de la quadrique et du plan arrière

C_2 est la conique d'intersection de la quadrique et du plan avant

f2 définit la silhouette de la quadrique

P_1 et P_2 déterminent les demi-plans contenant la quadrique

Les profondeurs déterminées en chaque pixel sont :

ZQ_f et ZQ_b les profondeurs minimales et maximales de la quadrique

ZP_1 et ZP_2 les profondeurs du plan arrière et avant.

Pour l'éclairement, on associe à chaque pixel du contour une normale qui peut être :

NQ_f ou NQ_b les normales avant et arrière de la quadrique

NP_1 ou NP_2 les normales aux plans

Pour simplifier le schéma, nous avons appelé Processeur Elémentaire Quadrique (PEQ) une unité effectuant le calcul de $f1(x, y) \pm \sqrt{f2(x, y)}$, qui utilise donc un P.E.1, un P.E.2 et un extracteur de racine carrée. Le contrôle du contour est effectué par l'algorithme précédent, dans une unité spécialisée.

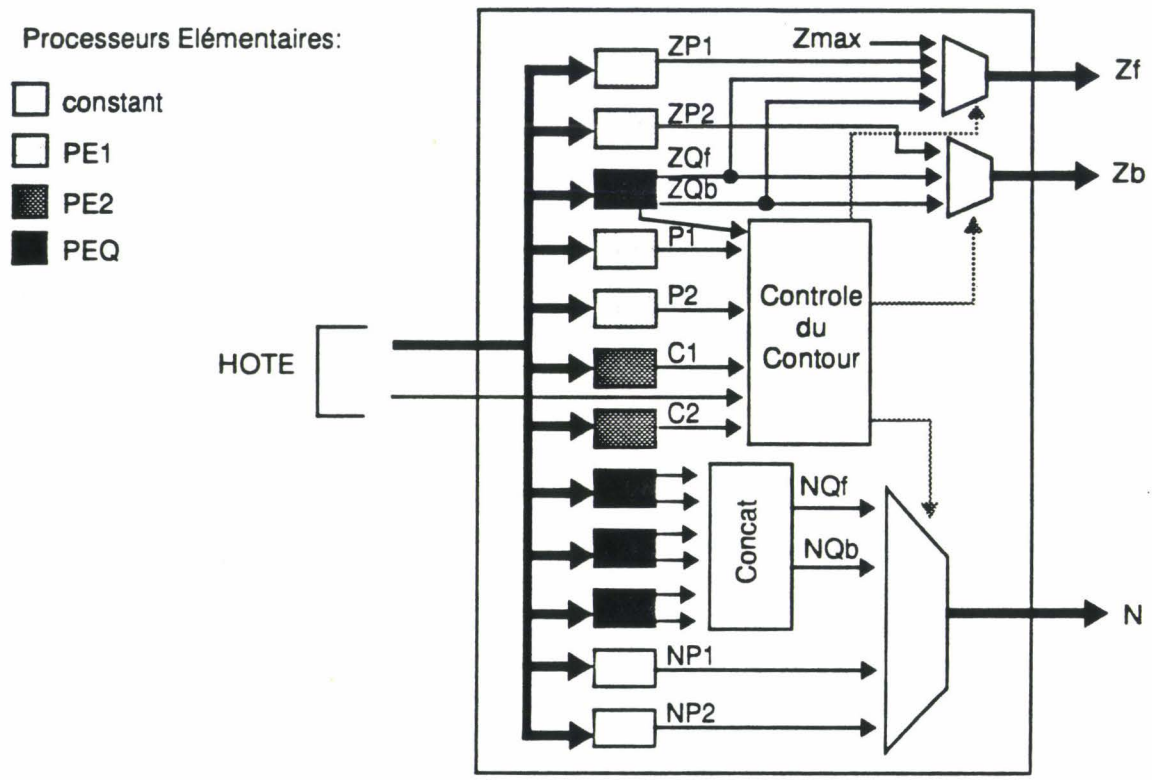


Figure 2.20 : Le Processeur Objet Quadrique 1ère génération

Limites

La conversion d'une quadrique par définition de son contour amène des limitations dans deux domaines:

- Lors de la modélisation d'une scène, on ne peut employer que des quadriques délimitées par deux plans de coupes, ce qui implique que l'on ne peut pas représenter d'objets complexes: les quadriques doivent être utilisées en tant que primitives lors de la modélisation. De plus les volumes quadriques ne peuvent pas être "sculptés", puisque les deux plans de coupes doivent impérativement être parallèles (sinon il y a risque d'intersection dans le volume quadrique).

- Le Processeur Objet Quadrique n'est pas très modulaire, puisque l'algorithme de définition du contour dépend du nombre de plans. La complexité matérielle du processeur est dû, en dehors de la conversion de la surface quadrique elle-même, à l'utilisation de P.E.2 pour les coniques d'intersection plan-quadrique et à l'implantation de l'algorithme de définition du contour.

2.4.2 La seconde Génération

La méthode d'affichage précédente étant trop limitée, nous avons décidé d'explorer une autre voie pour la conversion. Nous avons défini un nouvel algorithme, qui dérive des méthodes de résolution d'arbre C.S.G. au niveau pixel, dans lequel les plans de coupe de la méthode de conversion précédente définissent des demi-espaces. La primitive est alors définie par l'intersection de la quadrique et des

demi-espaces. Un module de coupe définissant l'intersection d'une quadrique et d'un demi-espace a été conçu afin d'être utilisé lors de la définition d'un processeur de conversion.

La méthode de conversion

- Les plans de coupe.

Un plan est déterminé par une équation du premier degré

$$Pl(x, y, z) = ax + by + cz + d = 0. \qquad \text{(Eq.6)}$$

La profondeur z_p en chaque pixel étant calculée par une expression du premier degré en x et y si $c \neq 0$, il est alors possible de déterminer si un point $P(x, y, z)$. se situe d'un côté ou de l'autre du plan en évaluant $z - z_p$ au pixel (x, y).

Le cas $c = 0$ est celui des plans perpendiculaires à l'écran, et bien qu'il ne soit pas possible de déterminer une profondeur pour les pixels, nous pouvons utiliser le plan pour délimiter la quadrique: L'équation d'un plan perpendiculaire à l'écran est du premier degré en x et y puisque le coefficient de la variable z est nul : nous pouvons donc l'identifier à l'équation d'une droite. Cette équation de droite $D(x, y) = 0$ peut alors être utilisée pour définir un contour écran en définissant deux demi-plans $D(x, y) \geq 0$ et $D(x, y) < 0$.

Un plan est donc utilisé pour définir deux demi-espaces, l'objet que l'on veut afficher étant contenu dans l'un des deux.

- L'algorithme que nous présentons ici s'applique successivement à tous les plans de coupe, quelles que soient leurs orientations. Il utilise en entrée un objet défini par zf et zb (profondeur minimale et maximale du volume de l'objet) et zp la profondeur du plan ou la valeur indiquant une position par rapport à un demi-plan associé à un plan perpendiculaire. Après l'application de l'algorithme, nous obtenons l'objet coupé avec sa nouvelle profondeur minimale et maximale (zf et zb).

```
Algorithme d'intersection avec un demi-espace
si le plan frontière du demi-espace est perpendiculaire
  alors si (zp<0) alors zf=zb=Zmax (valeur de fond d'écran)
si le plan est avant
  alors si (zp>zb) alors zf=zb=Zmax
                   sinon si (zp>zf) alors zf=zp et N=Np
si le plan est arrière
  alors si (zp<zf) alors zf=zb=Zmax
                   sinon si (zp<zb) alors zb=zp
```

Lors de l'étude précédente sur la définition de l'objet à afficher, nous avons pu remarquer que les quadriques vues de face définissent deux volumes distincts. La conversion des objets volumiques nécessite alors de couper ces deux volumes par les plans pour afficher l'objet désiré si les surfaces quadriques utilisées sont vues de face [Nyir92a]. La coupe de l'objet par un plan utilise alors l'algorithme sur chacun des volumes.

Tel que décrit ci-dessus, l'algorithme définit les objets volumiques puisqu'il dérive directement de l'affichage d'arbres CSG. Nous avons alors adapté celui-ci, à l'affichage des patchs quadriques, qui sont des surfaces quadriques délimitées par des plans.

Algorithme d'intersection avec un demi-espace pour les patchs
si le plan frontière du demi-espace est perpendiculaire
alors si (zp<0) alors zf=zb=Zmax (valeur de fond d'écran)
si le plan est avant
alors si (zp>zb) alors zf=zb=Zmax
sinon si (zp>zf) alors zf=zb et N = NQb
si le plan est arrière
alors si (zp<zf) alors zf=zb=Zmax
sinon si (zp<zb) alors zb=zf

Les patchs étant des surfaces, une seule application de l'algorithme décrit ci-dessus est utilisé en un pixel pour effectuer l'intersection avec un plan. Une unité unique pouvant traiter le cas des objets volumiques ou des patchs est définie en regroupant les deux algorithmes de calcul d'intersection.

Les autres améliorations

En reprenant la définition de la normale à la quadrique, nous pouvons exprimer chaque composante par $N_i(x, y, z) = f1_i(x, y) + Kz$ avec $f1_i(x, y)$ une fonction du premier degré et K une constante. Le calcul de la normale à la surface quadrique peut alors s'effectuer à l'aide de trois P.E.1, de six multiplieurs et de six additionneurs, au lieu de trois P.E.Q, ce qui diminue le coût de l'implantation puisque la complexité matérielle d'un Extracteur de Racine Carrée est nettement supérieure à celle de deux multiplieurs.

Nous pouvons lors de la conversion reporter le calcul de la normale après l'application des plans de coupe, en modifiant les modules effectuant ces opérations : lors de l'application d'un plan de coupe, on ne sélectionne plus une normale, mais seulement un numéro la désignant. Grâce au numéro de normale obtenu pour l'objet final, nous pouvons déterminer la normale quadrique parmi les deux possibles en chaque pixel, et n'effectuer que les calculs utiles.

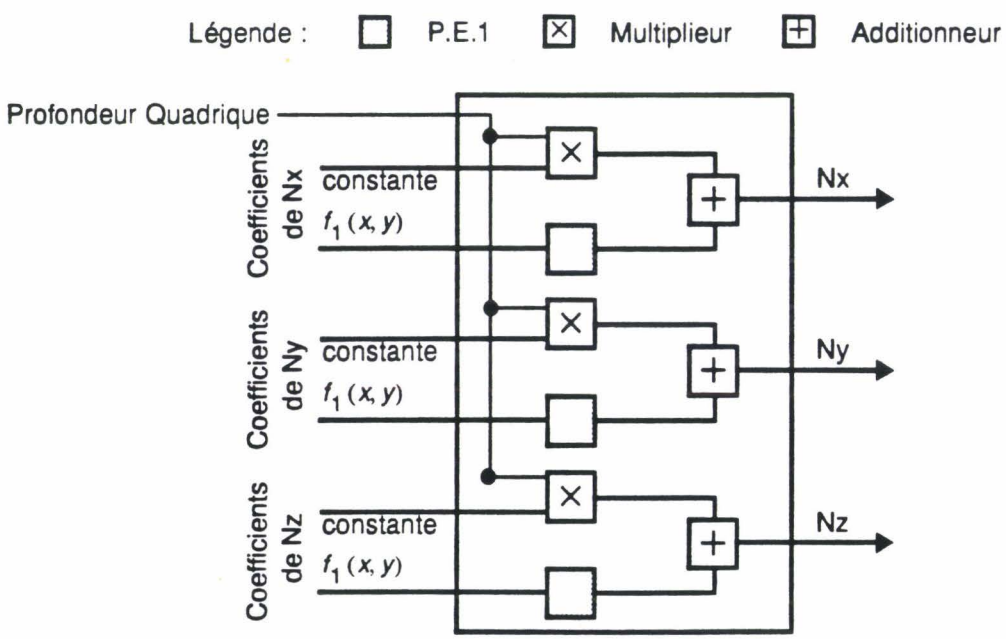


Figure 2.21 : Calcul de la normale quadrique

Cette amélioration permet de réduire le coût de l'implantation matérielle du processeur, en définissant un nouveau module qui ne calcule qu'une des deux normales quadriques possible en chaque pixel (Figure 2.21).

La définition du P.O.Q seconde génération.

La méthode de coupe générale permet d'utiliser un seul P.E.1. et un module effectuant l'intersection par plans de coupe. Du fait de cette modularité, le nouveau Processeur Objet Quadrique que nous avons défini peut comporter un grand nombre de plans de coupe. En fait le nombre de plans de coupe n'est limité que par la définition matérielle du P.O.Q et par son coût.

La version améliorée du P.O.Q à deux plans de coupe ne comprend plus alors que 6 P.E.1, 1 P.E.2, 1 E.R.C. pour ce qui est des unités de bases, 3 multiplieurs, 4 additionneurs/soustracteurs et un peu de logique pour la normale (Figure 2.22).

Le P.O.Q à deux plans de coupe que nous décrivons ci-dessus utilise un Z-buffer servant à regrouper les deux volumes obtenus en final. Dans le cas d'une utilisation autre que l'affichage de volume quadrique, il serait possible d'obtenir en sortie toutes les valeurs utiles.

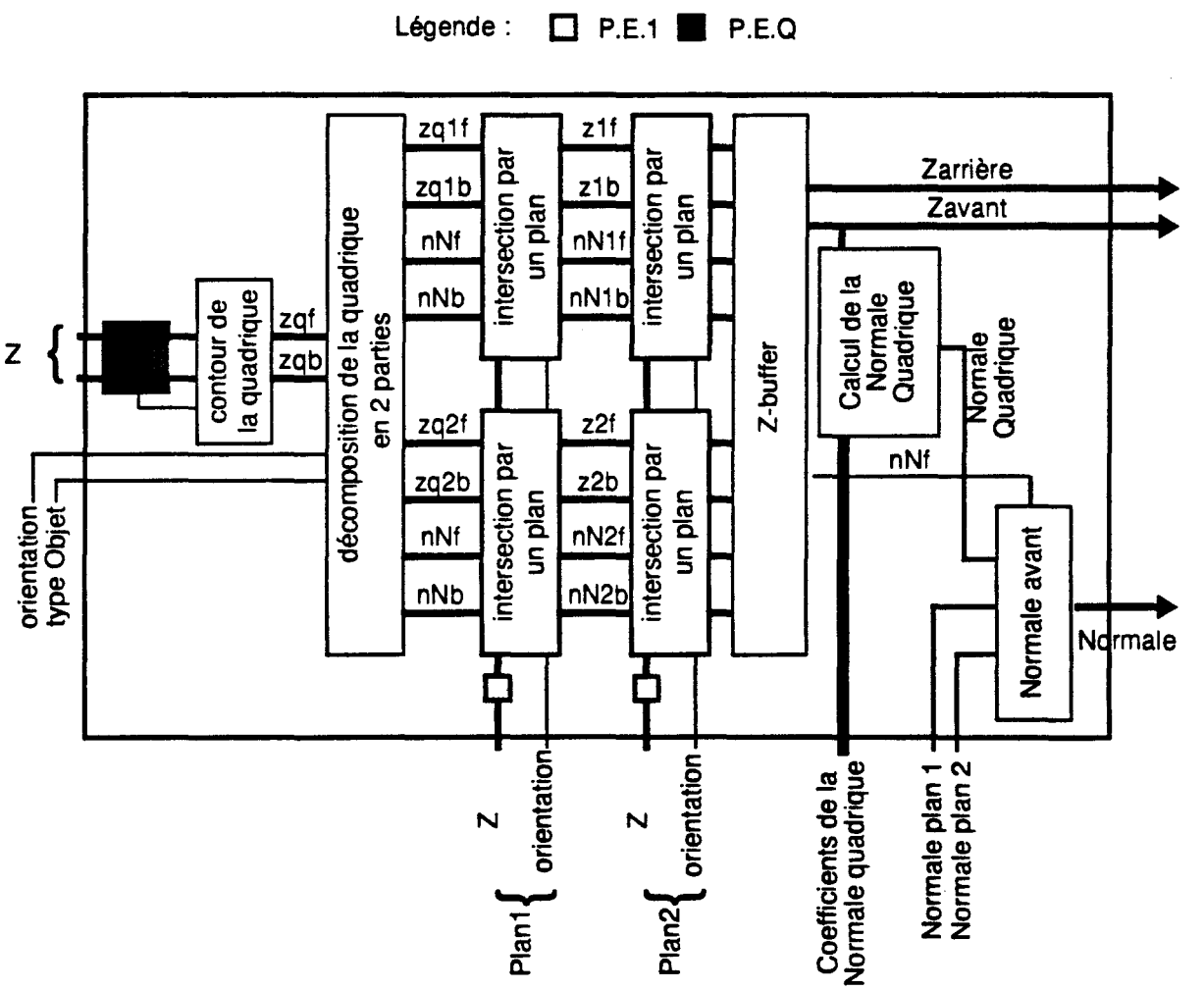


Figure 2.22 : Le Processeur Objet Quadrique seconde génération

Avantages

Nous pouvons donc remarquer que cette seconde génération de P.O.Q, permet l'utilisation des plans de coupe aussi bien pour délimiter la quadrique que pour la sculpter. Nous pouvons lors de la modélisation, utiliser les patchs quadriques sans être limité. Le rendu d'objets complexes sera ainsi réalisable par cette primitive.

Lors de la définition matérielle du P.O.Q., le nombre de plans de coupe à implanter sera choisi en fonction des primitives que l'on veut afficher.

2.4.3 La simplification du Processeur Objet Quadrique

Afin de réduire la complexité matérielle du processeur que nous venons de définir, nous avons simplifié les hypothèses de départ : puisque seul l'affichage de quadriques volumiques nécessite la gestion de deux volumes, on peut définir un Processeur Objet spécialement étudié pour convertir uniquement les objets surfaciques appelés des patchs quadriques.

Si l'on applique cette simplification à la seconde génération du P.O.Q. pour réduire le coût de l'implantation matérielle (Figure 2.23), les modifications apportées sont [Lapo93] :

- la suppression du module de décomposition en deux parties, du Z-buffer final et des normales aux plans de coupe.
- la simplification des modules de coupe par un plan

Une reformulation de l'algorithme de coupe par un plan, étudiée spécialement pour la définition du P.O.Patch a par ailleurs permis de paralléliser le découpage par les plans et a réduit la taille des chemins de données utilisés; Chaque module de coupe génére deux signaux EFi et EBi indiquant si la surface est effectivement coupé par le plan avant i à l'avant (EFi=0) ou coupé par le plan arrière i à l'arrière (EBi=0). Les signaux générés par les différents modules de plans de coupe sont ensuite combinés afin de déterminer si la partie visible du patch est la face avant de la surface ou la face arrière, ou si le patch n'est pas présent.

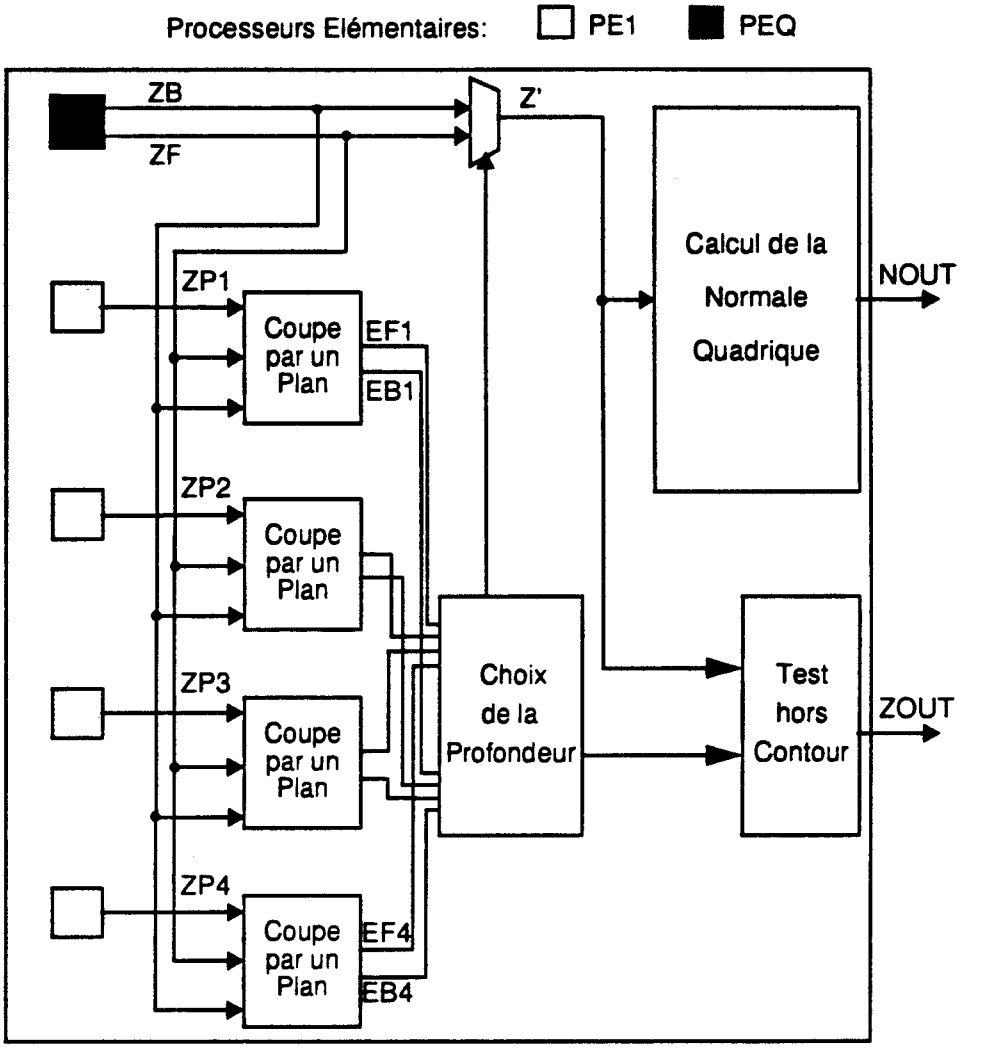


Figure 2.23 : le P.O. Patch Quadrique

2.4.4 Eléments pour l'utilisation des P.O.Q

Les Processeurs Objets que nous venons de définir travaillent en entiers en effectuant des évaluations d'expressions. Nous allons maintenant détailler deux types d'opérations différentes à effectuer sur les coefficients utilisés en conversion. Le premier type d'opérations est dû à la définition des processeurs. Les calculs s'effectuant en entier, les coefficients doivent être normalisés avant d'être convertis en entier. Les quadriques étant définies à partir d'expression, il est possible d'utiliser les Processeurs Objet dans une machine employant un parallélisme haut niveau; les coefficients doivent alors être modifiés en conséquence.

La normalisation des coefficients

Afin de garder le maximum de précisions sur les résultats de la conversion, il faut normaliser les valeurs initiales pour que les différents modules d'un processeur utilisent au maximum la dynamique disponible.

Trois types de coefficients sont à considérer :

• Les coefficients permettant l'obtention des profondeurs d'un objet. Les valeurs de profondeur des différents objets de la scène doivent être calculées à la même échelle si on veut les comparer. La même valeur multiplicative K_p doit être donc employée sur tous les coefficients des expressions de profondeurs de tous les objets.

La valeur K_p étant globale, elle ne peut être définie que par la connaissance de la valeur maximale prise par les profondeurs des objets dans une scène : elle est donc difficile à connaître dans le cas général. Cependant, l'application d'une perspective sur les objets d'une scène induit, dans le repère final de la transformation, une valeur maximale de profondeur D, égale à la distance entre l'observateur et l'écran

De ce fait, K_p peut être défini par $K_p = \frac{K_d}{D}$ avec K_d coefficient permettant d'employer le maximum de la dynamique dans l'unité de conversion, pour des calculs de valeurs comprise entre -1 et 1.

• Les coefficients employés pour le calcul des normales. Une normale ne dépend que de l'objet auquel elle est attachée et doit être normée lors des calculs d'éclairement par la méthode de Phong. Il est donc possible pour les calculs de multiplier les coefficients des expressions des composantes d'une normale par une valeur K_n, sans prendre en compte l'échelle utilisée pour les autres normales de la scène (dans les cas où une normale est calculée indépendamment de la valeur de profondeur de l'objet à convertir).

Puisque K_n ne dépend que des valeurs prisent par les trois coordonnées d'une normale, on calcule alors une valeur de K_n par normale. Pour une normale $\vec{N}(n_x, n_y, n_z)$, cette valeur est définie par :

$$K_n = \frac{K_d}{Sup(N_x, N_y, N_z)} \text{ avec } N_{i \in \{x,y,z\}} \text{ la valeur maximale atteinte par la coordonnée } i.$$

• Les coefficients des expressions définissant un contour. Si le contour d'un objet est prédéfini à l'aide d'expressions, seuls les signes des valeurs calculées sont utilisés. Il est donc possible de multiplier tous les coefficients d'une expression E de contour par une valeur K_c, indépendante des coefficients des autres expressions de contour.

Comme pour les coefficients des expressions des normales, on définit une valeur de K_c par expression E de contour en utilisant :

$K_c = \frac{K_d}{max(E)}$ avec $max(E)$, valeur maximale de l'expression E.

Les P.O.Q et le parallélisme au niveau écran

Un des avantages d'avoir défini la primitive quadrique à l'aide d'expression, est de pouvoir utiliser le Processeur Objet Quadrique (ou Patch quadrique) dans les machine exploitant un parallélisme au niveau de l'écran, comme le découpage de l'écran en zones ou en lignes, en ne modifiant que les coefficients utilisés en conversion.

• En effet, lorsque l'écran est découpé régulièrement en zones de $n \times m$ pixels (Figure 2.24), les zones sont juxtaposées et l'on passe donc d'une zone à une autre par une simple translation. De ce fait si les coefficients employés en conversion sont définis pour une zone $Z_{i,j}$, les nouveaux coefficients utilisés pour une autre zone $Z_{i+k,j+k'}$ sont calculés à partir des coefficients de la zone Z, par une simple translation de $kn \times \vec{x}$ et $k'm \times \vec{y}$.

Par exemple, si l'écran est découpé en zones de 128×128 pixels, le passage d'une zone $Z_{i,j}$o à la suivante $Z_{i+1,j}$ selon l'axe horizontal est effectué par une translation en $\hat{x}$ de 128 pixels.

Les coefficient a', b' et c' d'une expression $f1(x', y') = a'x' + b'y' + c'$ du premier degré défini pour la zone $Z_{i+1,j}$ sont calculés connaissant a, $b$et c de $f1(x, y) = ax + by + c$ la même expression définie pour la zone $Z_{i,j}$ par : $a' = a$, $b' = b$ et $c' = c + a \times 128$ puisque $x' = x + 128$ et $y' = y$.

En appliquant le même raisonnement pour les expressions du second degré, une expression $f2(x', y') = a'x'^2 + b'y'^2 + c'x'y' + d'x' + e'y' + f'$ de la zone $Z_{i+1,j}$ est définie à partir de $f2(x, y) = ax^2 + by^2 + cxy + dx + ey + f$ expression pour la zone $Z_{i,j}$ par $a' = a$, $b' = b$, $c' = c$, $d' = d + 2a \times 128$, $e' = e + c \times 128$ et $f' = f + a \times (128)^2$.

• Lors d'un découpage en lignes entrelacées (Figure 2.24.b), le traitement ne s'effectue que d'une ligne sur n en commençant à une ligne i donnée. La variable y' à utiliser dans les expressions de conversion est définie par $y' = n \times y + i$.

Une expression du premier degré initialement donnée par $f1(x, y) = ax + by + c$ devient $f1(x, y') = ax + (bn)y' + (c + bi)$ et une expression $f2(x, y) = ax^2 + by^2 + cxy + dx + ey + f$ du second degré est alors transformée en $f2(x, y') = ax^2 + (bn^2)y'^2 + (cn)xy' + (d + ci)x + (en + 2bni)y' + (f + bi^2)$.

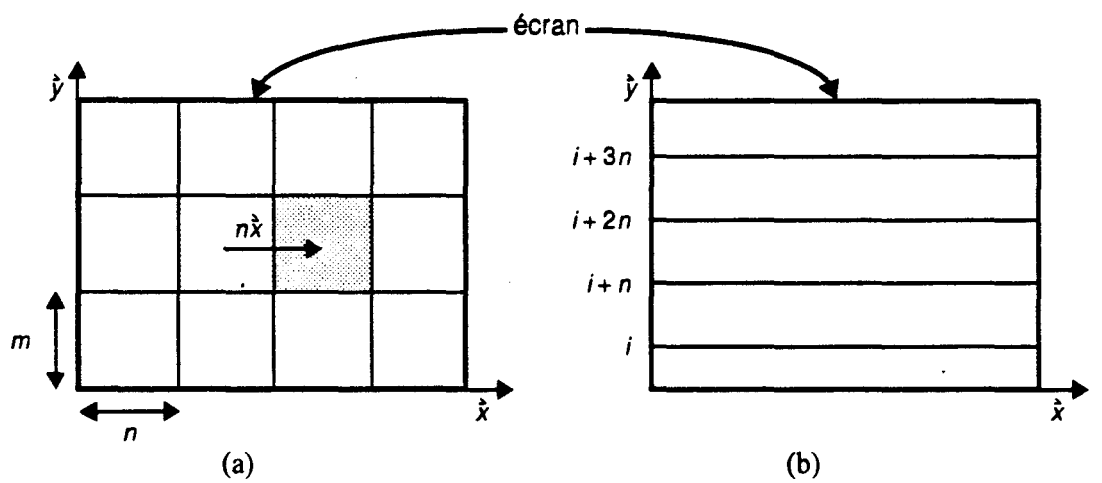


Figure 2.24 : Le P.O et le découpage écran

Les Processeurs Objet peuvent donc être utilisés par les machines employant un parallélisme au niveau écran sans être modifiés, puisque seuls les coefficients des expressions changent. S. Karpf a démontré que le parallélisme écran à employer par les machines de rendu géométrique est le découpage de l'écran en zones [Karp93], les processeurs que nous avons défini pourront donc parfaitement être utilisés avec ce choix de parallélisme haut niveau.

2.4.5 Résumé

Nous avons défini un module de conversion utilisable par une machine de rendu géométrique, permettant de traiter les objets définis à l'aide d'une surface quadrique et de plans de coupe permettant de limiter et même de sculpter l'objet. L'utilisation de la quadrique impose que les calculs d'éclairement

se fassent en chaque pixel, puisque la conversion des objets fournit la normale à la surface en chaque point et non une couleur. L'emploi du post-éclairement étant déjà envisagé pour les facettes, cette contrainte est parfaitement acceptable.

Parmi les différentes version de Processeurs Objets Quadriques, le P.O.Q. seconde génération est le plus général, puisqu'il permet la conversion des volumes ou des patchs définis à partir de quadriques. Son approche modulaire permet une implantation matérielle "facile".

La version Processeur Objet Patch quadrique est plus proche des concepts actuels du rendu, puisque la primitive est surfacique. Il est donc vraisemblable que la réalisation, dans un avenir proche, d'un processeur objet de conversion de quadriques utilise cette définition, qui est moins coûteuse et plus facile à intégrer. Il faut poursuivre les travaux sur les objets volumiques (polyèdres voir annexe A) pour déterminer les apports de ces primitives importantes pour le rendu.

Un point remarquable de ces processeurs de conversion est leurs entières compatibilités avec les différentes formes de parallélisme au niveau de l'écran, puisqu'ils emploient des expressions lors de la conversion. Il est possible de les intégrer dans toute machine utilisant un Z-buffer et un post-éclairement.

Chapitre 3 :
De nouvelles questions

Certains algorithmes utilisés en synthèse d'images sont basés sur l'utilisation de la primitive facette lors de l'affichage. L'utilisation d'une autre primitive amène de nouvelles questions à tous les niveaux de la création d'images de synthèse.

La modélisation est la première phase à étudier, puisqu'elle fournit les objets à convertir. Les machines actuelles utilisant la facette comme primitive d'affichage, les primitives de modélisation sont approchées par un ensemble de facettes. Afin d'employer la primitive quadrique lors de la phase de rendu, il faut que les objets modélisés puissent être décrits ou approchés à l'aide de cette primitive. Nous présentons les différentes voies envisagées, et les chemins qu'il reste à explorer, lors de la modélisation.

En marge de la conversion, certaines techniques sont employées pour améliorer la qualité des images générées. Ces méthodes sont appliquées afin de traiter trois problèmes différents : le réalisme global d'une scène, la qualité de la visualisation dans le milieu discret de l'écran et l'aspect des objets que l'on affiche. Un grand nombre des traitements utilisés emploie le fait que la primitive de conversion est la facette. Si l'on emploie la quadrique comme primitive ces méthodes ne sont plus applicables. Nous détaillons différents algorithmes utilisés avec la primitive facette, puis nous envisageons leur adaptation avec la quadrique.

3.1 Modélisation et Rendu

Dans notre cas, la scène que l'on veut visualiser doit être décrite par un ensemble de quadriques (et de facettes quand cela est nécessaire).

Nous avons défini des objets à l'aide de quadriques que nous employons en conversion, afin de construire des scènes de tests pour un simulateur utilisant les P.O.Q (voir le chapitre Simulation et Animation). Les objets sont définis par une union de quadriques volumiques et surfaciques, qui sont modélisées directement par les équations des surfaces quadriques et des plans de coupe. La modélisation utilisée a très vite montré ses limites, les objets construits ne peuvent pas avoir des formes trop complexes.

L'emploi des quadriques en rendu nécessite donc la définition d'un véritable modeleur permettant de définir des objets complexes à l'aide de volumes et de surfaces quadriques. Bien que certaines quadriques soient utilisées en modélisation, la plupart des objets d'une scène ne sont pas de ce type. L'utilisation des quadriques, ou de morceau de celles-ci, pour le rendu des objets a mené à revisiter la modélisation d'une scène.

La modélisation C.S.G. est un terrain de choix pour l'emploi des quadriques, puisque celles-ci sont simples à utiliser en tant que volumes initiaux. Nous commençons donc par l'étude de cette technique,

qui reste cependant limité à l'utilisation des volumes quadriques.

La modélisation surfacique peut être utilisée pour définir des surfaces quadriques, mais là aussi elle nous limite à l'utilisation d'une des deux formes possibles de la primitive.

3.1.1 La définition de volumes quadriques

Dans le cadre du C.S.G, les primitives de modélisation sont des volumes. L'intérêt de la quadrique est de pouvoir définir simplement ces volumes (Figure 3.1). L'utilisation de la quadrique comme volume initial est donc intéressante, si la primitive d'affichage est elle aussi quadrique.

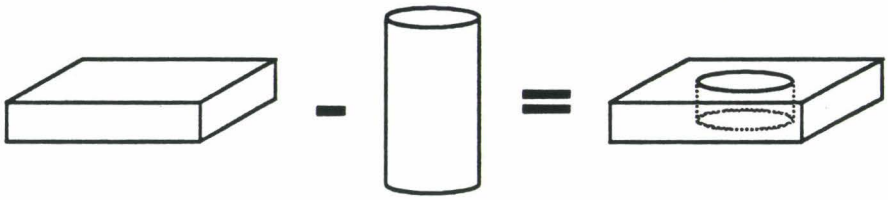


Figure 3.1 : La quadrique et le C.S.G.

Lors du rendu, si la primitive d'affichage est la facette, les objets doivent être décrits à l'aide de cette surface. Or le rendu d'une quadrique en facettes implique une perte de la qualité visuelle des objets et/ou une augmentation du nombre de primitives à afficher. Ceci explique la baisse de qualité de représentation, lorsque l'on visualise des objets définis par des arbres C.S.G à l'aide de la primitive facette.

Il est important de remarquer aussi que la phase habituelle de conversion des arbres C.S.G. en un ensemble de surfaces, nécessite des calculs d'intersections complexes, qui ajoutent des approximations lors du rendu et pénalisent l'interactivité de la phase de modélisation.

Les équipes qui étudient la primitive d'affichage quadrique dans le cadre du C.S.G. cherchent à éviter la phase de conversion des objets définis par des arbres C.S.G. en un ensemble de surfaces, phase jusqu'alors obligatoire, afin d'obtenir plus d'interactivité en modélisation et un rendu de meilleur qualité [Kede89][Klei93]. Une étude menée par Alain Preux [Preu91] a montré que l'affichage des objets définis par des arbres C.S.G. n'est pas très compatible avec le temps réel, puisque le traitement d'un arbre nécessite l'utilisation de listes pour mémoriser un certain nombre de profondeurs en chaque pixel.

La conclusion de cette étude sur la modélisation C.S.G montre que la primitive quadrique offre deux avantages :

- une qualité supérieure de représentation des objets.
- une diminution du nombre de primitives utilisées lors de la phase d'affichage.

3.1.2 La définition de surfaces quadriques

Deux approches sont envisageables dans le cas de ce type de modélisation : la modélisation directe des objets et l'approximation des surfaces utilisées par les modeleurs actuels par des patchs quadriques.

Modélisation directe de patchs

Cette approche consiste à remplacer les surfaces utilisées habituellement en modélisation (Beziers, B-spline, ...), par des quadriques.

En utilisant des valeurs de poids aux sommets et aux milieux des arêtes d'un tétraèdre (Figure 3.2), il est possible de définir toutes les quadriques dans la base de Bernstein. Le patch est alors défini par la surface quadrique coupée par les quatre côtés du tétraèdre. Deux tétraèdres de modélisation ayant un coté commun et les mêmes valeurs de poids sur ce côté et sur le milieu des arêtes, définissent deux patchs adjacents dont le raccord est en tangence. Il est donc possible de construire des objets dont la surface est continue, à l'aide de ces patchs quadriques. Cette méthode est efficace pour définir des objets, mais nécessite toutefois une manipulation de 10 valeurs (poids) par tétraèdre, ce qui est quelque peu difficile.

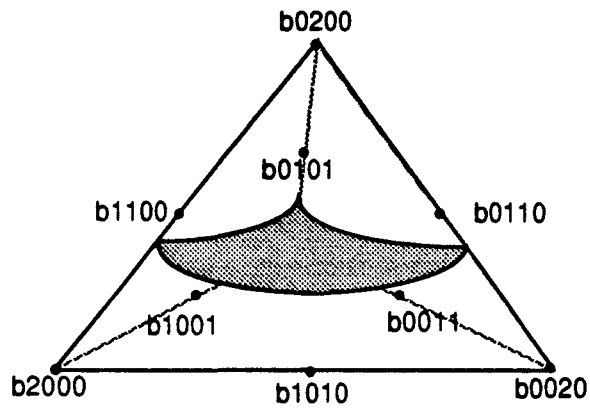


Figure 3.2 : Modélisation d'un patch quadrique dans un tétraèdre

Le raccord en tangence ne permet pas de contrôler les variations brutales d'éclairage survenant aux jonctions des patchs. Pour éviter ces phénomènes, il est possible de raccorder deux patchs quadriques en courbure (en continuité C2), mais il est nécessaire alors que ceux-ci soient deux morceaux de la même quadrique. La modélisation d'un objet quelconque ne serait plus possible dans ce cas.

M. Froumentin étudie la possibilité d'utiliser en modélisation des Macro-patchs, définis par un ensemble de plusieurs patchs quadriques [Frou93]. La manipulation de ces Macro-patchs est basé sur les concepts permettant de définir les carreaux de Bézier déjà utilisés en modélisation. Cette approche permet une utilisation plus simple de la quadrique en modélisation.

Approximation des surfaces de plus haut niveau

Actuellement, les surfaces de haut niveau utilisées lors de la phase de modélisation sont facettisées pour le rendu géométrique. Cette approximation est nécessaire du fait que les logiciels et les machines d'affichage utilisent la facette comme primitive. De nombreuses études en cours cherchent à définir une méthode générale d'approximation par carreaux quadriques.

W. Damhen [Damh89] puis B. Guo [Guo93] ont étudié des méthodes (Figure 3.3) consistant à remplacer chaque facette de l'approximation habituelle par des patchs quadriques.

- Damhen utilise la méthode de Powel-Sabin [Powe77] pour découper la facette en six sous-facettes, afin de définir par l'intermédiaire de six tétraèdres, six patchs quadriques parfaitement raccordés en tangence.

- Dans la méthode de Guo, chaque facette détermine la base d'un tétraèdre ; le patch quadrique modélisé alors doit passer par les sommets de la facette et posséder en ces points les normales définies

pour l'éclairement. Pour raccorder en tangence tous les patchs générés par cette méthode, Guo utilise des carreaux supplémentaires définis également à l'intérieur de tétraèdres.

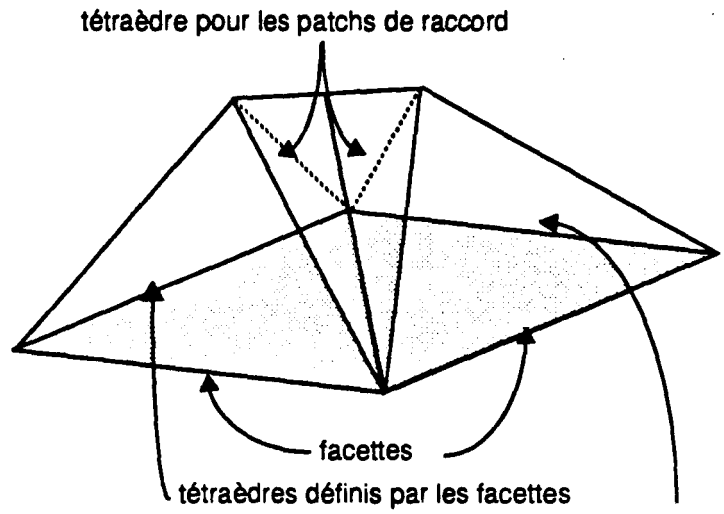


Figure 3.3 : Modèle d'approximation de Guo

Ces méthodes, bien que très coûteuses en nombre de patchs générés (4 à 7 patchs par facettes), démontrent donc qu'il est possible de remplacer les facettes dans les cas d'approximation.

3.1.3 Quadriques volumiques ou quadriques surfaciques

La quadrique permet de définir en modélisation des objets volumiques ou surfaciques. Le Processeur Objet Quadrique que nous avons défini, peut par ailleurs convertir ces deux types d'objets quadriques. Cependant les deux approches de modélisation habituellement utilisées (C.S.G. et surfacique [Brun90]) ne permettent pas de prendre en compte les deux aspects des quadriques en même temps.

L'utilisation d'objets volumiques semble naturelle pour l'affichage de scènes tridimensionnelles, nous avons également défini un Processeur Objets pour la conversion de volumes polyèdriques (voir annexe A). Les volumes sont convexes, puisque l'algorithme employé pour leur conversion utilise des intersections de demi-espaces. La conversion d'objets définis par des arbres C.S.G. ne permettant pas le temps réel, les objets volumiques ne sont utilisables que pour le rendu d'objets construit par unions de volumes de base. L'affichage des quadriques volumiques étant plus complexe que l'affichage de patchs, la réalisation de P.O.Q ne peut être envisagée que si les volumes quadriques sont réellement nécessaires pour le rendu de scènes. Dans le cas contraire, l'utilisation de Processeurs Objets Patchs moins chers à réaliser est suffisante.

Les scènes habituellement traitées sont définies par une modélisation surfaciques, les objets à afficher étant décrits à l'aide de surfaces, les primitives volumiques sont inutiles. Il semble donc que l'approche à adopter soit plutôt la conversion de surfaces quadriques. Cependant, l'utilisation d'objets quadriques volumiques lors du rendu n'est pas à exclure totalement, car l'affichage de ces volumes uniquement à l'aide d'objets quadriques surfaciques et de facettes peut poser des problèmes. Par exemple, pour fermer un objet quadrique surfacique comme un cylindre à l'aide d'une surface plane (Figure 3.4), il faut convertir des surfaces planes à bords coniques ; l'affichage d'un objet plan dont les bords sont coniques

nécessite soit la définition d'un processeur spécialisé dans la conversion de ce type d'objets, soit une modification du P.O.Patch afin d'y intégrer la conversion de ces objets. De plus, l'utilisation d'objets uniquement surfaciques nécessite dans le cas précédent, l'emploi de plusieurs primitives d'affichage. L'utilisation et la conversion d'un seul objet volumique est donc préférable.

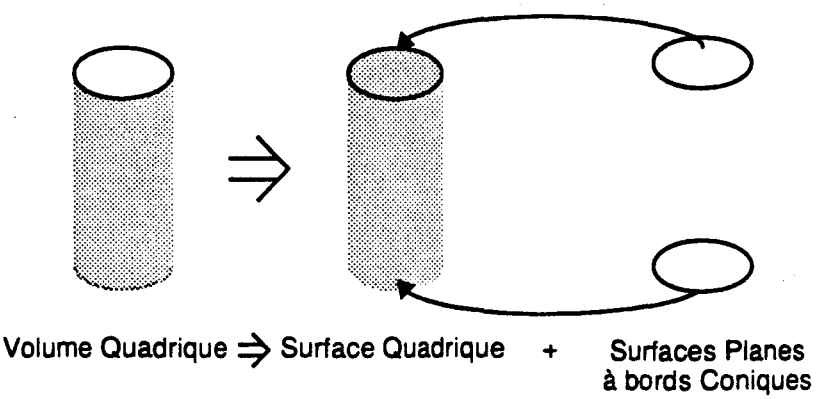


Figure 3.4 : Construction d'un volume à l'aide de surfaces

Il semble que les quadriques volumiques ne puissent pas être négligées pour l'affichage des scènes, surtout que la qualité volumique de la primitive peut être également exploitée dans le cadre d'un rendu avec application des ombres portées, comme nous le verrons par la suite. La réalisation d'un modeleur permettant l'emploi des volumes et des surfaces quadriques est dans ce cas nécessaire.

Afin de déterminer parmi les deux versions de processeurs objets (quadriques et patchs) celui qui sera réellement exploitable en rendu géométrique, il est nécessaire d'attendre des résultats sur la modélisation à l'aide de quadriques. Cette étude actuellement en cours (sujet de la thèse de M. Froumentin) permettra d'apprécier l'intérêt réel du volume pour le rendu.

3.2 L'amélioration de la qualité

Bien que l'utilisation de la quadrique améliore de fait la qualité de base des images générées, certaines méthodes peuvent permettre de se rapprocher encore de la réalité. Nous avons étudié les techniques déjà définies pour les facettes, afin de les adapter à la primitive quadrique.

Nous pouvons classer ces techniques en trois catégories :

• Les méthodes augmentant le réalisme global d'une scène affichée. Celles-ci s'appliquent généralement lors de l'éclairement des points par la prise en compte des ombres portées.

• Les algorithmes d'antialiassage traitant le passage du monde continu de la scène au monde discret de l'écran.

• Les techniques permettant d'afficher des objets ayant des caractéristiques particulières, comme une couleur non uniforme ou un aspect non régulier.

3.2.1 Les Ombres Portées

Afin d'augmenter le réalisme des images, il est important de prendre en compte l'ensemble des objets d'une scène. L'éclairement d'un objet contribue énormément à la représentation de sa forme, mais il

n'est pas suffisant pour rendre l'aspect tridimensionnel d'une scène. En effet, la reconnaissance des positions relatives des objets est généralement conditionnée par d'autres facteurs présent dans les scènes réelles (exemple : un objet posé sur le sol semble voler si on ne voit pas son ombre). Un des facteurs les plus important est la visualisation des ombres produites dans une scène [Wang92], il faut alors prendre en compte ce phénomène, si l'on veut que l'image finalement produite soit plus réaliste puisque plus proche de la représentation tridimensionnelle d'une image.

Les méthodes d'application des ombres portées

Les premières méthodes permettant de visualiser les ombres dans une scène, utilisent la géométrie de la primitive employée, en l'occurrence la facette.

Le principe de l'approche polygonale, étudié par Appel (1968) et par Bouknight et Kelley (1970), consiste à projeter les côtés des polygones générant les ombres sur les polygones à afficher, afin de définir des arêtes séparant le côté à l'ombre du côté éclairé. Il suffit, lors de la conversion, de déterminer si le pixel d'une facette à afficher se trouve à l'ombre, en testant les arêtes définies.

Un algorithme [Schn88] a notamment été proposé avec le parcours d'un arbre BSP permettant de trier les facettes par rapport à la source, afin de générer les ombres. Ces ombres sont projetées et mémorisées dans un arbre SVBSP par facette. L'affichage est effectué par parcours de l'arbre BSP pour l'élimination des parties cachées et par le parcours des arbres SVBSP pour la génération des ombres. Cet algorithme est plus une accélération de l'approche polygonale qu'une nouvelle méthode de génération des ombres.

La méthode à deux passes consiste dans un premier temps à regarder la scène depuis la source, en distinguant les parties de polygones visibles et celles cachées (donc à l'ombre). Dans un second temps, il est alors possible de définir des polygones d'ombres (représentant les parties à l'ombre) et de les ajouter à la scène. Lors du rendu, ces polygones d'ombre sont alors traités comme les autres polygones par les méthodes d'élimination des parties cachées, mais ils ne sont pas éclairés.

Les deux méthodes précédentes décrivent les deux approches possibles pour la définition des ombres portées suivant que l'on considère les objets ou les sources (Figure 3.5):

- Il est possible de définir pour chaque objet, les ombres qu'il engendre.
- Une première passe permet de connaître les objets, ou les parties d'objets visibles depuis la source.

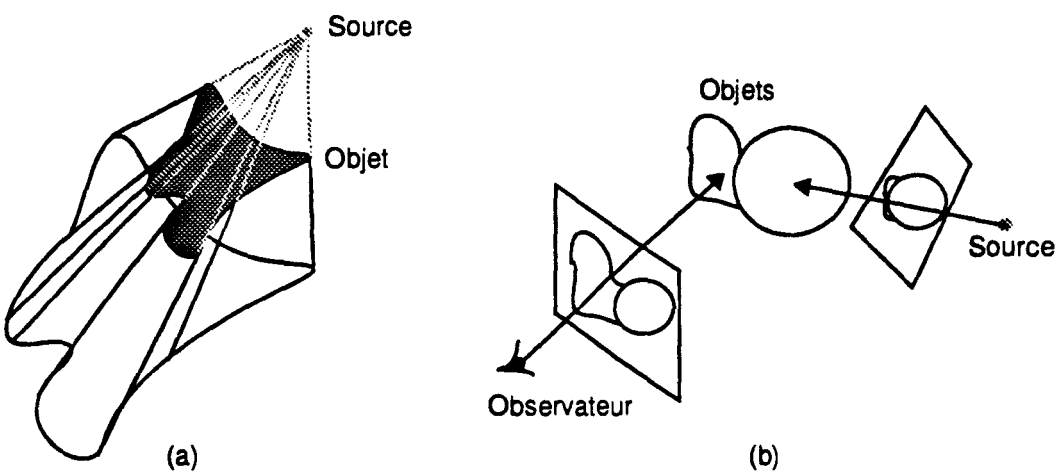


Figure 3.5 : Deux approches des ombres portées

Crow [Crow77] utilise la première approche (Figure 3.5 a), en déterminant pour chaque objet un volume d'ombre dans lequel les autres objets ne sont pas éclairés. Cette méthode appliquée aux facettes définit des volumes polyèdriques, infinis dans une direction. Lors du rendu il "suffit" de tester si le point à afficher (coordonnée du pixel et profondeur de l'objet) est dans un volume d'ombre polyèdrique dû à une source, pour savoir s'il est éclairé ou non par cette source. Le principal problème de la méthode est le nombre de volumes d'ombre engendrés, qui est le nombre d'objets composant la scène multiplié par le nombre de sources. L'équipe de Pixel-Planes 4 [Eyle88] a utilisé la méthode de Crow, en définissant un volume polyèdrique à l'aide de deux tableaux, le premier mémorisant la profondeur avant et le second la profondeur arrière en chaque pixel. Le test d'appartenance d'un point à un volume est effectué par rapport aux deux profondeurs mémorisées. Nous avons défini un Processeur Objet Polyèdre utilisable en rendu géométrique[Nyir92b], qui permet lors de l'affichage en scan-line de définir directement le volume (profondeur avant et arrière) en chaque pixel, sans passer par une mémorisation intermédiaire : l'application des ombres portées est effectuée au rythme de la conversion.

Williams [Will78] a défini une méthode utilisant la seconde approche (Figure 3.5 b). Les objets de la scène sont convertis une première fois en prenant la source comme point de vue, et les valeurs de profondeurs obtenues après élimination des parties cachées sont mémorisées dans un tableau annexe. Lors du rendu, les objets sont convertis dans le repère écran et en chaque pixel une valeur de profondeur Z est calculée. L'algorithme permettant de déterminer si un pixel (x, y) est éclairé ou non, passe par le changement de repère de l'écran vers la source du point $P(x, y, z)$ associé au pixel (coordonnées du pixel et valeur de profondeur), puis par un test des valeurs des nouvelles coordonnées (x', y', z') obtenues par rapport aux valeurs mémorisées dans le tableau. Un pixel est éclairé, si le point $P'(x', y', z')$ est à la même profondeur que le point $P''(x', y', z'')$ correspondant du tableau (le test est effectué avec une certaine marge d'erreur due aux calculs du changement de repère). Plusieurs inconvénients sont inhérents à cette méthode : l'utilisation d'une mémoire annexe par source éclairant la scène, et surtout la mauvaise qualité des ombres due aux changements de repère (plusieurs pixels peuvent utiliser la même valeur d'entrée dans le tableau annexe).

Les ombres portées et les quadriques

Nous ne pouvons utiliser les premières méthodes, puisqu'elles ont été conçues spécialement pour les facettes. En effet, elles utilisent le fait que les bord des facettes sont des droites et qu'il est possible de diviser la primitive à l'aide d'autres droites. La méthode de Williams [Will78] semble être intéressante puisqu'elle est utilisable quelque soit la primitive, et donc avec la quadrique. Nous n'avons pas retenu cette méthode compte tenu de ses inconvénients et parce qu'une autre méthode nous a semblé plus appropriée avec la primitive que nous avons développé.

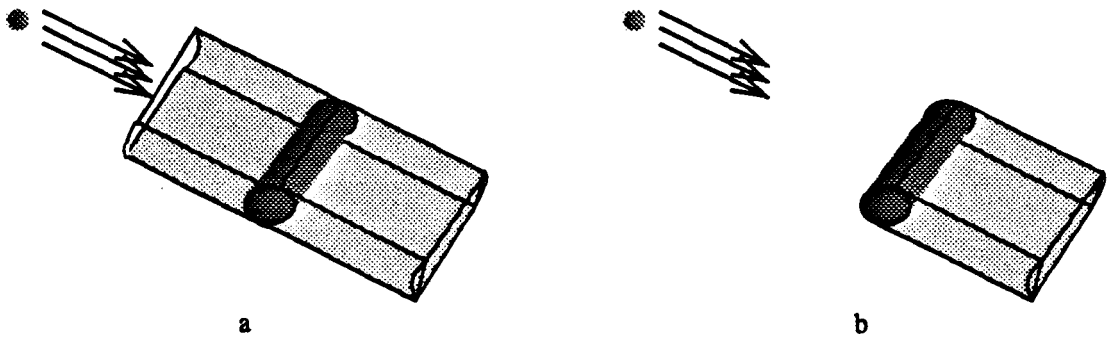
La méthode de Crow [Crow77] utilisant des volumes d'ombre et la primitive quadrique pouvant être volumique, nous avons étudié une adaptation de la méthode au rendu de quadriques.

Revenons aux principes de la méthode : il s'agit de définir à partir d'un objet, le volume à l'intérieur duquel les points sont invisibles pour une source.

En premier lieu, il convient de déterminer le type de volume en fonction de l'objet. Lors de la définition de la première version du P.O.Q., nous avons constaté que le contour écran d'une quadrique coupée peut être défini à l'aide d'un ensemble de coniques et de demi-plans. Ce contour étant déterminé par la projection orthographique de l'objet, nous pouvons en déduire que la quadrique coupée observée par une source directionnelle possède un contour qui peut être décrit de la même manière.

Un volume (Figure 3.6.a) construit par la translation de ce contour dans la direction de la source, est

alors défini à l'aide d'un ensemble de volumes quadriques et de demi-espaces. Le volume d'ombre de la quadrique (Figure 3.6.b) peut alors être défini comme étant ce volume limité à l'objet [Nyir92b].



a b

Figure 3.6 : Volume d'ombre d'une quadrique

Si la source lumineuse est ponctuelle, on se ramène au cas d'une source directionnelle par application d'une transformation perspective sur la scène vue depuis la source (la source est rejetée à l'infini). Ensuite par transformation inverse des volumes d'ombres obtenus (la transformation est bijective sur son domaine d'application), on se place à nouveau dans l'espace initial de définition de la scène.

Pour définir le volume d'ombre, nous ne pouvons utiliser que des quadriques coupées par des plans, aussi il est important que le volume d'ombre total soit défini uniquement à l'aide de ces primitives. La définition du volume d'ombre total d'une primitive quadrique doit donc passer par la détermination de sous-volumes représentables par des quadriques coupées, qui seront ensuite combinés.

Si l'on considère l'étude effectuée sur le contour de la quadrique, ainsi que les remarques précédentes, nous sommes tout naturellement amenés à considérer 3 sortes de volume:

- Le premier de ces volumes est bien évidemment le volume de la quadrique à afficher:

Le volume de la quadrique à afficher V_q est facilement représentable puisqu'il découle directement de notre représentation de la primitive sous forme d'objet plein.

- Le volume défini par la silhouette de cette quadrique:

La silhouette conique de la quadrique permet de définir un volume quadrique V_s, construit par la translation de cette silhouette suivant la direction de la source. Un volume V'_s défini par la limitation de V_s au niveau de la quadrique, peut être obtenu en coupant V_s par le plan polaire. Cette limitation n'est pas suffisante, puisque l'objet de départ est aussi coupé par des plans. Afin de prendre en compte les limitations de la quadrique à afficher, nous utilisons l'intersection de ses plans de coupe et de son plan polaire pour déterminer les droites D_g. Les plans P_g construit par la translation des droites d_g suivant la direction de la source lumineuse, servent alors à couper V'_s. Le volume finalement obtenu est appelé V_0 (Figure 3.7).

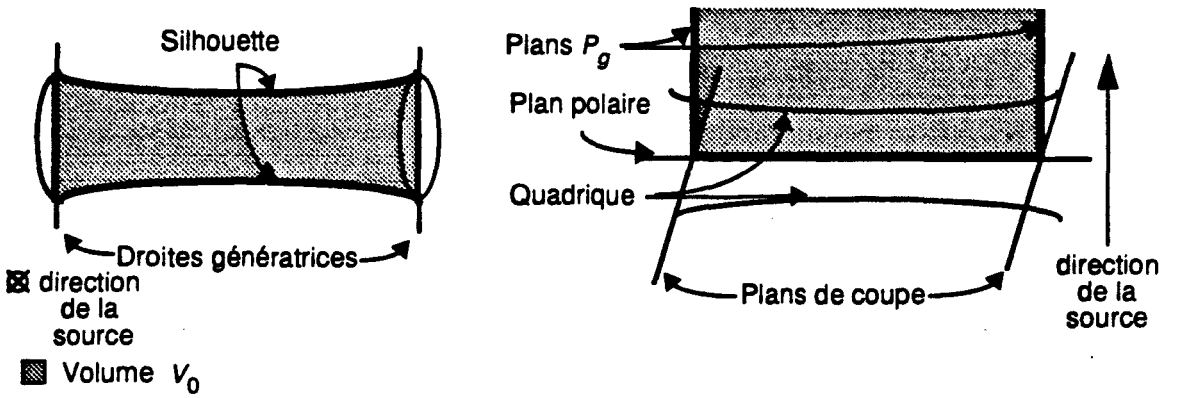


Figure 3.7 : Volume défini par la silhouette

• Les volumes induits par les intersections des plans et de la quadrique.

L'intersection d'un plan P_i et d'une quadrique définit une conique. Le volume quadrique V_{pi} construit par la translation de cette conique suivant la direction de la source et coupé par le plan P_i est le volume d'ombre induit par la coupe de la quadrique par le plan P_i.

Il faut alors prendre en compte le fait que deux plans de coupe P_i et P_j s'intersectant dans la quadrique ne gênèrent seulement qu'une partie des deux volumes V_{pi} et V_{pj}. Pour cela, nous déterminons la projection de la droite d_{ij} intersection de P_i et P_j puis nous construisons le plan P_{ij} par translation de cette droite suivant la direction de la source. Le plan P_{ij} est alors utilisé pour couper V_{pi} et V_{pj} afin d'obtenir les deux parties cherchées appelées V_i et V_j (Figure 3.8).

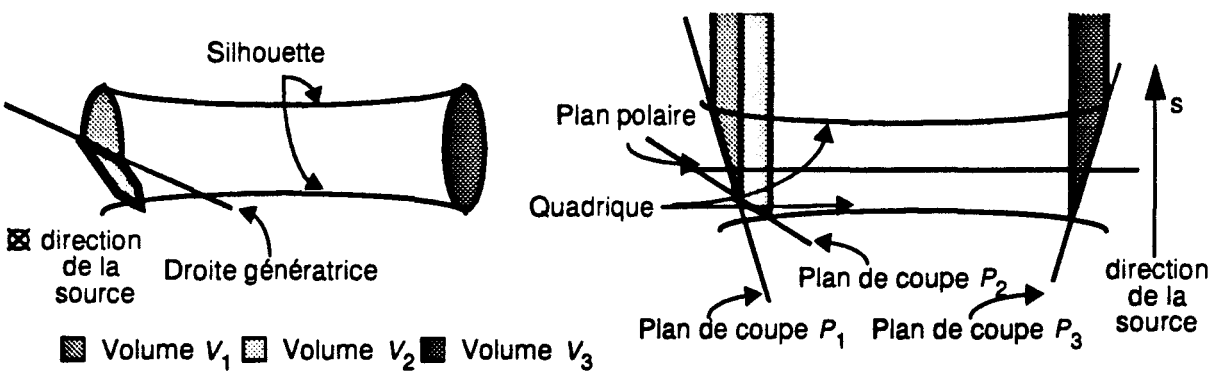


Figure 3.8 : Volumes définis par les intersections plans-quadrique

En utilisant ces différents volumes, nous allons pouvoir définir le volume d'ombre total de la quadrique. Il faut tout d'abord connaître le type de l'objet que l'on affiche, puisque le volume d'ombre Vt d'un objet plein et d'un patch ne sont pas les mêmes.

- Le volume d'ombre total généré par un objet volumique défini par la quadrique est :

$$V_{tv} = V_q \cup V_0 \cup_{i=1\ldots n} V_i \text{ avec } n \text{ nombre de plans de coupe.}$$

Nous pouvons remarquer que les objets se trouvant à l'intérieur du volume de la quadrique ne seront pas visibles lors de l'affichage. Le volume $V'_{tv} = V_0 \cup_{i=1\ldots n} V_i$ peut donc être utilisé à la place de V_{tv} pour les calculs des ombres portées.

Le volume d'ombre total à utiliser n'étant composé que par des unions de sous-volumes primaires, un point est dans V_{tv} s'il est au moins dans l'un des volumes V_0 ou V_i.

- Pour les patchs quadriques le problème est tout autre, puisque l'objet est surfacique. Si l'on compare les volumes d'ombre V_{tv} d'un objet volumique et V_{tp} d'un patch défini par la même quadrique coupée, on s'aperçoit que le second peut être déduit du premier.

Si l'on considère les quadriques vues de côté, par rapport au repère de la source (la quadrique est convexe en profondeur), le volume d'ombre est :

$$V_{tp} = (V_q \cup V_0 \cup_{i=1\ldots n} V_{fi} \cup_{i=1\ldots m} V_{bi}) - \cup_{i=1\ldots n} (V_{fi} \cap V_q) - \cup_{\substack{i=1\ldots n \\ j=1\ldots m}} (V_{fi} \cap V_{bj})$$

avec V_{fi}, (resp. V_{bi}) les volumes d'ombre dus aux plans avant (resp. arrières) par rapport au repère utilisé et n, m le nombre de plans avant et arrières

Le volume d'ombre des quadriques vues de face (la quadrique comporte une concavité en profondeur), s'obtient en séparant l'objet en deux parties, à l'aide du plan polaire. Chacune des parties est alors convexe et son volume d'ombre peut être défini comme précédemment.

Connaissant les volumes d'ombre défini par les objets quadriques, il suffit, lors de la conversion d'une scène et après élimination des parties cachées, de déterminer si le point (x, y, z) visible est à l'ombre (à l'intérieur d'un des volumes V_t) ou non.

Le cas des objets volumiques est le plus simple, puisque les volumes d'ombre total sont composés par l'union des volumes primaires qui peuvent être traités séparément (Figure 3.9). Si une scène ne comporte que des objets volumiques, un point est à l'ombre s'il est à l'intérieur d'un des volumes primaires d'un des volumes d'ombre total calculés pour la scène (Figure 3.10).

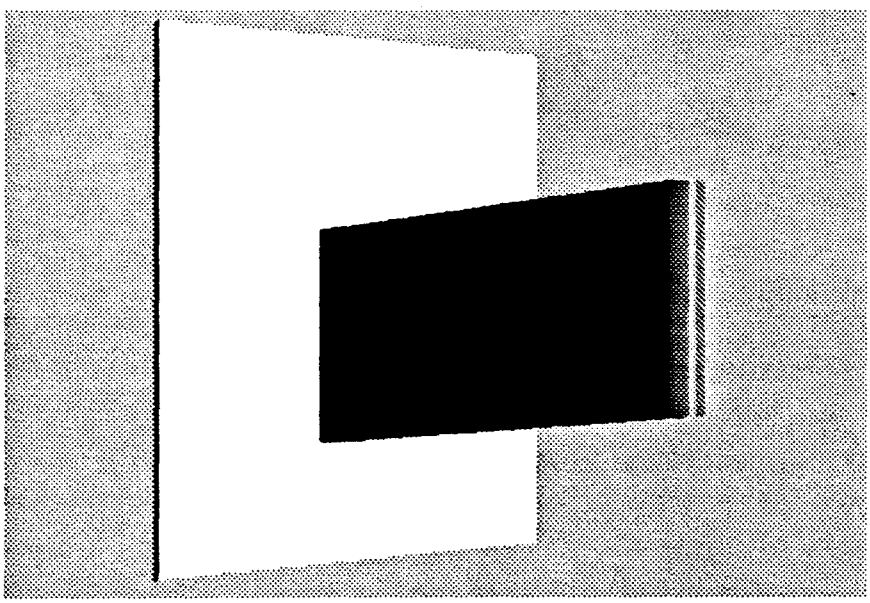


Figure 3.9 : Un volume cylindrique et le volume d'ombre associé
(image réalisée à l'aide du simulateur décrit au chapitre 4)

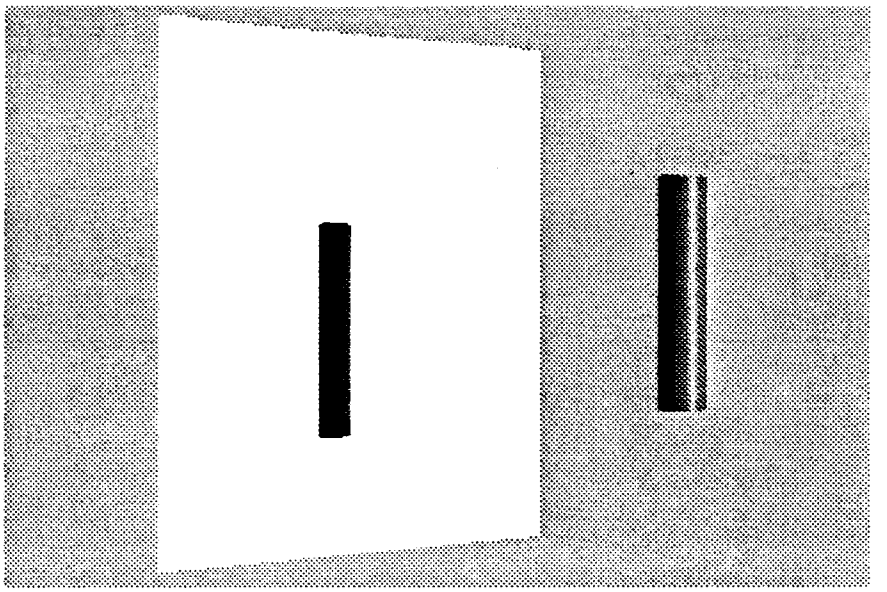


Figure 3.10 : Un volume cylindrique et son ombre portée sur un plan
(image réalisée à l'aide du simulateur décrit au chapitre 4)

Le volume d'ombre d'un patch quadrique étant décrit par une expression C.S.G., le test permettant de déterminer si un point est éclairé ou non est complexe. Il est possible d'utiliser deux solutions pour connaître la position d'un point par rapport au volume d'ombre total d'un patch : la construction du volume V_{tp} en question puis un test d'appartenance, ou la composition de tests sur les volumes primaires.

Application de le méthode retenue à la définition de spots

En reprenant le principe de la méthode de Crow [Crow77], nous pouvons aussi définir des volumes de lumière. Un point d'un objet est éclairé par la source lumineuse d'un spot, s'il se trouve dans le volume de lumière associée (Figure 3.11), un spot étant défini par une source lumineuse et un volume dans lequel la source est active.

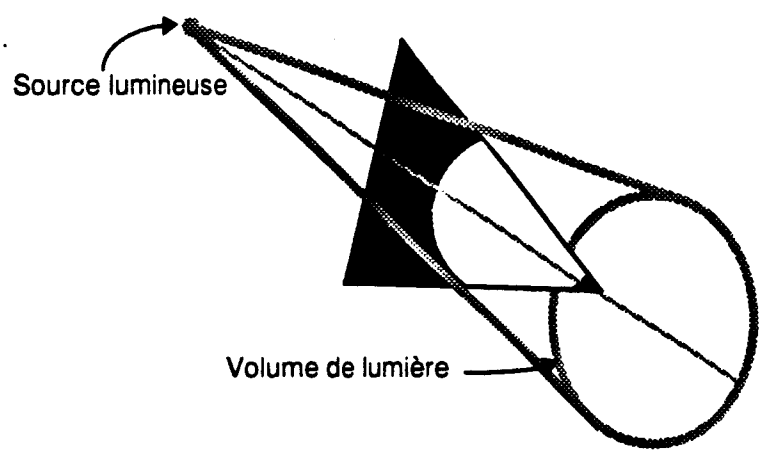


Figure 3.11 : Exemple de spot lumineux défini par un volume cônique

Le test permettant de savoir si un point est dans un volume de lumière, peut être effectué après élimination des parties cachées, mais doit être menée avant l'application des ombres portées, la source du spot générant aussi des volumes d'ombre.

Conclusion

Nous avons décrit une méthode d'application des ombres portées des quadriques en employant la méthode de Crow, qui est utilisable si l'on veut augmenter le réalisme des images générés. La méthode est d'autant plus facile à implanter que les P.O.Q définis permettent de convertir des volumes quadriques, qui sont la base des volumes d'ombres à employer avec la primitive d'affichage quadrique. De plus l'utilisation des P.O.Q dans une machine de rendu géométrique offre la possibilité de définir des spots lumineux, ceux-ci employant des volumes de lumière pouvant être traités de façon similaire aux volumes d'ombres.

3.2.2 L'Aliassage.

Dans le cas de la synthèse d'image, l'aliassage se situe surtout au niveau de la discrétisation des objets dans l'espace de l'écran composé de pixels, phase qui correspond à la conversion.

Malgré une amélioration de la qualité de l'image due à l'utilisation d'une primitive de plus haut niveau que la facette, il reste le problème incontournable du passage du monde réel (objets) à l'affichage sur un écran discret (pixels). Des méthodes permettant de traiter ce problème ont été définies dans le cadre du rendu à l'aide de facettes. L'utilisation d'un rendu à base de quadriques amène à se poser à nouveau la question pour cette nouvelle primitive.

D'après le théorème de Shannon utilisé dans le traitement des signaux, si la fréquence d'échantillonnage utilisée pour discrétiser un signal est inférieure à deux fois la fréquence maximale de ce signal, alors les hautes fréquences ne sont pas correctement restituées et peuvent apparaître comme des basses fréquences dans certain cas (Figure 3.12). L'interprétation de ce théorème en synthèse d'image se traduit par l'apparition d'un effet de crénelage sur le contour des objets, et la disparition certains objets plus petits que le pas d'échantillonnage. En effet, la grille d'échantillonnage employée est superposée à la grille des pixels (la fréquence d'échantillonnage de la conversion est la distance entre deux pixels consécutifs), ce qui implique une perte d'informations entre deux points voisins.

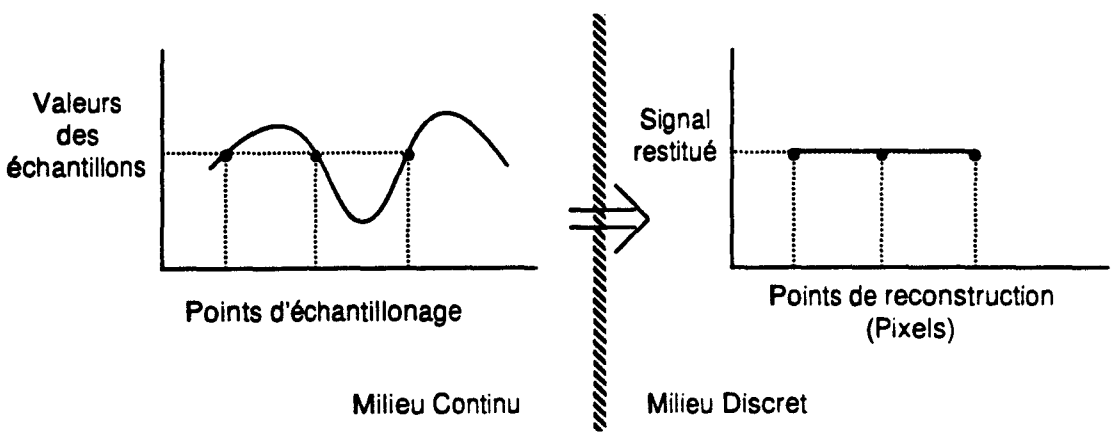


Figure 3.12 : L'aliassage

Il existe deux catégories d'algorithmes tentant de pallier ce problème: les méthodes utilisant les informations sur l'objet dans le milieu continu (pré-filtrage) et celles recherchant des informations supplémentaires sur l'objet en augmentant la fréquence d'échantillonnage lors de la conversion (sur-échantillonnage).

Le pré-filtrage.

Les méthodes de pré-filtrage [Ghaz87][Preu92] ne sont utilisées que pour l'antialiassage de bord des primitives, le contour de l'objet étant connu à priori. Le principe est de définir en chaque pixel un pourcentage de couverture P_0, qui correspond à la surface commune au pixel et à l'objet. Bien que les pixels soient carrés (ou rectangulaires), certaines méthodes utilisent des "pixels ronds", afin de prendre en compte l'influence des pixels voisins et ainsi obtenir un lissage de l'image (Figure 3.13).

La couleur associé à un pixel, où un objet ayant une couleur C_o est présent, est $C_p = P_o \times C_o$.

- Si plusieurs objets non recouvrant sont présent en un même pixel, la couleur du pixel est la somme du produit des différents taux de couverture obtenus avec les couleurs correspondantes $C_p = P_o \times C_o + P_1 \times C_1 + \ldots + P_n \times C_n$.

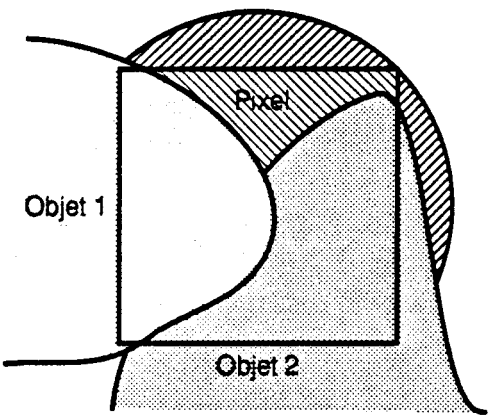


Figure 3.13 : Objets couvrant partiellement un pixel

- Lors de la présence de plusieurs objets se recouvrant en un pixel, un taux de couverture ne peut plus être utilisé directement puisqu'il est défini à partir d'un objet et d'un pixel, sans tenir en compte des autres objets de la scène.

Cette méthode, comme nous l'avons indiqué, est utilisée uniquement pour améliorer la qualité visuelle des bords des primitives d'affichage, les arêtes implicites dues à l'intersection de deux primitives ne sont donc pas traitées.

• Le pourcentage de couverture d'une facette est facile à déterminer, puisque le contour de celle-ci est décrit par un triangle, qui peut être représenté par l'intersection de trois demi-plans. Pour chaque demi-plan, on calcule un pourcentage de couverture en utilisant la distance D_i du centre du pixel à la droite limite du demi-plan en question (Figure 3.14) :

soit $F(x,y) = a \times x + b \times y + c = 0$, l'équation d'une droite D_i et soit (x_p, y_p) les coordonnées du centre P d'un pixel. La distance de P à la droite D_i est alors:

$$\frac{a \times x_p + b \times y_p + c}{\sqrt{a^2 + b^2}}$$

Cette méthode est donc facilement applicable lors de la conversion des facettes par équations, puisqu'il suffit de diviser les coefficients de l'équation des droites de contour par $\sqrt{a^2+b^2}$.

On effectue ensuite un traitement définissant le pourcentage de couverture de la facette en fonction des trois taux de couverture obtenus et des pentes des droites limitant les demi-plans.

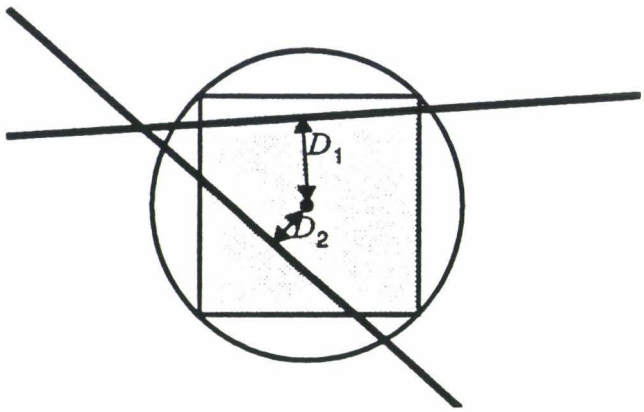


Figure 3.14 : Calcul du taux de couverture d'une facette

L'adaptation de la méthode d'antialiassage des bords de polygones à la primitive quadrique nécessite le calcul de la surface commune à une conique et à un pixel (le contour écran d'une quadrique est une union de coniques).

• Le calcul surfacique étant trop complexe pour être employé lors de la conversion, nous avons envisagé d'utiliser une méthode similaire au calcul du taux de couverture d'une facette, à partir des distances des centres des pixels aux côtés de la primitive.

Malheureusement, si la distance du centre d'un point à une droite est facile à déterminer, il n'en va pas de même pour la distance du centre d'un pixel à une courbe conique à partir de l'équations de celle-ci. Comme exemple, nous pouvons citer le calcul de la distance d'un point P à un cercle C de centre O l'origine du repère et de rayon R qui est défini par $Ds = \sqrt{x_p^2+y_p^2}-R$; L'équation du cercle C est $F(x,y) = x^2+y^2-R^2 = 0$ et $F(x_p, y_p)$ n'a alors aucun rapport avec la distance Ds.

La définition même de la notion de distance pose des problèmes puisque plusieurs points d'une courbe conique peuvent être à distances égales d'un point quelconque.

L'emploi des méthodes d'antialiassage par pré-filtrage ne pourrait être adapté qu'avec la première définition de Processeur Objet Quadrique, les versions suivantes générant des arêtes implicites à la conversion, lors des intersections de la quadrique avec ses plans de coupe. Pour la seconde génération de processeurs de conversion des quadriques, il faut étudier d'autres techniques d'anti-aliassage

Le sur-échantillonnage

Le principe du sur-échantillonnage est basé sur l'utilisation de plusieurs points d'échantillonnage au sein d'un pixel, afin d'obtenir une grille échantillonnage plus fine que la grille de restitution (l'écran); le calcul de la couleur des pixels est effectuée ensuite à l'aide d'un filtre passe-bas. Les défauts liés à l'échantillonnage d'un signal continu sont toujours présents mais amoindris.

Différents types de répartitions des points d'échantillonnage au sein d'un pixel peuvent être employés :

- La répartition régulière (Figure 3.15.a) permet d'utiliser les méthodes d'affichage scan-line puisque les points d'échantillonnage peuvent être considérés comme des centres de sous-pixels. On effectue

alors soit un calcul direct des valeurs de couleurs des objets aux points d'échantillonnage (avec une grille de conversion plus fine que la grille des pixels), soit une accumulation au sein d'une table (A-Buffer) des valeurs successives de couleur obtenues en déplaçant les objets d'une distance équivalente au pas d'échantillonnage (en gardant la grille de conversion calquée sur les centres des pixels).

- La répartition aléatoire (Figure 3.15.b) de type loi de Poisson, permet une meilleure définition de la grille d'échantillonnage puisqu'elle est redéfinie en chaque pixel, mais empêche la conversion des objets en scan-line puisque les coordonnées des points d'échantillonnage ne sont plus placées sur une grille régulière.

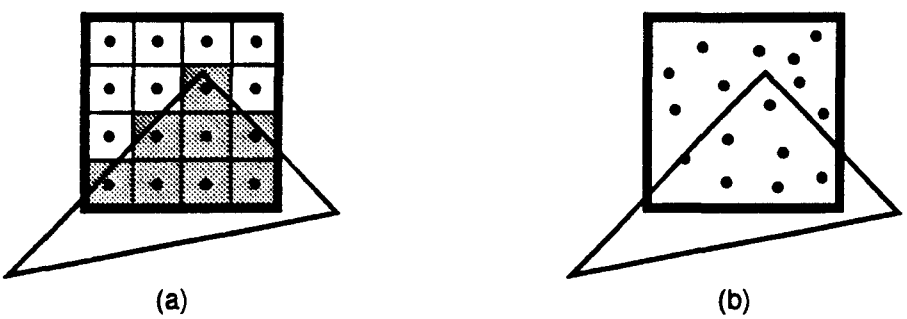


Figure 3.15 : Répartition des points d'échantillonnage

Le principal défaut de ces méthodes est de ne pas prendre en compte, à chaque image, les petits objets, ce qui produit des clignotements lors des animations (les petits objets sont définis comme ayant une taille inférieure au pas d'échantillonnage lors de l'utilisation d'une grille régulière). La répartition aléatoire des points d'échantillonnage au sein des pixels ne permet pas de garantir l'affichage des objets ayant une taille inférieure aux pixels. Cependant les petits objets sont en général mieux pris en compte avec cette répartition qu'avec une répartition régulière utilisant le même nombre de points d'échantillonnage.

Le grand avantage de ces méthodes est qu'elles peuvent être utilisées avec toute primitive de conversion. Un autre avantage non négligeable est le traitement direct des arêtes implicites, qui surviennent inexorablement.

• Pour les facettes, on utilise généralement une grille d'échantillonnage régulière car sa mise en œuvre est plus simple que pour une grille aléatoire. Si on utilise la méthode par accumulation des valeurs, la conversion d'une facette par équations permet alors d'effectuer directement une translation de l'objet en modifiant certains coefficients utilisés par l'algorithme de conversion.

• L'utilisation du sur-échantillonnage avec la primitive quadrique permet l'emploi de toutes les versions de processeur de conversion de quadriques, y compris celles générant des arêtes implicites. La méthode par accumulation est aussi simple avec la primitive quadrique qu'avec la primitive facette, puisque les processeurs que nous avons défini utilisent des expressions du premier et du second degré et qu'une translation peut être effectuée en modifiant certains coefficients, comme nous avons pu le faire remarquer dans le chapitre précédent.

3.2.3 Les Textures

La couleur intrinsèque d'un objet n'est en général pas uniforme sur toute la surface. Cette diversité de teinte entre deux pixels voisins est due, soit à la couleur que l'on choisit initialement à la modélisation, soit à la matière dans laquelle l'objet a été taillé. Evidemment, l'emploi du terme texture permet de regrouper ces deux notions dont les fondements sont communs, mais dont les domaines d'applications sont différents.

La couleur que l'on choisit initialement pour l'objet est définie sur la surface de celui-ci. Nous avons donc une définition bidimensionnelle de cette texture, que l'on cherche à plaquer sur une surface tridimensionnelle.

Si l'on définit un objet comme ayant été taillé dans une certaine matière, la couleur d'un point de la surface ne dépend que de la position de ce dernier. Il convient alors d'avoir un cube de cette matière (ou une autre représentation volumique) pour affecter une teinte à un pixel.

Les machines d'affichage utilisant la primitive facette, les méthodes permettant de texturer les objets peuvent prendre en considération cette caractéristique du rendu. Le problème si l'on veut employer la quadrique lors du rendu est alors de tenir compte de ses spécificités afin de d'afficher des objets n'ayant pas une couleur uniforme sur leurs surfaces.

Nous allons étudier les méthodes employées pour appliquer les textures, puis nous indiquerons les voies qu'il semble intéressant d'explorer afin de générer des quadriques texturées. L'aspect synthèse de textures bidimensionnelles ou tridimensionnelles n'est pas abordé dans chapitre, puisque notre but n'est pas la modélisation d'objets, mais leur affichage.

Textures bidimensionnelles

Les textures bidimensionnelles sont définies dans une table suivant un repère $(T, \vec{U}, \vec{V})$. Dans la suite de ce document, nous appellerons couleurs les valeurs de la table, bien que celles-ci puissent être considérées comme des valeurs de couleurs ou comme des valeurs utilisables lors des calculs d'éclairement (exemple: valeurs pour la perturbation des normales). L'utilisation de textures bidimensionnelles implique que l'on définisse des méthodes permettant de passer de l'espace texture, à l'espace objet et vice-versa. Si les objets texturés sont définis par des expressions paramétriques de surfaces, les deux variables peuvent être alors utilisées comme coordonnées dans la table des textures [Heck86]; cette approche utilise la paramétrisation des objets. D'autres méthodes ont été définies pour plaquer des textures sur des objets complexes sans tenir compte de leurs formes paramétriques afin de généraliser l'algorithme à employer.

- Méthodes utilisant la paramétrisation

On considère une surface S telle que :

$$X = X(u, v)$$
$$Y = Y(u, v)$$
$$Z = Z(u, v)$$

les paramètres u et v représentent en chaque point(X, Y, Z) définis, les coordonnées de la couleur dans le plan de la texture choisie.

La première approche (Figure 3.16 a) pour plaquer une texture, est de balayer l'espace des paramètres u et v dans leur domaine de définition, pour générer l'ensemble des points de la surface. Nous avons alors directement en chaque point calculé les coordonnées de la couleur. Le problème de cette méthode est que l'affichage est effectué sur un écran discret (puisque composé de pixels) à partir de l'espace

objet. Il est donc très difficile d'échantillonner l'espace des paramètres u et v pour que les valeurs X et Y obtenues lors des calculs soient effectivement les coordonnées d'un pixel.

Une seconde technique (Figure 3.16 b) est d'essayer de déterminer en chaque point (X, Y, Z), les valeurs de u et v correspondantes. Il faut donc pour chacunes des expressions paramétriques trouver une expression inverse. Si l'on considère une expression paramétrique comme étant une fonction à deux variables, l'expression inverse doit être alors la fonction inverse. Malheureusement les expressions paramétriques emploient généralement des fonctions trigonométriques et les expressions inverses ne permettent pas de définir des valeurs u et v uniques pour un point donné.

Une méthode (Figure 3.16 c) a été proposée afin de résoudre ce problème d'inversion. A partir du plan écran, on emploie deux niveaux de résolution [Catm80].

La première étape consiste à fixer une fonction $F_u(u)$ dans l'expression paramétrique des coordonnées pixels $X(u, v)$, afin de restreindre le nombre de solutions pour u. Ensuite, on fixe une (ou plusieurs) fonction $G_u(X)$ telque $F_u(G_u(X)) = X$. Le résultat de $G_u(X)$ donne alors une valeur possible de u.

La seconde étape consiste à reprendre la seconde expression $(Y(u, v)$ et à la représenter sous la forme $Y(u, v) = Y_X(v)$ pour une fonction $G_u(X)$ fixée. On peut alors définir $G_v(Y)$ de la même façon que $G_u(X)$.

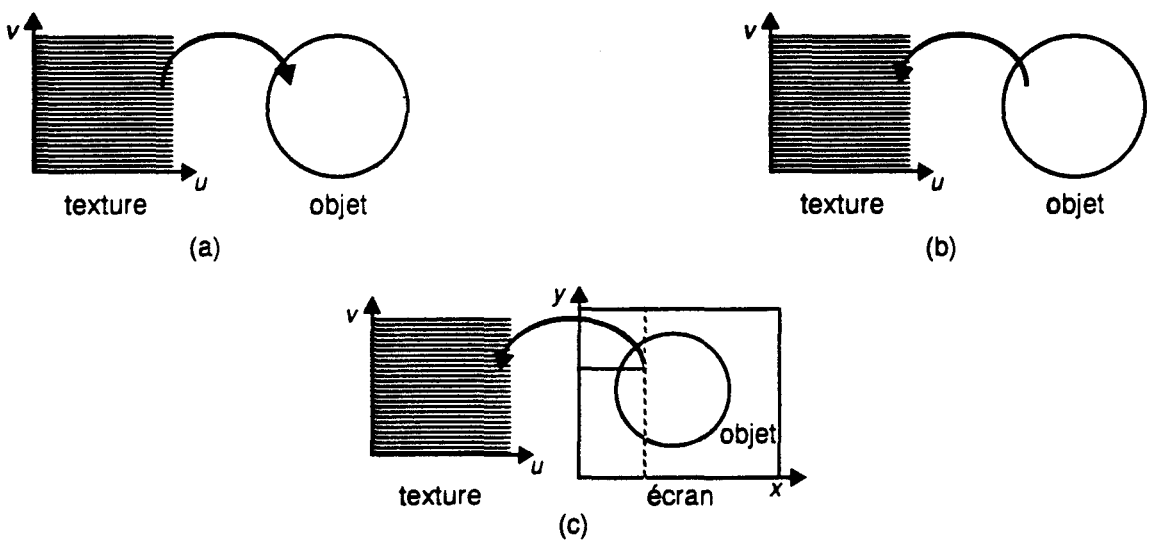


Figure 3.16 : Trois approches pour le placage de textures

- Méthodes générales

La première méthode que nous proposons d'étudier consiste à définir en certains points de l'objet à afficher, un repère local correspondant à un repère particulier dans la table de texture [Gaga91].

Gagalowicz propose de définir ce repère (Local Coordinate System) à l'aide de la normale $\vec{N}$ et de deux autres vecteurs $\vec{\Delta x}$ et $\vec{\Delta y}$ dans le plan tangent à la surface, aux points choisis P_c. En chaque point P de la surface, on détermine le point P_c le plus proche et on calcule les coordonnées (x_{pc}, y_{pc}, z_{pc}) de P dans le L.C.S. associé à P_c. Les coordonnées (x_{pc}, y_{pc}) permettent de rechercher la couleur dans la table des textures, à partir du repère local associée au point P_c. Le problème est alors de trouver les deux vecteurs dans le plan tangent, afin d'éviter toute déformation anormale de la texture. Gagalowicz utilise pour cela les géodésiques et leurs tangentes $\vec{T}$, qui fournissent la distance et la trajectoire de déplacement des L.C.S. les unes par rapport aux autres (Figure 3.17).

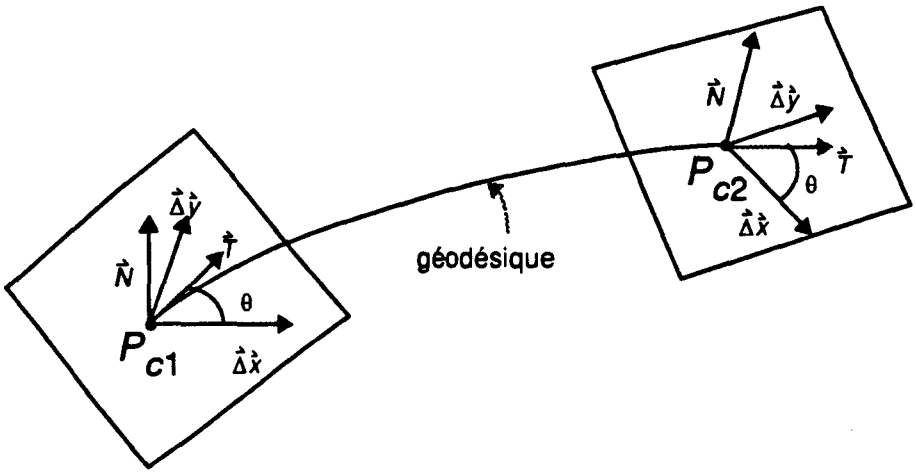


Figure 3.17 : Déplacement d'un L.C.S. d'un point à un autre

Le placage des textures s'effectue en choisissant un point de départ, puis en calculant toutes les géodésiques joignant ce point aux autres points.

Les L.C.S. sont donc une solution au placage de textures, si l'on sait déterminer pour chaque surface les géodésiques nécessaires.

Le calcul des géodésiques : une géodésique est définie comme la courbe de la surface décrivant le plus court chemin entre deux points P_1 et P_2 de cette surface. Analytiquement, il s'agit de déterminer les fonctions $X(t)$, $Y(t)$ et $Z(t)$ telles que

$$S = \int_{t0}^{t1} \sqrt{X'^2 + Y'^2 + Z'^2}\, dt$$

soit minimale avec

$P_1(X(t0), Y(t0), Z(t0))$, $P_2(X(t1), Y(t1), Z(t1))$ et $X' = \frac{d}{dt}X(t)$, $Y' = \frac{d}{dt}Y(t)$, $Z' = \frac{d}{dt}Z(t)$.

Deux méthodes analytiques peuvent être employées :

- La méthode traditionnellement employée est le calcul des géodésiques à partir des expressions paramétriques de la surface $X(u, v)$, $Y(u, v)$ et $Z(u, v)$, en définissant $u = u(t)$ et $v = v(t)$.

Le calcul de S devient :

$$\int_{t0}^{t1} F(u, v, u', v')\, dt$$

et la condition de minimisation est obtenue par les équations différentielles suivantes :

$$\frac{dF}{dt} - \frac{d}{dt}\left(\frac{dF}{du'}\right) = 0 \text{ et } \frac{dF}{dt} - \frac{d}{dt}\left(\frac{dF}{dv'}\right) = 0.$$

- La seconde méthode s'emploie pour les surfaces connues par leurs équations implicites $Q(x, y, z) = 0$. La fonction à minimiser est alors :

$$S' = \int_{t0}^{t1} F\, dt$$

avec $F = \sqrt{X'^2 + Y'^2 + Z'^2} - Q(x, y, z)$.

Nous obtenons alors trois équations différentielles qui déterminent la géodésique :

$$\frac{dF}{dt} - \frac{d}{dt}\left(\frac{dF}{dX}\right) = 0, \frac{dF}{dt} - \frac{d}{dt}\left(\frac{dF}{dY}\right) = 0 \text{ et } \frac{dF}{dt} - \frac{d}{dt}\left(\frac{dF}{dZ}\right) = 0.$$

Une méthode de calcul des géodésiques par approximation [Ma86] peut être employée si les méthodes analytiques ne conviennent pas. Elle consiste à construire un graphe à partir des points choisis sur la surface et à définir pour chaque point, le chemin minimal le séparant du point de départ. Les géodésiques étant approchées par une suite de segments, sont donc discrétisées.

Parmi les autres choix possible pour définir les L.C.S., la méthode utilisant une découpe de l'objet en tranche offre l'intérêt d'être assez simple, si l'on connaît la normale en chaque point. Le L.C.S est défini par la normale à la surface $\vec{N}$, le vecteur $\vec{U}$ tangent à la courbe d'intersection de la surface et du plan de la tranche et par leur produit vectoriel $\vec{V} = \vec{N} \wedge \vec{U}$.

Gagalowicz précise que lorsque la déformation n'a pas d'importance aux yeux du concepteur, il est possible d'utiliser des L.C.S. définis à partir des expressions paramétriques de la surface. Les L.C.S. obtenues sont définies par :
$\vec{\Delta x}\left(\frac{dX}{du}, \frac{dY}{du}, \frac{dZ}{du}\right)$, $\vec{\Delta y}\left(\frac{dX}{dv}, \frac{dY}{dv}, \frac{dZ}{dv}\right)$ et $t\vec{N} = \vec{\Delta x} \wedge \vec{\Delta y}$
Quel que soit la définition employée pour les L.C.S., le principe de cette méthode de placage est une "facettisation" de l'objet pour l'application de la texture.

La seconde méthode générale de placage des textures, appelée méthode à deux passes, est décrit dans [Bier86]. Elle consiste à utiliser une surface intermédiaire sur laquelle la texture est pré-plaquée, et ensuite à effectuer le placage à partir de cette surface. La couleur en un point P de la surface affichée est déterminée à partir de la couleur en P', intersection entre la surface intermédiaire et une droite passant par P.

Suivant le cas, quatre surfaces intermédiaires peuvent être utilisées principalement car elles sont facilement texturables :

- Le plan qui est bien sûr la surface de la texture.
- Le cube est topologiquement équivalent à la sphère, et possède alors la propriété de pouvoir englober les objets.
- Le cylindre puisque celui-ci est une surface développable.
- La sphère qui a une courbure constante en tout point.

Le choix de surface intermédiaire influe beaucoup sur les déformations de texture obtenues.

Une surface intermédiaire étant choisie, plusieurs algorithmes peuvent être employés pour calculer la couleur d'un point P.

- Il est possible de calculer l'intersection entre la surface intermédiaire et une droite passant par P et un autre point fixe G arbitrairement choisi pour l'objet (en général G est intérieur à l'objet) (Figure 3.18.a).
- Si l'on connaît la normale $\vec{N}$ au point P, le calcul d'intersection s'effectue entre la surface intermédiaire et la droite passant par P, dont un vecteur directeur est $\vec{N}$ (Figure 3.18.b).

Le troisième algorithme emploie la surface intermédiaire pour la construction de la droite. Le point P' cherché est défini par $\vec{PP'} \wedge \vec{N}_{P'} = 0$, avec $\vec{N}_{P'}$ la normale au point P' (Figure 3.18.c). La(les) droite(s) doit alors passer par P, ce qui implique dans certain cas, qu'il n'y a pas de solution unique P' pour déterminer la couleur de P.

Les études montrent que le minimum de déformation de la texture pour toute surface intermédiaire est obtenu par le troisième algorithme [Bier86].

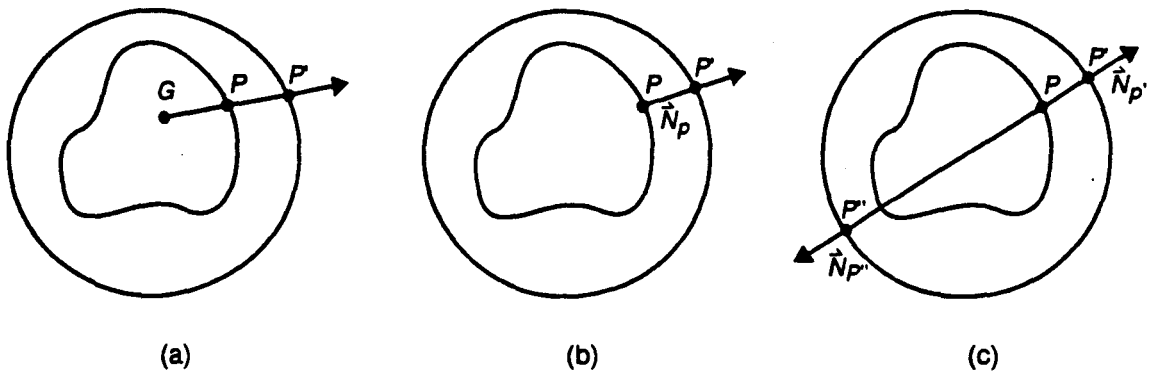


Figure 3.18 : Le cylindre comme surface intermédiaire (représentation bidimensionnelle)

• Applications des textures bidimensionnelles aux facettes :

La facettisation des objets permet l'emploi des différentes méthodes de placage des textures, puisque les coordonnées "exactes" dans la table des textures sont seulement calculées aux sommets des facettes. Lors du rendu, les coordonnées (u, v) de la couleur des autres points des facettes sont obtenues par interpolation bilinéaire (voire trilinéaire dans certain cas) des valeurs aux sommets (u_s, v_s). Les facettes sont petites afin de réduire les déformations engendrées par la méthode d'interpolation employée. Actuellement les objets sont facettisés, les valeurs u et v en chaque sommet sont définies pendant le phase de modélisation et interpolées lors de l'affichage pour obtenir des valeurs u_i et v_i en chaque point de la surface. Nous pouvons remarquer cependant, qu'il est nécessaire d'utiliser des petites facettes afin de minimiser les erreurs dues aux fonctions d'interpolation employées.

• Etude des méthodes existantes de textures bidimensionnelles avec les quadriques:

Si l'on désire utiliser les quadriques comme primitives d'affichage scan-line, il est indispensable de connaître la couleur de chaque point, à partir de ses coordonnées. Il est alors impossible d'employer la méthode par balayage de l'espace des paramètres. Le problème posé par l'inversion des fonctions paramétriques pourrait alors être résolu en utilisant la méthode à deux niveaux. Bien que cette méthode soit attrayante lors d'un affichage en scan-line, deux problèmes se posent dans le cas des quadriques: les fonctions $G_u(X)$ et $G_v(Y)$ employées sont complexes et ont une forme dépendant du type de la quadrique à visualiser. Il est donc très difficile, voire impossible, de les implanter matériellement.

L'approche paramétrique pour plaquer les textures semble donc inadaptée à l'utilisation des quadriques comme primitive d'affichage.

L'application de la méthode de Gagalowicz à la primitive quadrique soulève bien des problèmes difficiles à résoudre. En effet, le calcul d'une géodésique sur une sphère ou un cylindre peut être défini de manière assez simple, mais il n'en va pas de même pour les autres type de quadrique. De plus, les géodésiques sont définies par des courbes de plus haut degré que les coniques, ce qui pose le problème de la réalisation matérielle de leurs évaluations. L'emploi des géodésiques discrètes permet d'éviter ces problèmes, mais le choix du plan tangent à utiliser en chaque pixel reste difficile.

La méthode à deux passes est générale puisqu'elle ne tient pas compte de la primitive employée à

l'affichage, elle semble donc mieux adaptée à la quadrique et permet même l'emploi de primitives d'affichage différentes au sein de la même machine de rendu.Bien que des problèmes de déformation de la texture existent, le choix de la surface intermédiaire et la possibilité de pré-déformer la texture reste sous le contrôle de l'utilisateur, afin qu'il puisse définir l'objet le plus librement possible. Les calculs des coordonnées u,v restent relativement simple si l'on emploie le troisième algorithme, puisqu'ils font intervenir la normale à la surface intermédiaire (donc la normale à un plan, un cylindre ou une sphère), puis une intersection entre une droite et la primitive employée, qui dans notre cas est une quadrique.

Le placage de texture sur une machine d'affichage employant la quadrique comme primitive ne peut donc apparemment pas employer les méthodes géométriques utilisant la forme paramétrique des surfaces. Parmi les deux méthodes générales présentées, seule la méthode à deux passes nous semble envisageable, la méthode de Gagalowicz étant trop complexe pour une implantation matérielle.

Avant d'envisager une implantation matérielle, deux problèmes restent à étudier :

- L'emploi d'une transformation perspective lors de l'affichage ne permet pas d'obtenir directement les points de l'objet après la conversion : l'application d'une des méthodes risque alors d'engendrer des calculs plus complexes que ceux décrit.

- Lors de l'application des textures bidimensionnelles, il faut effectuer un anti-aliassage, puisque les coordonnées u et v ne font pas parties de la grille d'échantillonnage utilisée pour mémoriser la table des textures : le choix de la couleur du point le plus proche de cette grille provoque en général un phénomène d'aliassage important, qui est accentué lors d'une animation (effets de vague sur les couleurs).

Textures tridimensionnelles

Le but des textures tridimensionnelles est de remplacer le placage de textures bidimensionnelles, qui est en somme une "mise en peinture", par une définition de l'objet dans la masse d'un matériaux: l'objet est "taillé" dans la matière désirée, comme on peut sculpter une figure dans un bloc de bois ou de granit, dans le monde réel.

La méthode consiste à attribuer à chaque point de l'objet une valeur ne dépendant que des coordonnées (x,y,z) et pouvant servir, comme dans le cas des textures bidimensionnelles, à définir une couleur, à perturber la normale au point en question ou à modifier une des caractéristiques photométriques de l'objet. Dans la suite de ce paragraphe, nous désignerons cette valeur par le terme couleur.

Une première remarque peut être formulée: la mise en œuvre des textures tridimensionnelles est indépendante de la géométrie de l'objet affiché, ce qui n'est pas le cas des méthodes de placage de texture. Les textures tridimensionnelles sont donc adaptées au rendu de primitives différentes, au sein d'une même machine d'affichage.

La seconde remarque est que naturellement le calcul de la couleur d'un objet par une technique de textures tridimensionnelles sur une machine d'affichage, peut être effectué après élimination des parties cachées, et donc une fois par pixel (mis à part le cas où les objets sont transparents).

La base des textures tridimensionnelles est la définition d'un maillage de l'espace (en général, l'échantillonnage est unitaire). Pour que la texture soit attachée à l'objet que l'on cherche à représenter, il faut que la définition du maillage s'effectue dans un espace associé à l'objet, afin de pouvoir appliquer les mêmes mouvements à la texture et à l'objet. Une valeur V en un point (x,y,z) est définie, par interpolation, à l'aide de valeurs associées aux huit nœuds du maillage les plus proches. Les valeurs V associées aux nœuds du maillage sont fixées une fois pour toute lors de la définition de la texture.

Sans nous attarder sur la synthèse de la texture, il est intéressant de voir comment sont générées les

textures tridimensionnelles [Lewi89]. Une couleur peut être définie à l'aide de deux approches:

- On définit en chaque nœud (x,y,z) la couleur par une fonction de génération de texture. Par exemple, la méthode de Perlin [Perl85], définit une fonction spécifique de couleur mat(x,y,z) pour chaque matière utilisant un bruit blanc noise(x,y,z).

- Un autre usage des valeurs V est non pas un calcul de couleur aux points par une fonction, mais une application d'un déplacement des points et donc de la couleur associée à ces points [Peac85].

Le calcul des valeurs V par une fonction de bruit blanc, permet de définir des matières comme le marbre ou le quartz, qui sont représentées au départ par un ensemble de plans parallèles (ou de veines) ayant des couleurs (au sens premier du terme) différentes et qui subissent des déformations dues à certaines pressions.

Il est de même possible de définir du bois, en considérant au départ un ensemble de cercles concentriques, puis en appliquant les déplacements. Cette approche est utilisée pour définir des pré-textures, les points à afficher n'étant pas modifiables : la couleur en un point (x,y,z) est alors déterminée par interpolation des couleurs aux huit nœuds les plus proches.

Deux types d'applications des textures tridimensionnelles sont alors possibles :

- Dans le cas des pré-textures, la couleur de chaque nœud du maillage est calculée.

- Dans le cas des post-textures, les coordonnées (x,y) des pixels de l'objet, auxquels on ajoute les profondeurs z calculées en ces points, servent de données pour la fonction employée. Il est donc nécessaire pour les post-textures de connaître les méthodes employées, afin d'étudier la possibilité de les implanter matériellement.

En reprenant l'exemple des textures de Perlin, la première phase est de calculer en chaque nœud du maillage, un gradient pseudo-aléatoire $\Gamma_{x,y,z}$ (accessible dans une table, à l'aide d'une fonction de hachage). Vient ensuite le calcul du bruit $Noise(x,y,z)$ en un point $P(x,y,z)$:

$$Noise(x,y,z) = \sum_{i=\lfloor x \rfloor}^{\lfloor x \rfloor+1} \sum_{j=\lfloor y \rfloor}^{\lfloor y \rfloor+1} \sum_{k=\lfloor z \rfloor}^{\lfloor z \rfloor+1} \Omega_{i,j,k}(x-i, y-j, z-k)$$

où, $\lfloor\ \rfloor$ est la fonction de valeur entière et où

$$\Omega_{i,j,k}(u,v,k) = \omega(u) \times \omega(v) \times \omega(k) \times (\Gamma_{x,y,z} \bullet (u,v,k))$$

avec $w(t) = 2|t|^3 - 3|t|^2 + 1$ pour $|t| \leq 1$, la fonction de pondération.

La fonction bruit $Noise(x,y,z)$ a les propriétés suivantes :

- Statistiquement invariante par rotation et par translation
- Posséder une bande passante limitée.

La valeur de bruit obtenu en P peut alors servir à définir la couleur (au sens premier du terme) ou à perturber la normale.

Exemple : couleur = blanc * Noise(x,y,z)

D'autres fonctions dérivées de cette notion de bruit peuvent être employée, comme la turbulence que Perlin définit en n'utilisant que les octaves de fréquence du bruit initial.

Suivant la matière désirée, il faut alors composer la fonction de bruit (ou une des fonctions dérivées) avec une autre fonction afin d'obtenir la couleur du point.

Dans le cadre de la définition d'une machine d'affichage en temps réel, il faut alors considérer les deux possibilités :

- L'exemple de la méthode de Perlin montre que dans le cas des post-textures, il est nécessaire de composer la valeur de bruit obtenu en chaque point, avec une fonction dépendant de la texture désirée. Cette contrainte imposerait lors d'une implantation matérielle, de prévoir différents types de calculs au sein d'une même unité et donc de limiter le nombre de textures différentes utilisées. En outre, les fonctions permettant de calculer les couleurs des points à partir du bruit [Perl89](fonctions trigonométriques, splines, ...), sont généralement incompatible avec le temps réel. L'approche à base de post-textures n'est donc pas une voie envisageable dans le cadre des machines d'affichage en temps réel.

- La définition de pré-textures sur les objets semble par contre une solution aux problèmes d'augmentation de réalisme dans les scènes visualisées en rendu géométrique, puisque seule une interpolation est nécessaire, quelque soit la texture. Le problème cependant avec cette approche est la place mémoire occupée par la table des textures et surtout son accès, si le rendu utilise un gros grain de parallélisme (exemple : découpage de l'écran et répartition sur plusieurs modules d'affichage).

Un problème auquel il est nécessaire de songer avant d'envisager l'implantation matérielle et que nous n'avons pas encore étudié, est le coût du changement de repère permettant de placer les points dans le repère de la texture à appliquer.

Conclusion

L'augmentation du réalisme des images, sur une machine de rendu géométrique, à l'aide de textures pose donc le problème de la complexité des calculs et de la généralité des algorithmes employés.

La complexité des calculs est bien entendu liée à la définition des coordonnées des points affichés (pixels plus valeur de profondeur) dans le repère de la texture, mais le plus important est la position de la méthode dans le processus de visualisation. Ainsi les méthodes de placage de texture sont étroitement liées aux objets convertis et leur application doit être effectuée lors de la phase de conversion des objets en pixel (donc une fois par objet et par pixel). Dans le cas des textures tridimensionnelles, le problème de complexité est moins contraignant puisque les méthodes peuvent bénéficier du parallélisme pixel, mais les problèmes sont alors l'accès et la taille mémoire utilisée pour le stockage.

Si la machine emploie des primitives de conversion différentes, seule les textures tridimensionnelles semblent applicables puisqu'elles permettent de généraliser aux primitives différentes, tout en employant les mêmes calculs lors de la phase de rendu. Les méthodes de textures bidimensionnelles utilisent en effet la géométrie de l'objet à convertir, ce qui augmenterait la complexité matérielle d'une machine les employant.

3.3 Perspectives

L'étude de la primitive d'affichage quadrique, en rendu géométrique, est relativement récente. De ce fait peu d'algorithmes efficaces ont été développés dans ce cadre, alors que de nombreux travaux ont été effectués spécifiquement pour la primitive facette.

La modélisation à base de quadriques volumiques ou surfaciques semble être un domaine encore peu abordé. De même l'approximation des surfaces employées en modélisation (B-Splines, patchs de Béziers, ...) n'a été étudiée pour l'instant que dans le cadre d'un rendu en facettes. Il est donc indispensable de s'intéresser à ces deux approches de la modélisation, afin de pouvoir employer la quadrique dans une machine de rendu géométrique et afin de choisir le processeur qui sera utilisé par les machine de rendu.

Ce chapitre montre aussi que certaines techniques employées avec la primitive facette peuvent être utilisées avec la quadrique. Nous avons ainsi défini une méthode simple permettant l'application des ombres portées.

D'autres techniques d'amélioration de la qualité des images (anti-aliassage, applications de textures) restent encore à explorer, afin d'être utilisées avec la primitive quadrique. Il est vrai que les algorithmes à employer avec la quadrique, doivent être suffisamment généraux afin d'être implantés matériellement, mais leur utilisation permet une qualité du rendu supérieure quelque soit la primitive employée en conversion.

De nombreuses questions sont donc posées par le changement de la primitive d'affichage. La définition d'objets complexes à l'aide de quadriques, est même très importante puisqu'elle conditionnent l'utilisation de la primitive quadrique lors du rendu. Les autres questions restant en suspens sont moins importantes, mais il faut y répondre pour que le rendu à base de quadrique devienne une réalité.

Chapitre 4 :
Simulations et Animations

Dans le but de valider les concepts utilisés pour la définition des processeurs objets quadriques, nous avons simulé le rendu d'objets avec un post-éclairement qui utilise le pipeline de Phong. Nous allons tout d'abord présenter l'environnement de la simulation et les fonctionnalités du programme, afin d'expliquer les organisations que nous avons employées et les choix que nous avons fait.

Un programme d'animation a été réalisé afin de tester les P.O.Q. simulés, lors de la conversion de différentes scènes; nous décrirons donc les structures utilisées afin de manipuler les objets, ainsi que les fonctionnalités du programme. Nous finirons par une description du module permettant de générer les volumes d'ombres portées des primitives utilisées.

4.1 La simulation du rendu

Le programme de rendu avait pour base la simulation de la machine IMOGENE sur une machine parallèle à base de Transputers T800 (le Multicluster II) équipée d'une carte vidéo GDS II. Par la suite le simulateur a été modifié afin d'augmenter la vitesse d'affichage des images générées. Le Multicluster II a été choisi pour sa puissance calcul et pour les possibilités de parallélisation qu'il offrait. En effet, la simulation d'une machine complète amène un flot énorme de calculs et une utilisation du parallélisme a été nécessaire.

Comme nous l'avons vu aux chapitres précédents, nous avons utilisé cette simulation afin de valider les concepts employés par les Processeurs Objets et comparer les données théoriques avec des données réelles. Les résultats que nous avons obtenu, nous ont permis de conclure sur l'utilisation des processeurs de conversion de quadriques dans les machines de rendu.

4.1.1 L'environnement de simulation

Nous présentons ici le Multicluster II, machine à base de Transputers, dont le principe est de pouvoir définir une machine multi-processeurs, avec un débit de communication inter-processeurs élevé. Nous parlons de la GDS II, du langage OCCAM et des différentes configurations possibles des Transputers.

Les T800

Le Transputer T800 [Inmo89] est un processeur 32 bits intégrant une unité de calculs flottants sur 64 bits. Son architecture comprend aussi une mémoire Ram interne de 4 Koctets, une interface mémoire configurable et quatre liens de communications point à point bidirectionnels (les liens sont en fait constitués de deux liens monodirectionnels solidaires).

Différents processus peuvent être exécutés en partage de temps sur un même Transputer. Deux processus voulant communiquer utilisent, soit une liaison point à point s'ils sont répartis sur deux

Transputers (communication synchrone sans mémorisation), soit un canal logique s'ils sont sur le même processeur (mémorisation d'un mot).

L'utilisation de Transputers en réseau nécessite la gestion d'une mémoire partagée, puisque chaque processeur possède son propre espace de travail.

Enfin, les Transputers ont été spécialement étudiés pour une programmation en langage OCCAM, mais supportent aussi d'autres langages comme le C ou le Pascal.

La GDS II

La GDS II est une carte graphique contenant un Transputer T800, un contrôleur vidéo G300, 2 Mo de RAM dynamique et 2Mo de RAM vidéo doubles accès.

Le T800 utilisé dispose de ses quatre liens pour être connecté à d'autres Transputers. Le Transputer de la carte peut calculer directement les données des pixels ou recevoir les informations graphiques à travers l'un de ses liens avant de les envoyer dans la mémoire vidéo.

Le contrôleur vidéo, initialisé à partir du Transputer de la carte, permet de supporter différents modes de résolution allant de 1280×1024 pixels en 256 couleurs à 800×600 pixels en 16 millions de couleurs.

Le Multicluster II

Le Multicluster II de Parsytec est une machine à base de Transputers, intégrant au maximum 64 processeurs (32 noeuds). Le système de communications utilise deux cross-bars statiques reconfigurables, un pour les liens pairs et un pour les liens impairs. La configuration utilisée pour l'implantation du simulateur, comprend 32 Transputers avec chacun une mémoire externe de 2 Moctets, une carte graphique GDS II et 8 noeuds d'entrée (4 transputers avec 4 Moctets de mémoire et 4 autres Transputers n'en ayant que 2 Moctets de mémoire).

Les points d'entrée dans le système sont des Transputers (noeuds d'entrée) dont un lien est connecté aux cross-bars par l'intermédiaire d'une unité de gestion des entrées/sorties. Il est donc possible que plusieurs utilisateurs emploient le système simultanément, en définissant le nombres de Transputers nécessaires et en configurant les deux cross-bars pour leurs applications respectives : cette approche permet donc la gestion optimale des ressources du système. Le seul problème du système est le lien unique de connexion des noeuds d'entrée avec le reste du système, qui restreint les possibilités de communications du système vers l'extérieur.

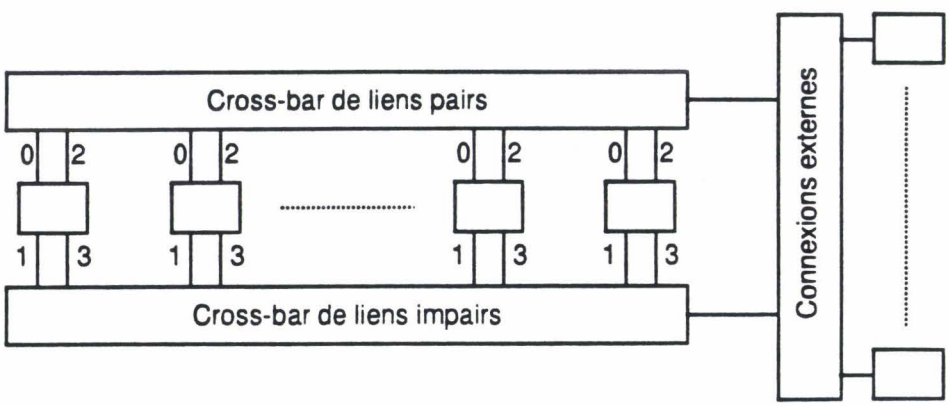


Figure 4.1 : Connexions sur le Multicluster II

Les Transputers peuvent être gérés par deux environnement différents : Multitool et Helios.

- Multitool (Mtool) est l'environnement de programmation en langage OCCAM. Les outils disponibles sont le compilateur OCCAM et un utilitaire de configuration des cross-bars du système et de chargement des transputers. Lorsque les différents processus d'un programme sont placés sur le système, la première démarche est de déterminer la topologie des liens de communications utilisée afin de fixer la configuration des cross-bars. Le chargement des processus peut ensuite être effectué par transfert du code exécutable de chacun d'eux sur les Transputers qui leurs sont alloués. Le langage OCCAM étant très proche de l'architecture, la gestion des communications inter-processus (et donc inter-Transputers) est laissée aux soins de l'utilisateur.

- Helios comprend à la fois un environnement de programmation en C et un système d'exploitation permettant de gérer les communications inter-processus. De ce fait, la conception des programmes devant être exécutés sur la machine multi-transputers est simplifiée, puisque l'utilisateur n'a plus a prendre en compte l'organisation des communications entre les différents processus. L'environnement d'Helios possède un système de gestion des communications réparti, implanté sous la forme d'un noyau (processus) par Transputer.

- Dans le cadre du projet IMOGENE, nous avons choisi d'utiliser l'environnement Mtool et le langage de programmation OCCAM, puisque le sujet initial était la simulation d'une architecture et que des tests devaient aussi être effectués sur l'organisation de la machine IMOGENE. Il était donc indispensable de pouvoir gérer les liens de communications.

Une seconde raison, moins importante, a également été émise: les données inter-Transputers sont pour la plupart attachées aux pixels à afficher. Le nombre de pixels traités étant important, il fallait que les communications entre processeurs soit effectuées le plus rapidement possible. Comme l'utilisation d'un processus de communication par Transputer ralentit les transferts de données dans le système, nous avons préféré Mtool et le langage OCCAM à Helios et au langage C.

Le langage OCCAM

Le langage OCCAM utilise deux concepts pour la programmation des Transputers :

- Les processus, qui sont des parties de code exécutables soit en partage de temps sur un Transputer, soit sur des Transputers différents.

- Les communications, qui sont utilisées pour effectuer le transfert de données entre deux processus.

• Les processus

Les blocs d'instructions séquentielles définis après l'instruction PAR sont considérés comme étant des processus qui s'exécutent en parallèle. Tous les processus ont la même priorité d'exécution, sauf lorsque le programmeur fait précéder la commande PAR du mot clé PRI : le premier processus défini alors est effectué en priorité haute, ce qui signifie qu'il est exécuté beaucoup plus vite que les autres processus (l'horloge associée à un processus prioritaire est 64 fois plus rapide que celle associée à un processus ordinaire).

Afin de gérer le parallélisme dans le réseau de Transputers, le langage OCCAM contient l'instruction de placement de processus sur les processeurs (PLACED PAR). Les utilitaires de configuration du système, de placement des tâches sur les Transputers et de chargement du code exécutable sont alors capables d'effectuer leurs missions.

• Les communications

Comme nous l'avons indiqué, le langage OCCAM a été choisi car il offre la possibilité de gérer

directement les liaisons inter-processus.

Ces liaisons sont définies à l'aide de canaux de communications reliés les uns aux autres. Une liaison inter-processus, qu'elle soit logique (communications entre deux processus sur un même transputer) ou physique (communications inter-Transputers), est définie lorsque deux canaux de communication sont reliés. Une liaison physique est détectée lorsque les deux canaux connectés sont placés sur les liens physiques de deux Transputers différents. Le langage impose donc de la connexion de deux canaux, que l'un soit utilisé toujours en émission et l'autre toujours en réception, afin de définir des communications monodirectionelles.

La transmission de données est un mécanisme synchrone sur les Transputers, l'émission et la réception d'une valeur sont des actions bloquantes. La langage OCCAM offre toutefois la possibilité de traiter les actions de réceptions comme des événements déclenchant l'exécution d'un bloc d'instructions, grâce à l'instruction ALT, ce qui permet d'outre-passer les blocages dûs à la synchronisation des communications.

Les liaisons étant créés implicitement lors de la connexion de deux canaux, certaines configurations ne sont pas réalisables sur le Multicluster II, puisqu'il emploie deux cross-bars séparés, empêchant la connexion des liens pairs d'un Transputer avec les liens impairs d'un autre.

Différentes topologie possibles des Transputers

Le MultiCluster II étant une machine multi-Transputers, ceux-ci peuvent être organisés de différentes manières, mais il faut toutefois prendre en compte le nombre de liens de communications disponibles sur un processeur. Voici un aperçu des solutions généralement employées pour relier les Transputers les uns aux autres (Figure 4.2).

- L'arbre ternaire est défini par une structure arborescente dont chaque noeud a trois fils. Cette représentation est possible sur un réseaux de transputers, puisque chaque processeurs a quatre liens de communications disponibles.

L'avantage principal de cette structure est sa simplicité d'organisation, qui permet la connexion et la gestion d'un grand nombre de Transputers. Plusieurs points d'entrée sont possibles suivant que l'on effectue plutôt de la diffusion de données (le point d'entrée est le lien libre du transputer à la racine) ou de la concentration (les points d'entrées sont les liens libres des feuilles de la structure).

Les problèmes posés par cette organisation peuvent cependant être importants :

- Un seul chemin de communication possible entre deux transputers et donc une possibilité de conflit insurmontable (Dead Lock).
- Les Transputers des nœuds intermédiaires de l'arbre ne sont accessibles que par le point d'entrée (ou les points d'entrée) choisi : le chemin peut être difficile à déterminer.

- L'hypercube dont la structure est définie par deux cubes dans le cas des Transputers, qui n'ont que quatre liens de communication. Les liaisons sont les arêtes des cubes et les deux cubes communiquent par leurs sommets. Cette structure est intéressante lorsque l'on peut créer des groupes de tâches qui communiquent entre elles. Cette configuration permet aussi la gestion des conflits et donc le fonctionnement du système en mode dégradé. Le défaut de cette structure est le nombre fixe de Transputers, qui est de seize (pas de possibilité d'extension) et la structure fermée, qui pose le problème du point d'entrée dans le système.

- Le Tore présente deux qualités, qui sont la simplicité de gestion de l'organisation et la possibilité de communications entre deux Transputers par différents chemin. Cependant comme pour l'hypercube, le problème de l'entrée se pose puisque la structure est fermée.

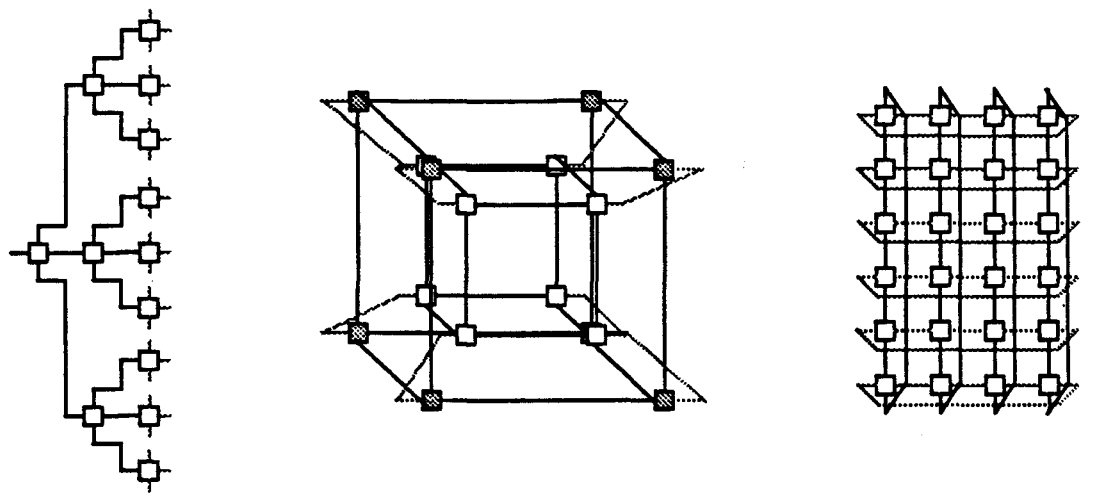


Figure 4.2 : Différentes organisations de Transputers

Les points critiques, dont il faut tenir compte lors de la définition topologique employée par le programme, sont:

- L'inactivité de certains Transputers lors de certaines exécutions, alors que d'autres sont surchargés.
- Les envois de données, qui sont des actions bloquantes et qui peuvent pénaliser certaines tâches.
- Les distances de communications, qui peuvent être longues.

Afin de bien définir la structure de Transputers à utiliser, nous devons alors définir les principales fonctionnalités du programme de rendu.

4.1.2 Les Processus du rendu

Pour la simulation, les quatre étapes du pipeline de rendu de Phong sont séparées, afin de définir les rôles des Transputers dans le schéma de l'organisation.

Nous distinguons donc quatre fonctionnalités, que nous avons représentées par quatre processus généraux:

- la préparation des objets (changement de repère et perspective)
- la conversion des objets en pixels
- l'élimination des parties cachées par un Z-buffer
- le post-éclairement des pixels

Notre objectif étant de simuler une machine de rendu, la partie préparation des objets est séparée de la partie rendu : les trois premières fonctionnalités sont implantées sur le réseau et la préparation des coefficients est simulée sur le nœud d'entrée.

La conversion étant, dans notre étude, la partie la plus importante de la simulation, nous commençons par décrire les différents Processus de conversion d'Objets primaires. Nous présentons ensuite le Processus Général de conversion des Objets, qui utilise les P.O. et les autres processus permettant le rendu d'une scène.

Les processus de la conversion

Dans un premier temps, nous avons défini des processus simulant les Processeurs Elémentaires calculant des expressions du premier et du second degré et les Extracteurs de Racines Carrées. Ensuite nous avons défini le processus de conversion des quadriques, qui est la simulation du processeur quadrique version 2. Nous avons ajouté dans la simulation d'autres processus de conversion (facettes, polyèdres, volumes d'ombre), afin de pouvoir visualiser des scènes complètes et tester le rendu visuel des images générées au moyen de primitives et d'algorithmes différents.

- Les Processus Elémentaires.

Afin de simuler les calculs des fonctions du premier et du second degré a un niveau très bas, les processus emploient les techniques incrémentales de calculs, qui permettent de définir une valeur à partir de la précédente. Les changements de pixels ou de lignes sont indiqués au moyen d'un signal de contrôle extérieur défini comme un événement. La gestion du chargement des coefficients et l'envoi des valeurs en chaque pixel, sont eux aussi effectués par réception d'un signal de contrôle extérieur.

Un module de conversion utilisant des processus P.E., doit alors au début de chaque nouvelle image indiquer la réinitialisation et fournir les coefficients des expressions employées. L'envoi d'un signal lors de chaque changement de ligne, puis lors d'un changement de pixel, permet le calcul d'une nouvelle valeur, qui est transmise en chaque pixel au module. L'ensemble des données et commandes entre un module et un processus P.E. transite par deux canaux. Nous avons appelé P.E.1 les Processus Elémentaires calculant les expressions du premier degré et P.E.2 ceux définissant les expressions du second degré, par similitude avec les Processeurs Elémentaires premier et second degré.

Toujours dans le souci de rester au plus près des schémas de réalisation, les calculs sont effectués sur 24 bits pour les expressions du premier degré et sur 48 bits pour les expressions du second degré, grâce à l'emploi de masques sur les valeurs.

- Les Processus de conversion des Objets primaires.

L'ensemble des Processus Objets employant la conversion en scan-line, les changements de lignes et le passage d'un pixel au suivant sont simulés à l'aide de deux boucles imbriquées. Les Processus Elémentaires sont initialisés au début du traitement d'un objet. Un Processus Objet envoie à chaque changement de ligne et à chaque changement de pixel, le signal correspondant aux P.E. qui lui sont associés, puis il reçoit de ceux-ci en chaque pixel, les valeurs des expressions qui lui sont nécessaires pour effectuer la conversion.

- Le premier Processus Objet défini est la simulation du Processeur Objet Quadrique de la première génération : l'organisation de la conversion suit donc le schéma fonctionnel que nous avons défini. Le processus emploie huit P.E.1 et six P.E.2.

Afin de tester les différentes méthodes utilisées pour le calcul d'une racine carrée, nous avons défini plusieurs procédures. La première d'entre elles fut l'application de la méthode d'approximation linéaire suivie de deux itérations de correction. La seconde procédure emploie le calcul de la racine carrée par approximation quadratique avec une itération de correction. Enfin, la dernière procédure définit le calcul d'une racine carrée exacte.

Les valeurs en un pixel sont traitées par l'algorithme de définition du contour (spécifique aux quadriques coupées par deux plans parallèles). Les deux valeurs de profondeur avant et arrière, ainsi que la normale, sont ensuite transmises au processus de Z-buffer.

- Le Processus Objet Facette est le second que nous avons défini pour comparer la primitive quadrique

et la primitive facette. Ce processus emploie la méthode de conversion des facettes par équations pour un calcul d'éclairement par la méthode de Phong, puisque le rendu emploie un post-éclairage, qui utilise les valeurs calculées par 7 Processus Elémentaires premier degré.

Après traitement, les valeurs envoyées sont deux valeurs de profondeur et une normale. Les deux valeurs de profondeur sont en fait, deux fois la même valeur, mais cette définition des valeurs permet de rester compatible avec les résultats envoyés par le Processus Objet Quadrique et attendus par le Processus Z-buffer.

- Après la définition de la seconde génération de Processeur Objet Quadrique, nous avons simulé le schéma de fonctionnement de ce nouveau P.O., au moyen d'un nouveau Processus. Les valeurs nécessaires à la conversion sont fournies par sept P.E.1 et quatre P.E.2. Pour simuler le module de plan de coupe défini dans la seconde génération de Processeur Objet, nous avons codé l'algorithme correspondant dans une procédure. Le Processus simule la conversion d'une quadrique coupée par trois plans. Cette définition nous a permis de vérifier la validé globale de l'algorithme lors de l'application de plusieurs plans de coupe sur une quadrique. Les coupes par les différents plans sont effectuées successivement, afin de suivre le schéma de fonctionnement défini.

L'étude de l'Extracteur de Racines Carrées ayant été effectué avant la simulation du P.O.Q. seconde génération, la méthode de calcul de racines carrées employée dans le processus est la méthode exacte, puisque celle-ci est réalisable matériellement.

Comme pour la première version, les valeurs transmises sont les profondeurs avant et arrière et les coordonnées de la normales en chaque pixel.

- En dehors de ces processus servant à la conversion des facettes et des quadriques, nous avons défini un processus de conversion de polyèdres convexes (voir annexe A). L'idée de base était de tester les possibilités de l'algorithme de coupe par un plan. Puisque cet algorithme permet de couper tout objet défini par deux valeurs de profondeurs avant et arrière, nous pouvons définir un polyèdre comme étant l'espace coupé par des plans, en prenant comme profondeurs aux départ Zmin et Zmax, qui sont les valeurs minimale et maximale. Comme nous avons fixé à six le nombre de cotés du polyèdre pouvant être converti, le processus emploie les valeurs fournis par six P.E.1. La normale envoyée en chaque pixel, en plus des deux profondeurs, est déterminée automatiquement à partir des normales aux plans de coupe.

Les processus généraux

- Le Processus Général de conversion d'Objets pouvant effectuer le traitement de différents objets, il lui appartient de créer et de faire s'exécuter le Processus Objet nécessaire. Le Processus Objet crée les Processus Elémentaires, leur transmet les coefficients des expressions, puis reçoit en chaque pixel les valeurs des expressions et calcule les données de l'objets en un pixel. Il transmet ensuite au processus d'élimination des parties cachées les données calculées.

- Un processus de Z-buffer reçoit les coordonnées de la position d'un point et les données associées, effectue l'élimination des parties cachées à partir d'un tableau contenant les points visibles déjà traités et la profondeur du point, puis transmet les données concernant ce point, s'il est visible.

- Les processus d'éclairement reçoivent pour un pixel donné la position du point, la normale et la couleur de l'objet visible, puis calculent la couleur en fonction des données de la scène concernant les sources (positions, types, ...) et l'observateur.

4.1.3 Organisations globales du simulateur

La première organisation du simulateur fut construite autour de l'architecture de la machine IMOGENE. Les différents problèmes liés à l'implantation de la structure de la machine sur le réseau de Transputers nous a conduit par la suite à envisager une nouvelle organisation, afin d'améliorer la vitesse du rendu et pouvoir ainsi générer des images plus rapidement. Cette redéfinition de l'organisation a porté sur le schéma général de la simulation de la machine, mais n'a pas modifié la structure des processeurs objets que nous avons définis.

La première organisation

La topologie en arbre ternaire étant très proche de la structure de la machine simulée, nous avons choisi une topologie de Transputers s'en approchant. L'organisation du simulateur doit tenir compte des différences essentielles entre la structure de la machine IMOGENE et les possibilités offertes par le réseau de Transputers :

- le nombre limité de Transputers disponibles nécessite d'utiliser plusieurs fois les processeurs de conversion, lors de l'affichage d'une scène complète. Il faut donc que le chargement des objets sur le Transputers de conversion soit effectué en fonction de la charge des processeurs.

- le système ne disposant que d'une connexion avec l'extérieur, il faut organiser un système de chargement des objets sur les processeurs de conversion prenant en considération cette contrainte.

Nous avons alors défini, en plus des processus de conversion, deux nouveaux processus permettant de gérer le système avec les contraintes définies ci-dessus :

- le processus de gestion des Transputers Objets gère la disponibilité des Transputers effectuant les conversions, reçoit les données pour l'affichage d'un objet et les transmet à un T.O. libre.

- le processus de routage reçoit et fait transiter des données vers un processus général de conversion ou vers un autre processus de routage, en fonction d'une destination pré-définie par le processus de gestion.

La répartition des processus sur la structure est effectuée en définissant les rôles des Transputers :

- le Transputer d'Affichage est sur la carte graphique GDSII, dont le contrôleur vidéo gère un écran de 720×540 pixels en 16 millions de couleurs. Le Transputer possède un lien relié au Transputer Maître et les autres liens reliés à des Transputer Z-buffer. Les processus dont il a la charge sont le processus d'éclairement et un processus de Z-buffer, ce dernier permettant d'effectuer l'élimination des parties cachées sur la globalité de la scène.

- les Transputers Z-buffer possèdent trois liens pour l'entrée des données et un lien pour la sortie (Figure 4.4). Ils peuvent alors être connectés en entrée sur des Transputers Objets et en sortie sur le Transputer d'eclairement ou reliés à d'autres Transputers Z-buffer.

- les Transputers Objets servant à la conversion des objets ont une connection reliée à un Transputer Z-buffer. Les trois autres liens sont connectés à d'autres T.O. pour former un réseau multi-pipeline pour l'acheminement des données pour la conversion. Ils ont à leur charge l'exécution en parallèle des processus pour la conversion et le Processus de Routage (Figure 4.3). La transmission de données est exécutée en priorité puisque la conversion d'un objet est un processus gourmand en nombre d'opérations et donc en temps d'exécution.

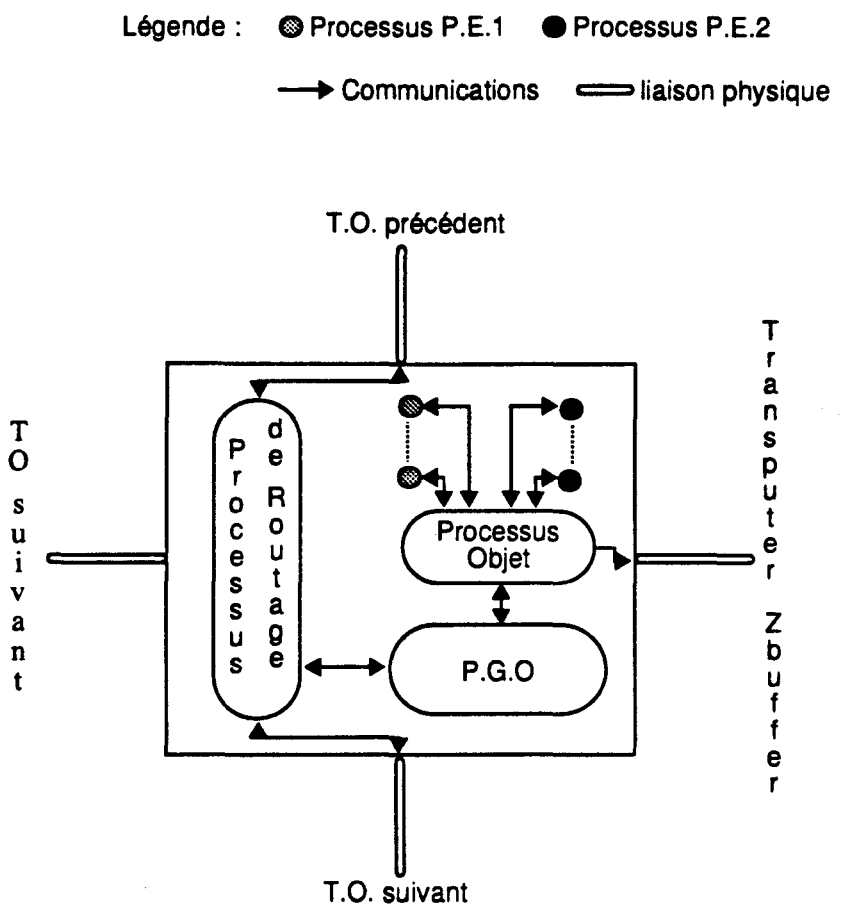


Figure 4.3 : L'organisation d'un Transputer Objet

- le Transputer Maître est réservé pour effectuer uniquement le processus de gestion des T.O. et la gestion des entrées/sorties du système. Il est connecté au noeud d'entrée et au Transputer d'affichage pour la gestion des entrées/sorties. Les deux autres liens restant servent de point d'entrée dans le réseau multi-pipeline, afin de transmettre les données de conversion des objets aux T.O. libres.

La configuration du premier simulateur (Figure 4.4) comprend 14 Transputers dont un Transputer Maître, neuf Transputers Objets, trois Transputers Z-buffer et le Transputer d'affichage. Les deux types de données qui transitent dans ce système sont les objets (du noeud d'entrée aux T.O.) et les pixels (après les T.O.). La préparation des objets est effectuée sur le Transputer du nœud d'entrée : le calcul des positions des sources dans la scène est effectué en premier, puis le traitement des objets est effectué dans leur ordre d'arrivée.

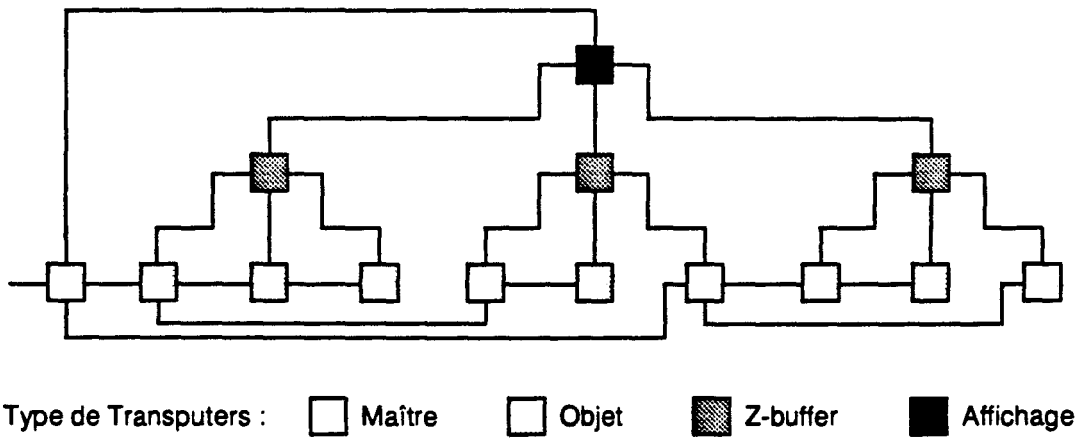


Figure 4.4 : La première organisation

Ayant remarqué que la charge du Transputer d'affichage était trop grande, nous avons par la suite ajouté un Transputer Z-buffer, ayant le rôle d'effectuer l'élimination des parties cachées sur l'ensemble de la scène, afin de supprimer le processus de Z-buffer du Transputer d'affichage.

La seconde organisation

Plusieurs raisons nous ont amené à modifier l'organisation que nous avions défini :

- La première de ces raisons est la charge supportée par le Transputer d'affichage, qui devenait alors un goulot d'étranglement dans le système. En effet, les calculs effectués par la méthode d'éclairement de Phong sont complexes et ne permettent pas de traiter suffisamment vite les données.

- La seconde raison est que nous voulions ajouter le traitement des ombres portées et des spots lumineux dans l'organisation du rendu. L'élimination des parties cachées s'effectuant en arbre sur des Transputers différents, elle n'était plus adaptée.

Avant de nous intéresser à la nouvelle organisation, nous allons rappeler les informations nécessaires aux traitement des ombres portées et des spots lumineux. Les valeurs mémorisées sont :

- la profondeur (un entier de 32 bits pour l'élimination des parties cachées)
- la normale (trois entiers de 16 bits pour les trois coordonnées)
- la couleur (un entier de 16 bits pour le numéro de la couleur prédéfinie)
- les numéros des sources (un entier de 16 bits indiquant quelle source est active en ce point)

La taille mémoire d'un Transputer étant de 2 Mo et les valeurs à mémoriser étant importantes (24 octets par pixel), il est nécessaire de répartir ces informations sur plusieurs Transputers.

La nouvelle organisation s'est alors attachée à résoudre ces problème, mais s'est éloignée de la structure de la machine IMOGENE. Nous avons gardé par contre les processus directement liés au rendu et nous avons défini de nouveaux processus pour le traitement des ombres et des spots lumineux. La charge du Transputer du noeud d'entrée servant à la préparation des données de la scène n'étant pas trop importante au vue de la charge du rendu, il n'a pas été nécessaire de la répartir sur plusieurs processeurs dans cette nouvelle organisation. Les seuls changements notables apportés dans le processus de préparation sont la gestion des sources après le calculs des coefficients des objets pour permettre l'emploi de spots lumineux, et le traitement des volumes d'ombres.

Nous avons choisi d'utiliser un parallélisme au niveau de l'image, ce qui permet de répartir la charge du calcul d'éclairement sur plusieurs Transputers et d'effectuer un Z-buffer sur une zone réduite de l'écran. Chaque Transputer d'éclairement est associé à un Transputer Z-buffer, ce qui permet de répartir les informations associées à chaque pixel et évite la surcharge des calculs d'éclairement sur des zones différentes.

Le choix du parallélisme devait prendre en considération les paramètres suivants :

- un spot doit être traité après la conversion des objets et avant l'application des ombres portées.

- un volume d'ombre occupe une zone importante de l'écran et doit être traité après la conversion des objets.

- la vitesse de rendu d'une image dépend du T.O. le plus lent, qui peut représenter un ralentissement important suivant l'organisation choisie.

- la structure des liaisons inter-Transputers est fixée au début d'une application et ne peut plus être changée ensuite.

• Nous avons le choix entre un découpage de l'écran en zones rectangulaires ou en lignes.

- Le découpage en zones rectangulaires pose des problèmes avec les deux derniers paramètres puisqu'elle ne prend pas en compte la cohérence spatiale pour la répartition de la charge de rendu d'un objet. Un Transputers Objet doit toujours être connecté au même Transputer Z-buffer et au même Transputer d'éclairement : il faut donc que la zone soit fixe. Les Transputers traitant les objets des zones fortement remplies ont une charge importante, alors que les Transputers s'occupant des zones presque vides sont peu sollicités.

- Le découpage en groupes de lignes entrelacées permet de répartir cette charge de rendu d'un objet sur plusieurs Transputers. Nous pouvons donc supposer que la charge globale de rendu sera équitablement répartie sur l'ensemble du système. Le découpage étant fixe, la gestion des informations concernant les lignes est équitablement répartie sur les Transputers Z-buffer.

Nous avons alors opté pour cette dernière parallélisation, qui ajoute une simplification de la gestion des objets aux qualités déjà rencontrées. Le Transputer de la carte graphique n'ayant que quatre liens, dont un est relié directement au transputer Maître, nous avons alors divisé l'écran en trois groupes de lignes; les différents transputers attachés à un groupe traitent donc une ligne écran sur trois.

• L'application des ombres portées nécessitant la conversion de volumes, nous avons défini des processus permettant de les convertir. A la différence des Processus Objets, les processus d'ombre ne calculent pas de normale puisqu'elle est inutile, mais emploient plus de plans de coupe, les volumes d'ombres en ayant plus.

- Le processus d'ombre quadrique permet de convertir des volumes quadriques à quatre plans de coupe, puisque le volume d'ombre d'une quadrique coupée par trois plans est construit par l'union de ceux-ci. Les valeurs en sortie sont les profondeurs avant et arrière du volume.

- Deux types de processus d'ombre polyèdrique ont été définis. Ils ne se différencient que par le nombre de plans de coupes employés (4 ou 10), puisqu'ils servent à la conversion des volumes d'ombres produits par des facettes ou des polyèdres à six côtés. Leurs schémas de fonctionnement est celui du Processus Objet Polyèdre, mais seules les deux valeurs de profondeurs sont transmises.

Les processus d'ombre permettent aussi de convertir les volumes de lumière, définis par les spots lumineux employés dans une scène.

• La topologie employée et la répartition des fonctionnalités étant différentes de la première

organisation, nous allons détailler les différences aux niveau des processus:

- La transmission des coefficients est toujours effectuée en pipeline sur les T.O., mais le routage des informations est simplifié puisqu'un objet est réparti sur trois T.O.

- Les processus de conversion ne changent pas, seules les valeurs des coefficients sont modifiées afin de tenir compte du décalage qui survient en passant d'une ligne à la suivante. Ainsi, si le processus de conversion effectue le traitement d'une ligne sur trois, les coefficients modifiés sont ceux relatifs à la variable de ligne y. Pour une fonction du premier degré $F1(x,y) = ax+by+c$, la nouvelle fonction employée lors du calcul incrémental est alors $F1(x,y') = ax+3by'+c$ avec y' la nouvelle variable de ligne, et une fonction du second degré $F2(x,y) = ax^2+by^2+cxy+dx+ey+f$ est redéfinie, pour le traitement d'une ligne sur trois, par $F2(x,y') = ax^2+9by'^2+3cxy'+dx+3ey'+f$.

- Les tests permettant de déterminer si un point (coordonnées d'un pixel et profondeur) est dans un volume d'ombre sont effectués par les processus Z-buffer, puisque ceux-ci sont les seuls à connaître la profondeur en chaque pixel visible. Les Transputers Z-buffer traitant moins de pixels, il a été possible d'associer au processus correspondant un tableau contenant l'état des sources (actif ou passif) en chaque pixel. Les Processus Z-buffer disposant des informations sur les points et les sources, le traitement des volumes de lumière leur est aussi possible.

- Les processus d'éclairement modifiés mémorisent les coordonnées de la normale et le numéro de la matière de l'objet visible en chaque pixel, puis effectuent les calculs d'éclairement à la fin de la conversion de tous les objets de la scène, y compris les volumes des spots lumineux et les volumes d'ombre. L'éclairement est effectué par la méthode de Phong, en employant des matières prédéfinies dans un tableau (taux de Rouge, Vert et Bleu de la couleur de base, coefficients diffus et spéculaires, ...)

La seconde organisation se compose de trois Transputers Objets (pouvant convertir aussi les volumes d'ombres et les volumes des spots lumineux), un Transputer Z-buffer (déterminant aussi les point à l'ombre d'une source et/ou dans le faisceau d'un spot) et un Transputer d'éclairement par groupe de lignes.

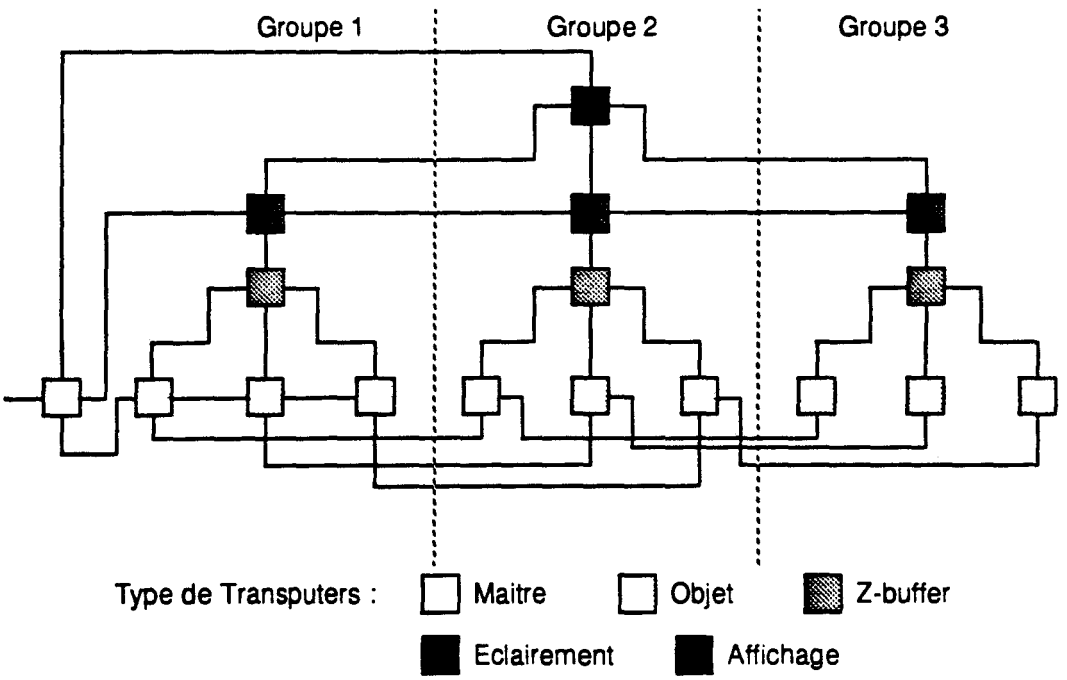


Figure 4.5 : La seconde organisation

Comme pour la première organisation, les informations transitant sur le réseau sont les objets (réels ou volumes d'ombres) et les pixels. Il faut noter cependant que les informations parvenant au Transputer de la carte graphique sont uniquement les coordonnées des pixels et leur couleur; la charge de communications est ainsi réduite. Les détails de la réalisation sont fournis en annexe B.

4.2 L'animateur

Afin de générer des images de test pour le simulateur, nous avons conçu un programme de création de scènes et d'animation, permettant une manipulation assez simple des objets et intégrant la gestion du séquencement. Ce programme est capable de communiquer directement avec le programme de simulation du rendu, par l'intermédiaire d'un fichier pipeline unix le reliant au processus de préparation des objets.

Le but de ce programme étant de créer et de gérer une animation, il faut déterminer les éléments manipulés dans une scène en séparant les objets topologiques de l'aspect géométrique. La structure de la base de données employée doit être ensuite définie en fonction de l'utilisation de ces éléments. Enfin, la gestion de l'animation et la gestion des ombres portées seront décrites.

4.2.1 La base de données des objets

Nous avons choisi d'utiliser le langage C++, puisqu'il permet de définir les caractéristiques communes à un ensemble d'éléments dans des classe génériques. La manipulation des éléments de la base de données est alors uniforme.

Les objets topologiques

Tous les objets topologiques manipulés dans une scène ont des caractéristiques communes, que nous avons regroupé dans la classe générique Objet, afin de définir des spécificités initiales (Figure 4.6). Un objet est défini par un nom, par sa position dans le repère global qui est définie par une matrice. Il peut subir des rotations, des translations, ou des changements d'échelle, qui sont des actions agissant sur sa position et qui modifient sa matrice de position. Enfin une action d'affichage permet de transmettre toutes les caractéristiques d'un objet vers le programme de rendu. Nous avons défini en plus de ces caractéristiques, la possibilité pour un objet d'être actif ou non dans la scène, sans perdre pour autant les informations qui lui sont associées. Afin de simplifier la manipulation, tout objet est créé au départ à la position origine du repère global.

```
// definition des Objets en general
class Objet {
 private:
 Name *nom;
 Mat44 matr;
 int Visible;

 public:
 // methodes de creation
 virtual void init(char *);
 virtual void reinit();
 // methodes de modification de l'objet
 virtual void chgrep(float [4][4]);
 virtual void transl(float , float , float );
 virtual void scale(float , float , float );
 virtual void rotatx(float );
 virtual void rotaty(float );
 virtual void rotatz(float );
 virtual void cacher();
 virtual void montrer();
 // methode pour la gestion
 virtual char* nom_obj();
 virtual void ecrire_obj(FILE *, Mat44);
 virtual void detruire();
};
```

Figure 4.6 : La classe Objet

La souplesse du langage que nous avons utilisé permet donc de définir tous les éléments manipulés comme étant les héritiers de la classe générique Objet. A partir de cette définition, nous avons alors créé trois classes différentes qui représentent les objets topologiques réellement utilisés dans une scène.

La première classe ObjetB est une définition des objets les plus simples, que nous avons appelés objets de base. Cette classe est définie, en plus des qualités acquises par héritage, comme ayant une description géométrique. Lors de la création d'une instance de cette classe, il faut que l'on crée la description géométrique qui lui est associée. L'affichage de la scène passe donc par l'affichage des objets de base, dans leurs positions respectives. Le fait pour un objet de base de devenir inactif dans une scène est qu'il passe de l'état visible à l'état invisible. Une source est définie comme un objet de base ayant une description géométrique particulière. Elle est manipulée par les mêmes actions que les autres objets, ce n'est que lors de l'affichage que la différence est faite. L'affichage n'est que la transmission de la position et des autres éléments liés à la source à un instant donné. Si une source devient inactive, elle passe de l'état éclairant à un état correspond à éteint.

La classe ObjetC permet de considérer différents objets visuels, comme faisant partie d'un groupe d'éléments liés par leurs positions. La caractéristique des instanciations de cette classe est donc d'avoir

une liste définissant les objets regroupés sous un nom. La création d'un tel groupe ne peut donc être effectuée que lorsque tous les objets du groupe sont instanciés. Les objets d'un groupe étant liés par leurs positions, un objet ne peut appartenir qu'à un groupe et sa position est définie dans le repère de ce groupe. La manipulation du groupe s'effectue alors par des actions sur le repère de celui-ci. Lors de l'affichage de la scène, la matrice de position du groupe est fournie à tous les objets qui lui sont liés, afin que ces derniers puissent être positionnés correctement. Comme pour les objets de base, un groupe inactif est un groupe invisible.

Enfin, la dernière classe héritière de la classe Objet est Camera, qui permet de créer plusieurs observateurs dans une scène, afin de visualiser celle-ci sous différents angles. L'information de focale associée à chaque caméra définit la distance entre l'observateur et l'écran de visualisation, ce qui permet d'appliquer différentes perspectives en fonction de la caméra choisie. Lors de l'affichage, une seule caméra est utilisée et définit la position du repère global par rapport à l'écran de visualisation.

Les primitives géométriques

La définition d'une classe générique Descripteur a été réalisée afin de prendre en compte l'aspect géométrique des éléments. Les caractéristiques communes à tous les objets géométriques visuels sont qu'ils possèdent une couleur et que leur aspects doit être connus à l'affichage. Les classes sont définies par la géométrie de la figure à afficher. Nous avons alors comme classes héritières Quadrique, Facette3P, Polyèdre et Source correspondants aux quatre figures pouvant être traitées par le programme de rendu.

La classe Quadrique définit les caractéristiques des objets quadriques, qui sont la matrice de représentation de la surface, les trois matrices des coefficients des équations des trois plans de coupe et les huit points de la boîte englobante 3D associée à la figure. La création de la surface quadrique est effectuée suivant le type de celle-ci (cône, cylindre,...) à l'aide des méthodes appropriées. Toute surface est définie dans le repère canonique de la figure correspondante. Un plan de coupe est défini à l'aide d'un point lui appartenant et d'un vecteur, qui représente sa normale ; la direction du vecteur est utilisée pour orienter le plan de coupe et définir ainsi un demi-espace.

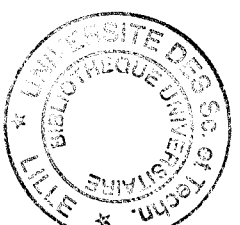
La classe Facette3P est utilisée pour l'instanciation des facettes triangulaires ayant une normale définie en chaque sommet. Les informations mémorisées pour une facette sont donc les coordonnées des trois sommets et des trois normales.

La classe Polyèdre détermine les informations nécessaires à la définition d'un polyèdre convexe à six côtés. Un tel polyèdre étant construit à partir de l'intersection de six demi-espace, les six plans limitant ces demi-espace sont mémorisés sous la forme de six matrices colonnes contenant les coefficients des équations. Comme pour la quadrique, un plan est créé par la donné d'un point de sa surface et d'un vecteur représentant la normale orientée.

La classe Source groupe les informations nécessaires à la définition d'une source de lumière. Elle définit donc, le type de la source (Directionnelle, Ponctuelle ou Spot) et sa couleur. Dans le cas des spots de lumière, une description du volume associé est utilisé par l'intermédiaire d'une instance de la classe Quadrique, qui permet de définir les informations géométriques du volume employé, mais qui n'apporte pas de restriction quant à la forme de ce dernier. La création permet de positionner les différentes caractéristiques d'une source, et certaines actions peuvent agir sur les caractéristiques d'intensité.

La structure

La structure d'une base de données est déterminée par le but final que l'on veut atteindre (dans le cas présent, l'utilisation de ses éléments pour l'affichage d'une scène).



La base de données est construite en partant du principe qu'il existe trois types d'éléments différents dans une scène. Bien que les éléments d'une scène soient identiques du point de vue de l'animation, leur rôle est complètement différent.

Le premier type regroupe tous les objets matériels de la scène, c'est à dire tous les objets ou groupe d'objets ayant une signification visuelle. Ces objets sont regroupés dans une liste appelée Scene (Figure 4.7 a) puisqu'ils sont les objets apparaissant à l'affichage.

Les sources lumineuses forment un type particulier d'objets, puisqu'elles ne sont pas matériellement représentables et ne font que contribuer à la définition de la scène. La liste Sources (Figure 4.7 b) contient toutes les sources lumineuses, ce qui permet lors de l'affichage de les différencier des autres objets.

Le dernier type que l'on peut définir contient l'ensemble des objets caméra. Ceux-ci ne font pas partie intégrante de la scène et ne sont utilisés que pour l'affichage. Leur rôle étant de définir la perception que l'on a d'une scène, un seul élément de la liste Monitor (Figure 4.7 c) les regroupant, est utilisé lors de l'affichage.

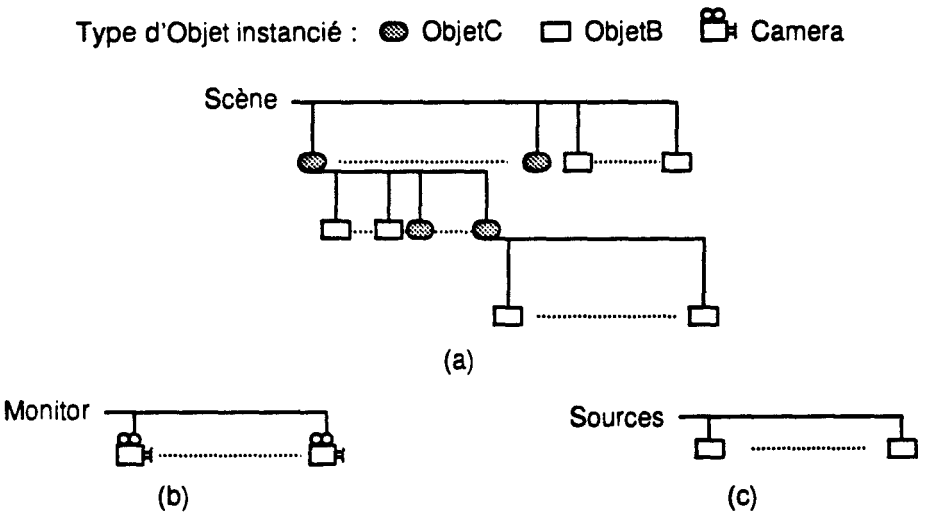


Figure 4.7 : Structure de la base de données Objet

La gestion

La gestion d'une telle structure est assez simple, puisque nous avons associé un nom à chaque élément de la base de données. Lors de la création d'un nouvel élément, il faut vérifier dans la base qu'il n'existe pas d'autre élément ayant le même nom. Une action est effectuée sur un objet par une recherche de l'objet par son nom, puis l'application de l'action sur l'élément trouvé. Nous employons aussi des actions globales, comme la transmission des données des objets matériels vers le programme de rendu ou la destruction de la scène, en diffusant dans la liste chaînée correspondante les informations relatives à l'application de ces actions.

4.2.2 Organisation générale de l'animation.

Nous voulions utiliser des séquences définies par des positions clés, afin de simplifier la construction de l'animation complète. Il nous a donc fallu définir une base de données liées aux images.

Afin de simplifier la création d'une image, les informations mémorisées sont les actions permettant de la définir et l'image antérieure ayant conduit à cette définition. Dans l'optique de simplifier la création de l'animation, nous avons séparé dans la base de données images, la définition des actions et les objets sur lesquels elles agissent. La structure de la base de données de l'animation est alors une liste chaînée d'images. Une image est définie par son numéro, la liste des objets manipulés auxquels on associe une liste des actions appliquées, et un indicateur de l'image précédente ayant servie à sa construction.

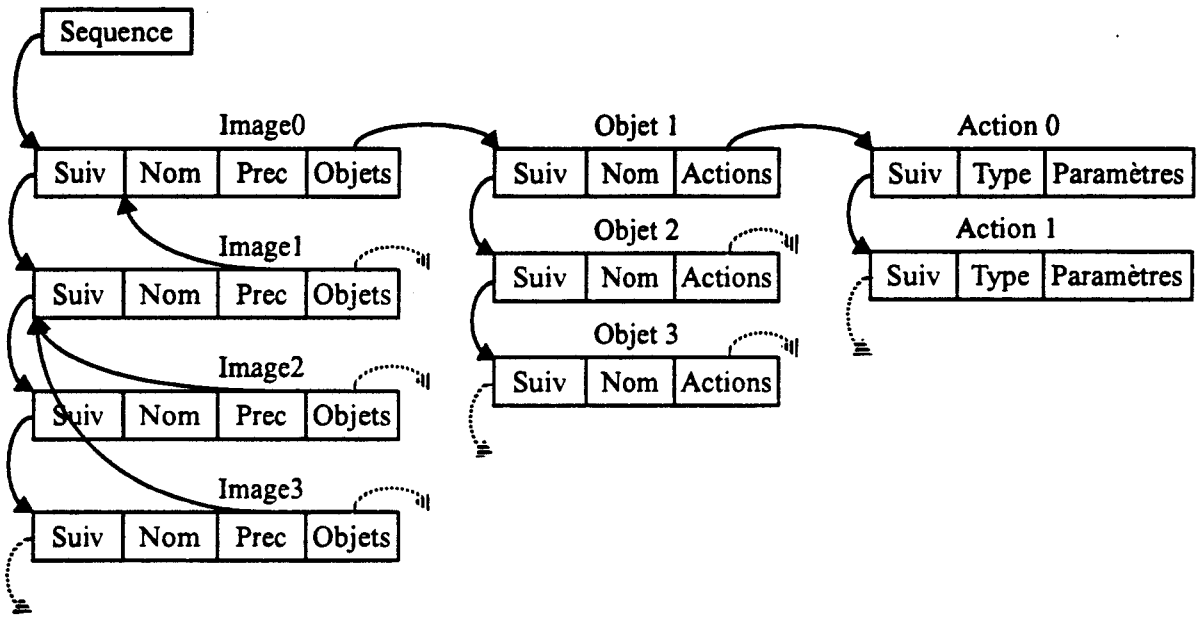


Figure 4.8 : Structure de la base de données images

Les actions sur la structure de données des images sont la création d'une nouvelle image à partir d'une image déjà définie, le choix des objets existant dans la base de données objet, que l'on veut manipuler dans une image et enfin les actions menées sur un objet pour définir sa position dans l'image.

La gestion de cette structure est effectuée par le parcours des différentes listes de la base de données, afin d'identifier l'élément que l'on veut manipuler, ou la position de l'élément que l'on désire insérer.

Afin de pouvoir utiliser à nouveau et améliorer les animations définies et pour permettre d'employer plusieurs fois des séquences, nous avons défini une syntaxe de sauvegarde des données dans des fichiers. Les deux structures des base de données sont mémorisées dans des fichiers différents, ce qui permet d'utiliser la définition d'une même scène avec une animation différente.

Pour favoriser l'interaction entre le programme et l'utilisateur, nous avons utilisé le concept offert par l'emploi de clients/serveurs Xwindow, basé sur le dialogue par l'intermédiaire de fenêtre et l'exécution par événements. L'animateur a été conçu à l'aide du programme Devguide, qui permet de créer des interfaces clients des serveurs Xwindow et qui permet de définir la gestion de ces interfaces. Le programme permettant la définition d'une scène et la création d'une animation associée à celle-ci, l'interface comporte deux parties (Figure 4.9). La partie modélisation permet la création des objets de la scène et la partie définition de séquence emploie les objets définis, afin de les manipuler pour construire les images (voir en annexe C, le détail de la création des objets et des séquences).

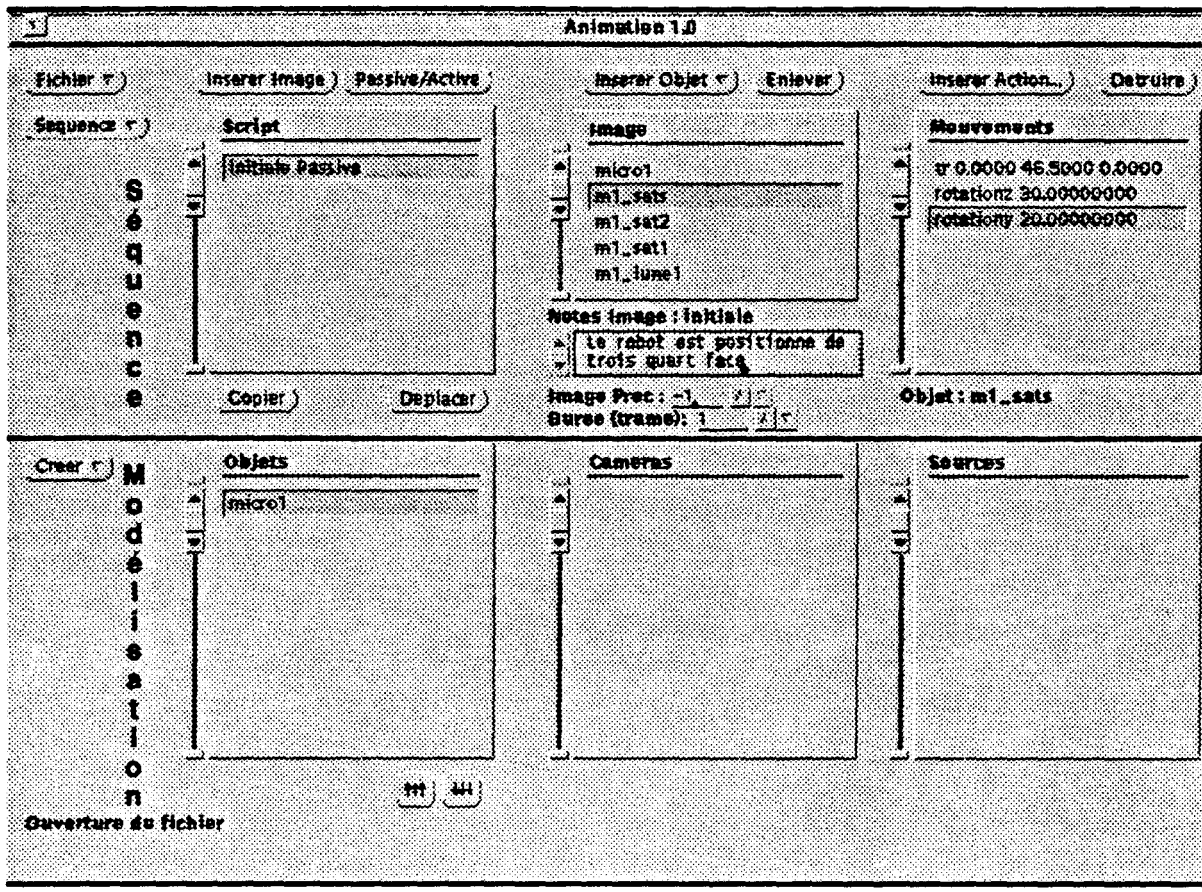


Figure 4.9 : L'interface de l'animateur

Nous avons intégré dans la partie définition de l'animation, la gestion des paramètres d'affichage et d'enregistrement.

4.2.3 La génération des ombres portées

Un algorithme de définition des ombres portées sur les volumes quadriques par la méthode de Crow ayant été proposé, nous avons créé un processus unix permettant la définition des volumes d'ombres sur les objets quadriques volumiques et facettes, afin de tester le résultat visuel de l'application de cet algorithme. Ce processus est exécuté en parallèle avec le programme d'animation, puisque ce dernier contient les objets des scènes affichées.

Les éléments de la base de données employée sont les caractéristiques d'une source et les objets de base déjà définis pour l'animateur. La base de données est représentée sous la forme d'une liste unique des objets de base de la scène dans le repère global. Lors de l'envoi des objets au programme de rendu et lorsque l'on désire intégrer les ombres portées dans une image, le programme d'animation transmet aussi les objets affichés au processus de calcul des ombres portées. Ensuite, l'animateur transmet les caractéristiques des sources de la scène. Dès la réception d'une source, le processus calcule les volumes d'ombres correspondants, connaissant les positions et les descriptions géométriques des objets présents, puis les transmet au programme de rendu : les sources sont donc traitées les unes après les autres.

4.3 Résultats

La première partie de cette conclusion est consacrée aux résultats de la simulation directement en rapport avec le P.O.Q. que nous avons défini. La seconde partie regroupe tous les enseignements que nous avons pu extraire de notre étude sur l'organisation de la simulation et de l'utilisation du simulateur.

4.3.1 La simulation

La simulation du rendu a confirmé que la primitive d'affichage quadrique peut être employée dans une machine de rendu géométrique, en démontrant la faisabilité des Processeurs Objets Quadriques.

Nous avons pu par ailleurs montrer l'amélioration intrinsèque de la qualité visuelle liée à l'utilisation de cette nouvelle primitive.

La qualité volumique de la primitive permet d'utiliser la méthode de Crow pour générer les ombres portées des objets quadriques, la définition d'un processeur permettant de convertir des volumes quadriques semble alors une solution pour intégrer la gestion des ombres portées, dans une machine de rendu géométrique utilisant des P.O.Q.

Le programme d'animation nous a permis d'effectuer des tests sur de nombreuses scènes (voir photos en annexe), en simplifiant leurs constructions et a montré la simplicité de gestion des primitives quadriques. La modélisation employée lors de la définition des scènes a montré la nécessité de définir un véritable modeleur intégrant les quadriques.

4.3.2 Bénéfices de la simulation

L'étude et la réalisation d'une simulation du processus de rendu, nous a apporté un certain nombre de renseignements à différents niveaux de la synthèse des images.

La simulation du processus de rendu nous a permis de tester différents types de processus objets dérivés du P.O.Q., comme le processus Polyèdre ou les processus d'ombre, qui ont montré la généralisation possible de l'algorithme de plans de coupe définis pour la conversion des quadriques et les autres utilisations envisageables du P.O.Q.

Nous avons pu tester également la possibilité d'employer un parallélisme au niveau de l'image, sans que cela ait d'influence sur la structure des P.O.Q. et des autres Processeurs simulés, ce qui prouve que l'utilisation de primitives d'affichage définies à l'aide d'expressions augmente la souplesse d'intégration des Processeurs de conversion dans une architecture de rendu géométrique.

Le simulateur a montré que l'utilisation d'un post-éclairement permet d'employer différentes primitives d'affichage, ce qui augmente les possibilités d'une machine de rendu utilisant le pipeline de rendu de Phong. Le simulateur permettant d'afficher des facettes et des quadriques, il a été possible de comparer ces deux primitives au niveau de la qualité de l'image générée.

La réalisation de la seconde version du simulateur nous a permis de proposer une organisation intrégrant la gestion des ombres portées et les spots lumineux. L'organisation employée montre alors qu'il est facile d'intégrer les ombres portées dans le cas du post-éclairement.

Chapitre 5 : Evaluations

L'étude de la primitive quadrique ayant montré la faisabilité de processeurs dédiés à sa conversion, il reste à définir les avantages de son exploitation dans une machine d'affichage. Ce chapitre s'intéresse donc à comparer les coûts des facettes triangulaires et ceux induits par notre nouvelle primitive au niveau de la préparation et au niveau de la conversion. Nous nous intéressons tout particulièrement au rendu avec post-éclairement des objets utilisant un découpage écran en zones.

La première partie de ce chapitre est consacrée à la machine de préparation permettant d'alimenter les processeurs de conversion. Les premières évaluations portent sur le coût de stockage des bases de données, le nombre d'opérations effectuées pendant la phase de préparation des coefficients pour la conversion, ainsi que le coût du tri si la machine d'affichage emploie un découpage en zones écran.

La seconde partie s'intéresse à la phase de conversion. Le coût des algorithmes de conversion que nous avons employés est détaillé, et une évaluation du nombre de transistors nécessaire à la réalisation d'un Processeur Objet Quadrique que nous avons défini, est exposée.

La dernière partie est une comparaison entre les facettes et les quadriques comme primitives d'affichage. Après un résumé des évaluations précédentes, nous étudierons la puissance théorique de conversion obtenue si l'on emploie la quadrique comme primitive.

5.1 La machine de préparation

Si la puissance requise avant la conversion est importante, la machine de préparation doit être structurée en conséquence, afin de ne pas devenir un facteur de ralentissement de l'affichage. Lors de l'utilisation d'une primitive dans une machine d'affichage, un point non négligeable à étudier est donc l'ensemble des opérations effectuées avant la conversion, puisque celles-ci déterminent la puissance et la structure (nombres de processeurs et organisation interne) de la machine de préparation.

5.1.1 Le coût du stockage

Un paramètre à prendre en considération lors de la manipulation d'une scène, est la taille mémoire nécessaire pour stocker les données relatives à la représentation géométrique d'une primitive. En effet, les problèmes posés lors de la manipulation de base de données dépendent pour beaucoup des accès aux mémoires, et des transferts inter-processeurs.

La facette

Dans le cas des facettes triangulaires, les données sont les coordonnées des sommets, et les normales en ces sommets, soit 6 réels par sommets et donc 18 réels par facettes.

L'organisation de la base de données d'une scène peut être effectuée suivant différents schémas

(winged-edge, edgehalf,...), et le nombre total d'informations nécessaires pour représenter un objet est difficile à estimer, puisqu'il dépend du nombre de faces, d'arrêtes et de sommets de l'objet. Le nombre d'informations utilisées pour une primitive ne peut donc pas être évalué, mais il est supérieur à 18 réels, du fait des différents pointeurs employés par la structure.

La quadrique

Les données nécessaires à la représentation géométrique d'une primitive quadrique, sont les 10 coefficients de l'équation de la surface et les coefficients des plans de coupe (4 par plan). Ainsi, si l'on désire mémoriser une primitive quadrique à trois plans de coupe, il faut 22 réels. En plus de ces informations, nous avons employé une boite englobante 3D, permettant de limiter la zone d'affichage de notre primitive, ce qui nécessite alors 3×8 réels de plus, soit au total 46 réels par primitive quadrique.

Conclusion

Si une quadrique remplace plusieurs facettes, la taille de la base de données permettant de manipuler une scène sera réduite, puisque de manière intrinsèque, la mémorisation d'une primitive quadrique nécessite 46 réels alors que la mémorisation de trois facettes utilise 54 réels. Les tests effectués par van Kleij [Klei93], sur des bases de données représentées suivant une structure de représentation par les bords le démontre par ailleurs, puisque en remplaçant une quadrique par 20 facettes le gain obtenu en moyenne est de 5 entre une base de données mémorisée sous la forme de facettes et quadriques et la même base de données stockée en facettes uniquement. L'avantage d'utiliser des quadriques peut donc se transcrire en une simplification des accès mémoire, une baisse des vitesses nécessaires aux transferts et une diminution des tailles des chemins de données.

5.1.2 La phase de Préparation.

Malgré l'évolution des processeurs qui permet un nombre d'opérations en virgule flottante sans cesse croissant, le nombre d'opérations effectuées lors de la phase de préparations est un facteur important, puisqu'il détermine la puissance nécessaire sur la machine de préparation.

Les opérations définies comme élémentaires sont l'addition, la soustraction et la multiplication. Les opérations plus complexes comme la division, ou le calcul d'une racine carrée sont équivalentes à huit opérations élémentaires [Akel88].

Cette phase comporte toutes les opérations nécessaires avant la conversion des objets depuis leur définition par le modeleur. On peut distinguer deux parties principales : le placement des objets dans le repère de visualisation et la définition des données pour l'unité de conversion.

La facette

Nous considérons ici la facette triangulaire utilisée pour les approximations. Elle est donc définie par la donnée de trois points ordonnés dans le sens trigonométrique représentant ses sommets, et par une normale en chacun de ces points.

- Les changements de repère et la perspective.

Quelques soient les méthodes utilisées pour la conversion et l'éclairement, le placement de la facette dans le repère écran reste identique. Il s'effectue d'abord par le produit de la matrice de changement de repère et la matrice de mise en perspective qui nécessite 112 opérations.

Le passage du repère global au repère observateur nécessite alors le produit de la matrice obtenue et de

la représentation des sommets en coordonnées homogènes, soit 12 multiplications et 12 additions par sommet, plus 9 multiplications et 6 additions par normale, soit un total de 117 opérations. Il faut noter que le passage des coordonnées homogènes au coordonnées cartésiennes d'un sommet utilise alors 11 opérations.

Le total pour cette partie commune à tous les algorithmes de rendu de facette est de 262 opérations.

• La préparation des coefficients

Cette partie dépend fortement des méthodes utilisées pour la conversion et pour l'éclairement. La conversion des facettes repose sur deux méthodes : le suivi de contour et la méthode par équation. De même, l'éclairement peut être calculé de deux manières différentes : Le modèle de Gouraud et le modèle de Phong.

- Le remplissage par suivi de contour :

Cette méthode nécessite de trier les 3 sommets en y, soit 3 soustractions et 3 tests. Il faut ensuite définir les 3 valeurs de différence en x et trois divisions, afin de déterminer les valeurs pour le suivi de contour. Les valeurs utilisées pour le calcul incrémental de la profondeur dans le repère écran $(o, \hat{x}, \hat{y}, \hat{z})$, sont les pentes du plans dans les repères $(o, \hat{x}, \hat{z})$ et $(o, \hat{y}, \hat{z})$, qui sont définies par 5 soustractions, 8 multiplications et 1 division. La visibilité d'une facette est définie par un test lors du calcul des pentes de profondeur. Cette méthode suppose que la facette est triangulaire dans le contour écran et nécessite alors une phase de clipping (6 opérations par sommet).

- La méthode par équations :

Le contour écran de la facette est décrit au moyen de trois inéquations en x et y. Ces inéquations sont déterminées par les équations des droites 2D reliant les sommets deux par deux. Puisque les sommets sont ordonnés, il ne faut que 6 soustractions pour les vecteurs directeurs et 6 multiplications plus 3 additions pour les constantes. Pour déterminer l'équation du plan support de la facette, il faut déterminer la normale au plan associé à l'aide de 2 soustractions supplémentaires (la troisième coordonnée n'intervenant pas pour le contour) et d'un produit vectoriel (6 multiplications et 3 soustractions). L'expression de la profondeur s'obtient alors par 1 divisions, 4 multiplications, 2 additions et 2 soustractions. Puisque la normale au plan support est déjà calculée, la visibilité de la facette est déterminée par un test.

Le coût de préparation d'une facette, sans prendre en compte les données nécessaires aux calculs de l'éclairement, est donc plus faible lorsque l'on utilise la méthode par équations (Tableau 5.1).

	Additions	Soustractions	Multiplications	Divisions	Tests	Total
Remplissage par suivi de contour	0	11	8	4	22	73
Remplissage par équation	5	13	16	1	1	43

Tableau 5.1 : Total des coûts de préparation

Nous allons maintenant aborder le coût de préparation des données utilisées pour l'éclairement. La différence essentielle entre les 2 méthodes d'éclairements des facettes est le calcul des couleurs ou des normales aux sommets. Les valeurs, que ce soit les trois composantes de la couleur ou les trois composantes de la normale, sont ensuite interpolées à l'intérieur de la facette comme la profondeur.

- Le coût du calcul des couleurs aux sommets (pour une source unique)

Ce calcul doit être effectué avant la transformation perspective. Il nécessite le calcul de l'expression $I_{objet} = I_{ambiant}C_{ambiant} + I_{diffus}C_{diffus}(\vec{N}\cdot\vec{L}) + I_{speculaire}C_{speculaire}(\vec{O}\cdot\vec{R})^{n}$, pour chaque sommet et pour chaque composante de la couleur.

Le calcul précédent est effectué avec des vecteurs normés. La normalisation d'un vecteur nécessite trois élévations au carré, deux additions, une extraction de racine carrée, une inversion de valeur et trois multiplications, soit au total 24 opérations. Les trois normales aux sommets doivent donc être normées. La source peut être ponctuelle ou directionnelle, donc il n'est nécessaire de recalculer le vecteur L et de le normer que seulement dans le dernier cas. Le calcul du vecteur $\vec{L}$ (non normalisé) est effectué par 3 soustractions. Le vecteur $\vec{R}$, qui dépend de la source et du vecteur normale, doit être recalculé à chaque sommet si la source est ponctuelle. Reste le vecteur observateur $\vec{O}$ qui est $\vec{O}(0,0,1)$, dans le cas d'un observateur placé à l'infini, et qui doit être déterminé en chaque point (donc 3 soustractions et une normalisation), si l'observateur est à distance finie.

Une fois que tous les vecteurs sont normés, il faut procéder à l'évaluation des différentes composantes de l'expression:

$I_{ambiant}C_{ambiant}$, les deux éléments étant des constantes pour un objet et une source donnée, il ne faut qu'une multiplication.

$I_{diffus}C_{diffus}(\vec{N}\cdot\vec{L})$, Le produit scalaire nécessite 6 multiplications et 3 additions, la composante pouvant alors être évaluée par 3 multiplications.

$I_{speculaire}C_{speculaire}(\vec{O}\cdot\vec{R})^{n}$. Avant de pouvoir effectuer le produit scalaire, nous devons déterminer le vecteur réfléchi $\vec{R}$ par l'expression $\vec{R} = 2(\vec{N}\cdot\vec{L})\vec{N} - \vec{L}$. Le produit scalaire $(\vec{N}\cdot\vec{L})$ ayant déjà été effectué pour la composante précédente, le calcul de $\vec{R}$ est effectué par 2 multiplications et une soustraction. Le calcul de la composante spéculaire utilise donc au total 10 multiplications, 3 additions, une soustraction, et une élévation à la puissance.

L'intensité I_{objet} est la somme des 3 composantes, il faut donc 2 additions supplémentaires. L'intensité d'un objet doit être calculée pour chacune des trois composantes de la couleur (Rouge, Verte, Bleue), donc trois fois. Le tableau ci-dessous résume le coût total de l'évaluation de l'expression I_{objet} pour une facette et pour chaque couleur.

	Normalisation	Add/Sous	Multiplications	Divisions	Puissance	Total
Calcul de l'intensité	3	9×3	16×3	0×3	1×3	207

Tableau 5.2 : Coût du calcul de la couleur aux sommets d'une facette

Le tableau ci-dessus ne tient aucunement compte du type de source et du type d'observateur utilisés. En

effet, le coût reste le même si la source et l'observateur sont tous deux directionnels, mais dans le cas où l'un d'eux est ponctuel (voire les deux), il est nécessaire de rajouter le calcul du vecteur pour chaque sommet, ainsi que sa normalisation. Le sur-coût de ce calcul est alors de 27 opérations.

Voici le coût du calcul de la couleur des sommets d'une facette pour différentes configurations de la source et de l'observateur.

Source Observateur	Directionnelle Directionnel	Directionnelle Ponctuel	Ponctuelle Directionnel	Ponctuelle Ponctuel
Calcul de la couleur	207	234	234	261

Tableau 5.3 : Coût du calcul de la couleur pour différents types de sources et d'observateur

• Le calcul de la couleur (ou des composantes de la normale) en chaque pixel :

- L'interpolation des couleurs R,V, B, lors d'un affichage par suivie de contour, nécessite la définition des pentes du "plan de couleur" pour chaque composant, soit 3*4 soustractions, 3*6 multiplications, 3*1 divisions, pour un total de 54 opérations.

- Si la méthode utilisée est celle par équations, on définit le "plan de couleur" dans l'espace $E(\grave{x}, \grave{y}, \grave{c})$, $\grave{c}$ étant le vecteur associé à la composante couleur. La normale à ce plan peut être déterminée par 2 soustractions et un produit vectoriel simplifié (4 multiplications et 2 soustractions), puisque certaines coordonnées des vecteurs utilisés ont déjà été définies pour le calcul de la profondeur. L'expression de la composante couleur est obtenue à partir de la normale par 1 divisions, 4 multiplications, 2 additions et 2 soustractions pour chaque valeur soit 3*24=72 opérations.

Nous pouvons alors distinguer en tout quatre méthodes de rendu des facettes pour une configuration de source et d'observateur donnée (Tableau 5.4), suivant les méthodes d'interpolations utilisées (suivi de contour ou équations) et suivant les valeurs à interpoler.

Valeurs Remplissage	R, V, B Suivi de contour	R, V, B Expressions	Nx, Ny, Nz Suivi de contour	Nx, Ny, Nz Expressions
Préparation des coefficients	334	322	127	115

Tableau 5.4 : Coût de la préparation des coefficients pour la conversion pour une source directionnelle et un observateur à l'infini

L'estimation ci-dessus ne prend pas en compte le passage des nombres flottants en nombres entiers pour les machines de conversion utilisant des coefficients entiers. Ce traitement utilise 2 opérations par droite

du suivi de contour, 6 opérations pour la profondeur, 11 opérations pour une expression du premier degré ou une normale et 6 opérations par couleurs.

• Le coût total de la phase de préparation et sensiblement indentique pour les deux techniques de conversion des facettes.

Valeurs Remplissage	R, V, B Suivi de contour	R, V, B Expressions	Nx, Ny, Nz Suivi de contour	Nx, Ny, Nz Expressions
Préparation d'une facette	626	641	434	449

Tableau 5.5 : Coût total de la préparation pour une source directionnelle et un observateur à l'infini

La quadrique

Les évaluations des coût de préparation sont effectués pour le P.O.Quadrique de la seconde génération (à trois plans de coupe) et pour le P.O. Patch Quadrique (à quatre plans de coupe). La surface quadrique à convertir est supposée être sous forme matricielle. Le nombre de plans de coupes peut être différent suivant la méthode de conversion utilisée, aussi allons nous étudier le coût de préparation d'un plan de coupe, puis nous étendrons les résultats au nombre de plans voulus. Les plans sont directement fournis sous forme matricielle pour le P.O.Q et décrits à l'aide de quatre points pour le P.O. Patch Quadrique.

• Les changements de repère et la perspective.

Le changement de repère est effectué à l'aide d'une matrice M_r, qui représente l'ensemble des mouvements appliqués à l'objet que l'on veut visualiser. La perspective est déterminée par la donnée de D, la distance de l'observateur à l'écran. La mise en perspective de l'objet peut alors être effectuée avec le changement de repère. Il est nécessaire de calculer une matrice M_{rp}, qui est le produit de M_r avec la matrice de mise en perspective M_p, en 112 opérations.

Contrairement aux facettes, qui sont représentées par leurs sommets, les quadriques et leurs plans de coupe sont décrits par leurs équations implicites. De ce fait, le changement de repère de l'objet n'utilise pas la matrice M_{rp}, mais la matrice inverse M_{rp}^{-1}. Or l'inversion d'une matrice comprend le calcul du déterminant (95 opérations) et de la matrice des cofacteurs M_{rp}^{*} (272 opérations), ainsi que la division de chaque élément de cette matrice par le déterminant (24 opérations), soit au total 391 opérations. Pour éviter ce calcul, il est possible pendant la phase d'animation de définir directement M_{rp}^{-1}, puisque le calcul d'une rotation inverse ou d'une translation inverse est simple à mettre en œuvre.

- L'équation d'une surface quadrique dans le repère écran est obtenue par le calcul de $M_q^{'} = M_{rp}^{-1} \times M_q \times M_{rp}^{-1t}$ soit au total 2×112= 224 opérations. La matrice obtenue contenant des informations redondante (en effet, l'équation d'une quadrique ne comporte que 10 coefficients), il est possible de supprimer 42 opérations en ne calculant que la partie triangulaire supérieure de $M_q^{'}$ (donc en 182 opérations). Comme nous avons pu le remarquer dans le chapitre 2 au paragraphe I.1.1, il est possible d'employer M_{rp}^{*} à la place de M_{rp}^{-1}, ce qui permet de ne pas calculer le déterminant et évite les divisions nécessaires à l'inversion de la matrice M_{rp}. Ceci permet donc de supprimer 119 opérations par

quadrique lors du calcul de la matrice de changement de repère.

- Un plan de coupe pour un P.O.Q peut être aussi défini par une matrice colonne M_{plan} et donc placé dans le repère écran par un simple produit de matrices $M'_{plan} = M_{pr}^{-1} \times M_{plan}$, en 28 opérations.

Comme pour la quadrique, il est possible d'utiliser M^*_{rp} à la place de M_{rp}^{-1}, puisque la matrice obtenue alors est multiple de M'_{plan} et représente donc le même plan. Nous avons indiqué que les plans de coupe des quadriques sont orientés grâce à leurs normales. L'équation obtenue en utilisant M^*_{rp} permet donc de définir la normale au plan au signe du déterminant près. Ceci implique qu'il est nécessaire malgré tout, de calculer le déterminant de la matrice M_{rp}, si l'on veut connaître l'orientation des plans de coupe. Le gain espéré des 119 opérations obtenu en utilisant M^*_{rp} se réduit à 24 opérations, puisque le signe du déterminant doit être pris en compte (la division de M_{rp} par ce déterminant n'étant pas utile). Le coût de changement de repère des trois plans du P.O.Q. est de $3 \times 28 = 84$ opérations.

- Si la quadrique a été modélisée dans un tétraèdre, les plans de coupe sont définis comme étant les côtés de ce tétraèdre. Donc pour le P.O. Patch, les plans utilisées sont définis par les quatre sommets du tétraèdre, qui sont placés dans le repère écran par un produit d'une matrice ligne avec la matrice M_{rp}. Comme pour les sommets d'une facette, ce calcul est effectué en 24+11 opérations par sommet, donc au total en 140 opérations. Les équations des plans seront alors définies lors de la phase de préparation des coefficients.

• La préparation des coefficients.

Cette phase contient l'ensemble des calculs nécessaires à la définition des coefficients utilisés par l'unité de conversion. Ces coefficients servent aux calculs de profondeur mais aussi à la détermination du contour, si l'unité de conversion utilise la première version d'affichage de quadrique.

Nous allons donc décomposer les coûts pour chaque type de coefficients :

- Les coefficients de profondeur de la quadrique

Les profondeurs de la quadrique sont définis à partir de

$$R = c \qquad S = 2ex + 2fy + 2i \qquad T = ax^2 + by^2 + 2dxy + 2gx + 2hy + j$$

et s'expriment sous la forme

$$z = \frac{-S}{2R} \pm \sqrt{\frac{S^2 - 4 \times R \times T}{4R^2}} \qquad \text{(Exp.13)}$$

Les coefficients de l'équation du premier degré nécessaire à l'évaluation de $\frac{-S}{2R}$ sont déterminés par 12 opérations.

L'équation du second degré servant à évaluer $\frac{S^2 - 4 \times R \times T}{4R^2} = \left(\frac{-S}{2R}\right)^2 - \frac{T}{R}$ peut être définie en 21 opérations.

Au total, l'ensemble des coefficients utilisés pour le calcul des profondeurs nécessite 33 opérations. La conversion des 9 coefficients en entier utilise 18 opérations.

Les coefficients de la normale à la surface quadrique sont obtenus par les expressions de ses composantes. La première phase est donc de définir les expressions des composantes de cette normale, ce qui nécessite alors 16 opérations du fait de la transformation perspective. Les 12 coefficients à transmettre à l'unité de conversion sont normalisés et convertis en entiers en 54 opérations.

- Les coefficients de profondeur d'un plan

L'équation du plan permet de déterminer les coefficients de l'expression du premier degré utilisée pour

l'évaluation de la profondeur, en 12 opérations. Les trois coefficients sont convertis en entiers en 6 opérations.

Si un plan est défini à l'aide de trois points, nous devons d'abord déterminer 2 vecteurs, puis effectuer le produit vectoriel pour obtenir l'équation. Ceci coûte alors 12 opérations supplémentaires pour déterminer les coefficients d'un plan.

- La normale à un plan est définie par 10 opérations du fait de la transformation perspective, auxquelles s'ajoutent les 15 opérations de normalisation et les 6 de conversion en entiers, soit au total 31 opérations.

Le P.O.Q emploie les coefficients de profondeurs de la quadrique, des plans, de la normale à la surface quadrique et des normales aux plans de coupe.

Le processeur objet servant à l'affichage des patchs quadriques utilise les mêmes coefficients que le P.O.Q, mais n'utilise pas les normales aux plans de coupes.

Dans le tableau ci-dessous, nous évaluons le coût de préparation pour les deux versions du processeur.

Version du Processeur	Cout Quadrique (Profondeur normale)	Coût Plans (Profondeur normale)	Total
P.O.Q.	126 (52 et 74)	147 (54 et 93)	273
P.O. Patch	125 (51 et 74)	120 (120 et --)	245

Tableau 5.6 : Préparation d'une quadrique suivant le P.O.Q. utilisé

Conclusion

Le tableau ci-dessous permet de résumer les coûts de préparation pour la primitive facette et la primitive quadrique suivant la méthode de conversion employée. L'utilisation d'un post-éclairement étant nécessaire avec les quadrique, la comparaison est donc effectuée en supposant que les facettes sont éclairées après la conversion.

La facette est triangulaire et plane, et l'objet quadrique est défini par la surface quadrique à trois plans de coupe pour l'utilisation du Processeur Quadrique et à quatre plans de coupe pour le Processeur Objet Patch Quadrique.

La colonne changement de repère comporte le prix intrinsèque de l'opération pour un objet et entre parenthèses le prix du calcul de la matrice utilisée, lorsque celle-ci est calculée pendant la phase de préparation.

Le coût global a été évalué en considérant que la matrice de changement de repère et sa matrice inverse sont définies avant la phase de préparation de l'objet. Le résultat entre parenthèses de la colonne total tient compte du coût de calcul des matrices utilisées pour les changements de repère.

Type d' Objet Converti	Changement de repère (calcul des matrices)	Préparation des coefficients	Total (avec calcul des matrices)
Facette par suivi de contour	150 (112)	172	322 (334)
Facette par équations	150 (112)	187	337 (449)
Quadrique trois plans de coupe	266 (479)	273	539 (1018)
Patch Quadrique à quatre plans de coupe	322 (479)	245	567 (1046)

Tableau 5.7 : Récapitulation du coût de préparation d'une primitive

Nous pouvons observer que la préparation d'une quadrique emploie deux fois plus d'opérations que celle d'une facette. Si une quadrique remplace plus de 2 facettes pour l'affichage d'une scène, la machine servant à alimenter les processeurs objets sera moins sollicitée et pourra alors fournir plus d'objets par image et/ou par seconde.

5.1.3 Le coût du découpages en zones écran.

Dans le cadre d'une approche objet, le découpage de l'écran de visualisation en zones rectangulaires permet d'allouer successivement un même processeur à plusieurs objets occupant des zones écran différentes. Le nombre de pixels générés inutilement (pixels traités mais n'appartenant pas à l'objet) est alors réduit.

Avant d'étudier le coût du découpage écran pour les deux primitives, nous détaillons le calcul du facteur de duplication afin de définir une probabilité de couverture de zones, qui est utilisée pour effectuer les évaluations.

Le Facteur de duplication et les probabilités

Le facteur de duplication lors d'un découpage en zones rectangulaires est défini par [Moln91] :

$$FD = \int_{ecran} \int P(x,y)\, R(x,y)\, dxdy \qquad \text{(Exp.14)}$$

Le facteur de duplication utilise $P(x,y)$, la probabilité que le centre de la boite écran d'une primitive soit le point (x,y) et $R(x,y)$, le nombre de région affecté par ce placement de la boite écran. Si l'on ignore les bords de l'écran alors :

$$P(x,y) = \frac{1}{HL} \quad \text{avec } H \text{ et } L \text{ la hauteur et la largeur des écrans virtuels.}$$

L'expression (Exp.14) devient donc :

$$FD = \int_0^{HL}\int_0^{} \frac{1}{HL} R(x,y)\,dxdy \qquad \text{(Exp.15)}$$

Si l'on remplace $R(x,y)$ par $S(x,y)$ l'ensemble des positions possibles pour des valeurs n et m données, on détermine alors la probabilité $Z(p)$ telle que $p = n \times m$ soit le nombre de zones couvertes par la boîte englobante.

$$Z(p) = \int_0^{HL}\int_0^{} \frac{1}{HL} S(x,y)\,dxdy \qquad \text{(Exp.16)}$$

Le calcul de l'intégrale peut être déduit de manière géométrique (Figure 1) comme pour de facteur de duplication, connaissant la position du centre C de la boite englobante $h \times l$ par rapport à un écran virtuel :

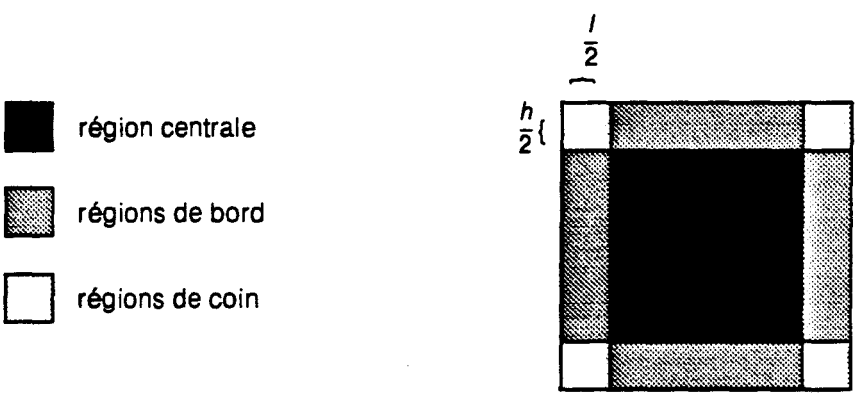


Figure 5.1 : Les différentes régions d'une zone virtuelle

- si C est dans une région de coin, quatre zones sont couvertes par la boite englobante.
- si C est dans une région de bord, deux zones sont couvertes.
- si C est dans la région centrale, un seule zone est couverte.

On obtient alors

$$Z(4) = \frac{4.\left(\frac{l}{2}\right)\left(\frac{h}{2}\right)}{HL} = \frac{lh}{HL} \qquad \text{(Exp.17)}$$

$$Z(2) = \frac{\left(2.\left(\frac{h}{2}\right)(L-l) + 2\left(\frac{l}{2}\right)(H-h)\right)}{HL} = \frac{hL + lH - 2.lh}{HL} \qquad \text{(Exp.18)}$$

$$Z(1) = \frac{(L-l)(H-h)}{HL} = \frac{LH - hL - lH + lh}{HL} \qquad \text{(Exp.19)}$$

La facette

Comme nous l'avons remarqué dans le chapitre 1, l'utilisation d'un découpage de l'écran en zones virtuelles n'est pas adapté au remplissage par suivi de contour. L'évaluation que nous avons effectué suppose donc que la facette est convertie par la méthode utilisant des expression.

Pour simplifier les calculs et éliminer rapidement un grand nombre de zones écran, on définit la boite englobante 2D de chaque facette par un tri sur les sommets. Si l'on considère une facette triangulaire, il faut effectuer 3 soustractions et 3 tests par coordonnées, soit au total 12 opérations. La boite englobante 2D est donc donnée par 4 points P1(x_{min}, y_{min}), P2(x_{min}, y_{max}), P3(x_{max}, y_{max}) et P4(x_{max}, y_{min}) définis à l'aide des valeurs x_{min}, y_{min}, x_{max}, y_{max} résultants du tri précédent.

L'ensemble des zones écran occupées probablement par la facette est déterminé à l'aide de 48 opérations.

Il est possible de ne sélectionner que les zones écran occupées effectivement par la facette, en utilisant le fait qu'au moins un des coins d'une telle zone doit être intérieur à la facette (Figure 5.2). La facette étant construite à l'aide d'équations, il suffit d'évaluer les expressions du contour pour chaque coin de chaque zone écran, pour connaître les zones écran à prendre en compte réellement. Le contour d'une facette est déterminé par trois expressions, qui nécessitent chacune 2 additions et 2 multiplications pour leur évaluation. Le test d'appartenance d'un point à une facette est donc effectué à l'aide de 15 opérations. Si l'on considère qu'il reste n×m zones écran après la première phase de tri, le nombre de points à tester est alors de (n+1)×(m+1).

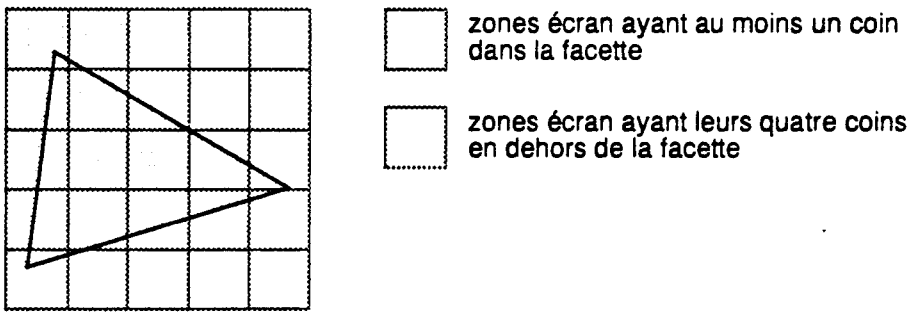


Figure 5.2 : Zones écran occupées effectivement par une facette

Nous pouvons déterminer n et m d'après le calcul du facteur de duplication [Moln91] défini par la probabilité $Z(p)$ telle que $p = n \times m$ soit le nombre de zones écran couvertes par la boite englobante.

Si l'on considère alors des zones écran de taille 32×32, et des boites englobantes de taille 20×20, on obtient alors :

$$Z(4) = 0.39 \qquad Z(2) = 0.47 \qquad Z(1) = 0.14.$$

En considérant les taille ci-dessus, nous pouvons évaluer le nombre moyen d'opérations nécessaire au tri complet d'une facette :

$$48 + (8 \times Z(4) + 6 \times Z(2) + 4 \times Z(1)) \times 15 = 145.5$$

On peut remarquer alors que le tri complet nécessite en moyenne 97.5 opérations de plus que le pré-tri.

Le passage d'une zone écran à une autre, s'obtient par une simple translation, les coefficients de la

facette dans la nouvelle zone s'obtiennent alors par 2 opérations par expression du premier degré à calculer, soit au total 14 opérations.

Dans le cas précédent, il faut donc 97.5 opérations pour supprimer 14 opérations lors de la préparation, dans au maximum 39% des cas (trois zones écran occupées sur quatre). Nous devons alors évaluer si la conversion d'une facette supplémentaire inexistante est négligeable ou non au vue du surcoût d'un tri complet.

Si la machine de conversion était orientée pixel, une facette ne coûterait qu'un cycle. La conversion d'une facette inexistante serait donc négligeable devant les opérations supplémentaires d'un tri complet. Mais dans notre cas, on considère que la machine de conversion est orientée objet, une facette supplémentaire inexistante signifie alors qu'un processeur sera utilisé pour rien. Dans ce cas le tri complet permet d'utiliser au mieux les unités de conversion, sachant que leur nombre est limité.

La quadrique

Etant donné que les objets quadriques à afficher sont définis à partir d'équations, une solution pour délimiter la partie de l'écran à traiter pour un objet est la définition de boites englobantes tridimensionnelles. Les patchs quadriques sont définis à l'intérieur d'un tétraèdre, celui-ci défini donc directement la boite tridimensionnelle à employer. Par contre, les autres objets quadriques n'ayant pas de boites englobantes lors de leur définition, il faut les construire.

- Si l'objet est un patch quadrique modélisé dans un tétraèdre, les 4 sommets du tétraèdre servent alors lors de la phase de tri. On ordonne les 4 points en 24 opérations et on détermine alors la boite écran du tétraèdre en 36 opérations, soit au total 60 opérations.

Nous pouvons affiner le nombre de zones écran (Figure 5.3) en déterminant le contour écran du tétraèdre, par le calcul de 6 équations (30 opérations) et en ne retenant que les 4 utiles (60 opérations). Le test d'un point par rapport aux 4 droites est de 20 opérations, donc si n×m est le nombre de zones écran après le pré-tri, (n+1)×(m+1)×20 est le nombre d'opérations à effectuer pour supprimer les zones écran dans lesquelles le tétraèdre n'apparaît pas.

Le passage d'une zone à une autre est défini par une simple translation en 26 opérations (12 pour l'expression du second degré et 14 pour les expressions du premier degré).

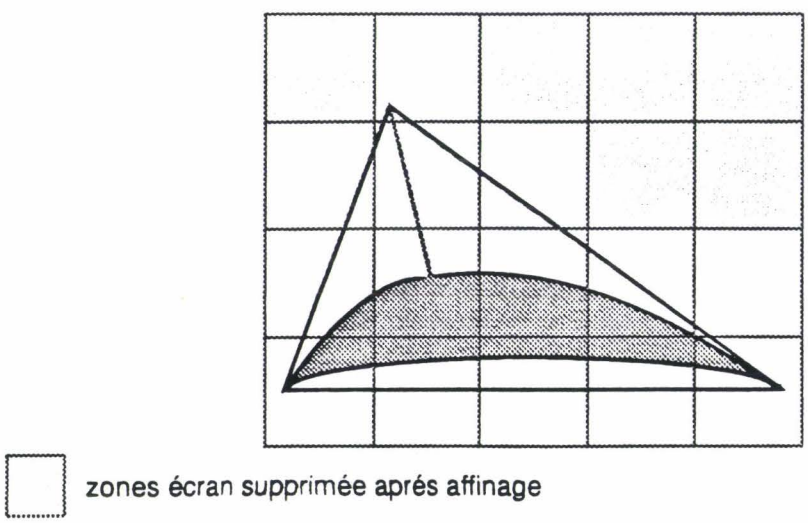


Figure 5.3 : Affinage d'un patch quadrique

La méthode d'affinage apporte un surcoût de 90+(n+1)×(m+1)×20 opérations, qui est acceptable si l'on suppose que n et m sont petits (si n=m=5, le surcoût est de 810 opérations et si n=m=2, il est de 270 opérations). Les patchs quadriques sont relativement petits ce qui laisse à penser que n et m ne seront pas des valeurs élevées et donc que le surcoût de l'affinage ne sera pas très grand.

• Les boites englobantes utilisées sont définies à l'aide de huit points pendant la phase de modélisation et permettent, moyennant un changement de repère, de situer facilement les objets que l'on veut afficher.

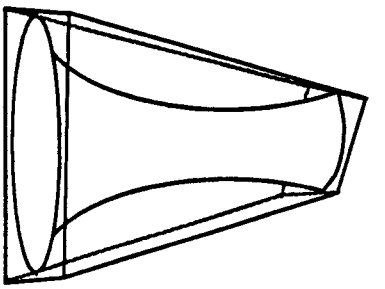


Figure 5.4 : Exemple de boite englobante 3D

- Le coût de la phase de tri, pour une quadrique donnée, doit d'abord inclure le changement de repère des 8 sommets de sa boite englobante, soit 192 opérations.

La seconde partie consiste en un tri des 8 points en x et en y en environ 112 opérations, afin de déterminer la boite écran de l'objet, c'est à dire la boite englobante 2D. Comme pour les facettes, le pré-tri permet de supprimer un grand nombre de zones écran où l'objet n'est pas présent en 36 opérations.

On peut affiner le nombre de zones écran à traiter par un test sur la boite englobante de la quadrique. La méthode consiste alors à déterminer le contour de la boite englobante (Figure 5.5) et à tester les sommets des zones écran. Pour définir le contour de la boite englobante, il faut calculer l'équation des 12 droites (projection de la boite sur l'écran) et ne retenir que les 6 définissant le contour (celui-ci ayant au maximum 6 cotés). Les 12 équations sont définies en 60 opérations, mais les tests permettant de ne garder que les 6 équations utiles nécessitent 360 opérations (pour une droite donnée, il faut en effet évaluer son équation en 6 points et tester le signe du résultat). Le test d'un point par rapport aux 6 droites du contour utilise 30 opérations, donc le nombre total d'opérations est au maximum de (n+1)×(m+1)×30, si n×m est le nombre de zones écran recouvertes par la primitive.

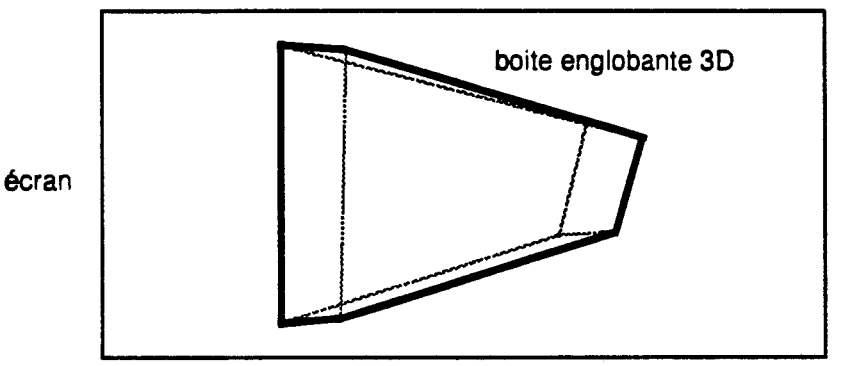


Figure 5.5 : Contour écran d'une boite englobante

Le nombre d'opérations nécessaire pour affiner le tri est alors de 420+(n+1)×(m+1)×30.

- Le passage d'une zone à la suivante est effectué en 24 opérations, puisqu'il s'agit d'une simple translation.

- Nous pouvons émettre 2 objections à l'utilisation de cette affinage:

Le coût de la méthode est trop important par rapport au bénéfice espéré. En effet, si l'on considère n et m relativement grands après le pré-tri, le nombre d'opérations pour affiner le nombre de zones écran est trop élevé (par exemple avec n=m=5, le coût est de 1500 opérations). Par contre, si n et m sont petits, le gain sera faible comparé au coût de la méthode (si n=m=2, la méthode nécessite 690 opérations alors que le gain ne peut être que d'une zone écran au maximum).

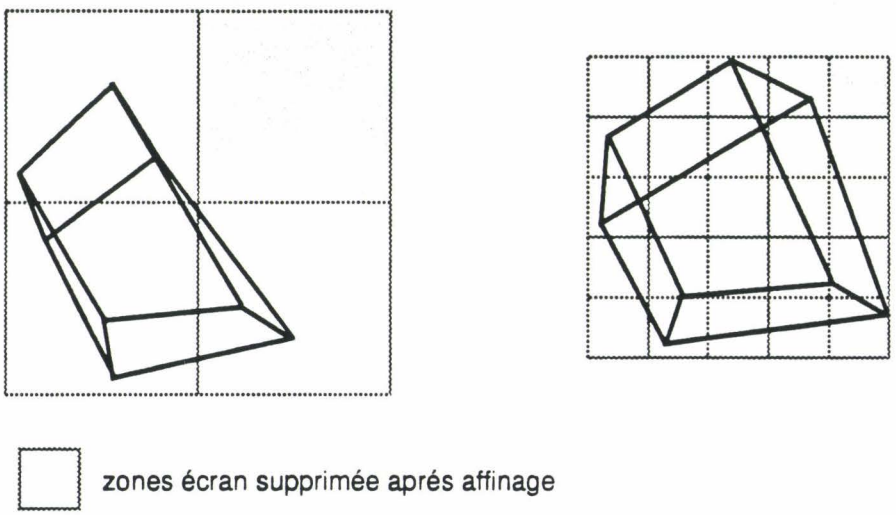


Figure 5.6 : Exemple de gain d'une zone écran après affinage du tri

Le test sur la boite englobante ne permet pas de supprimer toutes les zones écran inutiles (où l'objet n'apparaît pas) surtout lorsque la quadrique est fortement convexe (Figure 5.7). C'est cette dernière objection qui nous conduit à penser que l'affinage est inutile.

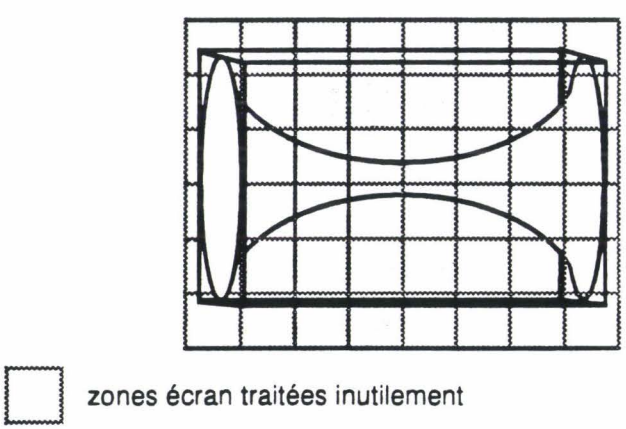


Figure 5.7 : Boite englobante d'une quadrique fortement convexe

Bilan sur le découpage écran

La première partie de la conclusion est la comparaison des coûts engendrés par le découpage par zones écran des facette et des quadriques. La seconde partie étudie la taille des zones écran, pour le découpage de l'écran, par rapport aux différentes approches de la primitive quadrique.

• Le coût du tri par zones est très lié à la définition de l'objet comme nous pouvons le constater (Tableau 5.8). Le problème avec la primitive quadrique est donc que sa définition, uniquement à base d'équations, ne comporte pas d'éléments qui ne permettent pas de la situer directement : l'ajout d'une boite englobante tridimensionnelle augmente alors considérablement le coût du tri.

L'affinage permet de supprimer des zones supplémentaires, mais son utilisation est conditionnée par deux points importants :

- Le coût engendré lors de la phase de préparation doit être comparé au coût de la conversion d'un objet supplémentaire.

L'affinage ne permet pas toujours de supprimer toutes les zones où l'objet n'apparaît pas. Cette considération est surtout importante dans le cas des objets quadriques, dont la boite englobante tridimentionnelle peut être beaucoup plus grande que l'objet lui-même.

Type d'Objet	Facette	Quadrique	Patch Quadrique
Pré-tri	48	340	60
Affinage pour n×m zones	[(n+1)×(m+1)×15]	[420+(n+1)×(m+1)×30]	[90+(n+1)×(m+1)×20]
Translation pour le passage d'une zone à une autre	14	24	26

Tableau 5.8 : Le coût du découpage écran suivant la primitive

Lors de la conversion par équation et interpolation de Phong, le coût total de la phase de découpage en zone d'une facette de boite englobante 20×20 pour des zones écran 32×32 est en moyenne de :

$$145.5 + (3 \times Z(4) + 1 \times Z(2)) \times 14 \approx 169 \text{ opérations}$$

Pour évaluer approximativement le coût engendré par un découpage en zones d'un patch quadrique, nous pouvons supposer que le rapport entre la taille des boites écran des objets et les zones est le même que pour le cas précédent. Si l'on reprend les probabilités obtenues pour les facettes, on obtient alors un nombre moyen d'opérations de

$$60 + 90 + (8 \times Z(4) + 6 \times Z(2) + 4 \times Z(1)) \times 20 + (3 \times Z(4) + Z(2)) \times 26 \approx 323$$

Comme pour le patch quadrique, si nous voulons le nombre moyen d'opérations pour un découpage en zone d'un objet quadrique, nous utilisons les probabilités obtenues avec les facettes. Ce coût est alors de :

$$340 + 420 + (8 \times Z(4) + 6 \times Z(2) + 4 \times Z(1)) \times 30 + (3 \times Z(4) + Z(2)) \times 24 \approx 995 \quad \text{opérations}$$

Pour le découpage écran, nous pouvons remarquer alors qu'un patch quadrique a un coût inférieur à deux facettes, mais que par contre l'objet quadrique équivaut à plus de quatre facettes.

• La taille des zones écran, lors du découpage de l'écran de visualisation, est fonction de l'approche (objet quadrique ou patch d'approximation). En effet, la taille d'un objet est largement supérieur à celle d'un patch, aussi il faut adapter le découpage pour que le nombre de pixels inutilement traités dans une zone écran soit minimal, tout en ayant un taux de duplication relativement faible.

Si l'on suppose que la taille moyenne de la boite englobante est de l'ordre de 80×80 pixels, l'utilisation de zones 128×128 pour le découpage écran permet d'obtenir un taux de duplication de 2.64, qui est le même que le taux de duplication des facettes (boites englobantes 20×20), pour un découpage écran 32×32.

5.1.4 Résultats de la préparation

D'après les résultats des différentes évaluations, si une quadrique remplace deux ou trois facettes, le nombre des accès mémoire est réduit et le travail effectué sur la machine de préparation s'en trouve diminué.

L'utilisation de la quadrique comme primitive de conversion allège donc considérablement le travail de préparation, ce qui permet de simplifier la structure de la machine effectuant le traitement de préparation des objets.

Il faut maintenant évaluer le coût de la conversion, qui est le plus important, avant de pouvoir conclure sur le nombre de facettes que doit remplacer une quadrique (ou un patch quadrique)

5.2 La phase de conversion

Dans le cadre de la comparaison entre la primitive facette et la primitive quadrique, il est indispensable de comparer les coûts de conversion. Bien que l'objet de cette thèse soit la définition d'un module matériel pour la conversion des quadrique, il est intéressant d'évaluer le coût de conversion suivant deux critères: le nombre d'opérations effectuées lors d'une conversion par un processeur général et le nombre de transistors d'un Processeur Objet pour la conversion. L'étude est menée dans le cadre d'une conversion en parallèle par plusieurs processeurs avec un découpage de l'écran en zones : la facette par suivi de contour n'est donc pas adaptée à ce schéma d'évaluation.

5.2.1 La conversion logicielle

La facette

Le coût de conversion d'une facette triangulaire par équation est de 7 opérations par ligne de 10 opérations par pixel (7 expressions du premier degré et 3 tests) lors d'une conversion par équations. Si la boite englobante a pour taille 20×20, le coût de conversion et de 4140 opérations.

La quadrique

La seconde méthode de conversion présentée pour convertir les quadriques à trois plans de coupe nécessite pour définir les valeurs de base à employées, 7 expressions du premier degré (1 addition par ligne et 1 par pixel), 1 expressions du second degré (3 additions par ligne et 2 additions par pixel), 1

calcul de racines carrées (8 opérations par pixel en calcul flottant), trois multiplications et 6 addition/soustractions. Le calcul des profondeurs et la normale emploie une décomposition en 2 parties qui est effectuée à l'aide d'un test suivi de 8 affectations. La coupe est effectuée par six modules d'intersections qui utilisent chacun 2 soustractions, 4 tests ainsi que 4 affectations. Le coût est alors de 95 opérations pour les pixels appartenant à l'objet et de 9 opérations pour les pixels hors contour, le nombre d'opérations par ligne étant de 10. En supposant que le nombre de pixels appartenant effectivement à la quadrique représente 75% des pixels traités, le nombre d'opérations est alors de $(0.75\times95+0.25\times9)\times n\times m+10\times m$. En prenant l'exemple d'une boite écran 128×128 pour l'objet quadrique, le nombre d'opérations est 1205504 (pour une boite écran 80×80, nous obtenons 471200 opérations).

La conversion de patch est effectuée à l'aide de 8 expressions du premier degré, 1expression du second degré, une extraction de racine carrée suivi de 3 multiplications et 6 additions/soustractions sont nécessaires aux pixels appartenant à la quadrique, ainsi que par l'utilisation d'un modules de coupe simplifiés par un plan (2 soustractions, 3 tests et 4 affectations). Le coût de conversion en soft est alors de 63 opérations aux pixels appartenant à la quadrique, de 10 opérations aux pixels extérieurs et de 11 opérations par ligne. Le coût total par patch quadrique en supposant toujours que seuls 75% des pixels appartiennent à la primitive, est alors de $(0.75\times63+0.25\times10)\times n\times m+11\times m$. Si la taille de la boite écran du patch est de 80×80, il faut alors 319280 opérations (pour une boite écran de 64×64 pixels, le coût est de 204480).

Le coût de conversion en soft d'une quadrique pleine, par les méthodes définies précédemment, est donc prohibitif puisque, au vue des résultats, une primitive quadrique devrait remplacer plus de 100 facettes.

Il est vrai que les algorithmes de conversions de quadriques utilisés lors de la définitions des processeurs objets, ont été développés dans un but d'implantation matérielle, ce qui explique leurs coûts élevés lors d'une conversion soft.

Dans le cadre d'une conversion sur processeur général, il faut alors se tourner vers les algorithmes tels que la subdivision scan-line. Reiner van Kleij [Klei93] a implémenté différentes méthodes exactes de rendu de quadriques en subdivision scan-line (voir le paragraphe II.1 du chapitre 2) ainsi que le rendu en facettes de scènes en C.S.G. Ces implémentations ont été testées sur différentes configurations de processeurs généraux. L'évaluation qu'il a menée (Tableau 5.9) montre qu'en moyenne le rendu exact des quadriques équivaut en temps au rendu de cette même quadrique facettisée en résolution 16-20 (une résolution en n signifie qu'une quadrique est représentée par n facettes triangulaires ou rectangulaires).

Résolution	Exacte	12	16	20	24
Algorithme de Subdivision	36,51	26,02	32,09	39,33	47,33
Algorithme du buffer de profondeur	35,30	39,39	47.50	57,59	68,19

Tableau 5.9 : Temps moyen de rendu (en seconde) obtenu par Reiner van Kleij

Les scènes de test utilisées par van Kleij sont définies par des arbres C.S.G. assez complexes dont les feuilles sont des quadriques ou des facettes. Dans le cadre d'un rendu géométrique, le C.S.G. ne serait alors utilisé que pour limiter les quadriques et l'arbre associé à chaque ensemble quadrique-plans de coupe serait donc limité en profondeur : les temps de conversion par l'algorithme de subdivision scan-line devrait être inférieur à ceux obtenus par van Kleij.

5.2.2 Complexité matérielle

Si l'on veut pouvoir utiliser la quadrique en temps que primitive d'affichage, il faut que le coût matériel du processeur utilisé pour sa conversion soit inférieur, ou tout au moins égal, au coût engendré par la conversion des objets primitifs qu'elle remplace.

Une estimation précise du nombre de transistors nécessaire à la réalisation d'un P.O. Patch Quadrique a été effectuée par H. Laporte [Lapo93], après que l'amélioration de l'algorithme de coupe pour un patch aie été définie.

Le processeur pourrait donc être réalisé en 225732 transistors, ce qui est un chiffre élevé mais non prohibitif, sachant que la bibliothèque de composants employés (ES2 SOLO1400 ECPD15 library) n'est pas optimale. Il serait aussi possible de réduire les coûts de la cellule de calcul de la racine carré et de la multiplication, qui sont les parties les plus coûteuses (voir Tableau 5.10), en les optimisant.

Cellules	Transistors
8 P.E.1	8×4116 = 32928
1 P.E.2	15120
1 E.R.C.	96598
3 multiplieurs	3×15872 = 47616
13 additionneurs et registres	13×(864+624)=19344
logique pour plan de coupe	1022
registres de synchronisation	13104
Total	225732

Tableau 5.10 : Coût de réalisation d'un Processeur Objet Patch

En comparaison, nous avons défini à partir de la même bibliothèque de composants, un processeur de conversion de facettes par équations, qui représente environ 35000 transistors.

Le coût du processeur patch est 7 fois plus élevé que celui d'un processeur facette, ce qui signifie qu'une quadrique doit remplacer au moins 7 facettes pour l'affichage d'une scène, afin que la réalisation d'un P.O.Patch soit raisonnable.

Nous n'avons pas d'estimation précise des coûts de réalisation des autres versions de processeurs objets

quadriques. On peut cependant approcher le coût de la version trois P.O.Q. en remarquant que seuls quelques éléments, qui ne sont pas les plus coûteux, sont foncièrement différents: le coût d'un P.O.Q version trois devrait être d'environ 300000 à 350000 transistors. En remplaçant alors plus de10 facettes par une quadrique, l'utilisation d'un P.O.Q seconde version serait avantageux. Malheureusement, l'intégration d'un processeur aussi complexe nécessite l'emploi d'une technologie de pointe, qui n'est disponible actuellement que dans le milieu industriel.

Résultats de la conversion

Nous pouvons évaluer le nombre de facettes que doit remplacer une quadrique pour que la conversion directe des quadriques soit crédible. Ainsi, lors d'une conversion logicielle, ce nombre est de une quadrique pour plus de vingt facettes. Un processeur de conversion de quadriques étant de sept à dix fois plus complexe qu'un processeur de conversion de facette, la définition d'une machine d'affichage à base de quadriques est plus à envisager, puisque le nombre de sept facettes remplacées par un quadrique est plus facile à atteindre.

5.3 La qualité du rendu

Dans les machines classiques, l'interpolation bilinéaire utilisée pour l'éclairement des facettes donne l'impression que les surfaces affichées sont courbes, mais les contours des objets et les intersections d'objets dans une image ne peuvent être convenablement représentés que par l'emploi d'un grand nombre de facettes. La perception humaine de la forme des objets étant en grande partie basée sur la silhouette des objets, l'augmentation de la qualité du rendu par le changement de primitives est indéniable, puisque les contours sont alors mieux représentés.

5.3.1 Le contour des objets

Lorsque l'on cherche à représenter un objet par un ensemble de facettes, le contour de celui-ci est défini par une suite de segments de droites. Afin de diminuer le défaut visuel, il est nécessaire d'augmenter le nombre de primitives lors de la facettisation. Les quadriques ayant un contour défini par des coniques, les objets sont mieux représentés lors de l'affichage.

Par exemple, l'approximation d'une sphère par un "petit" nombre de facettes est suffisante lors de l'affichage pour rendre l'aspect courbe de la surface, grâce aux méthodes de Gouraud ou de Phong. Le contour de l'objet affiché étant cependant polygonal, il est nécessaire d'augmenter le nombre de facettes afin de produire une image réaliste (Figure 5.8).

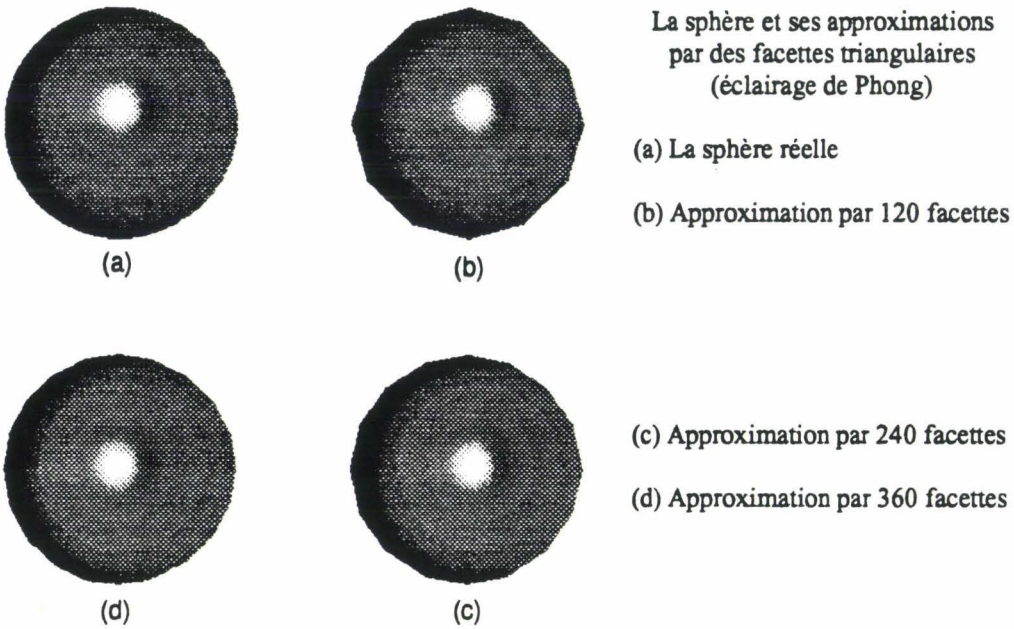
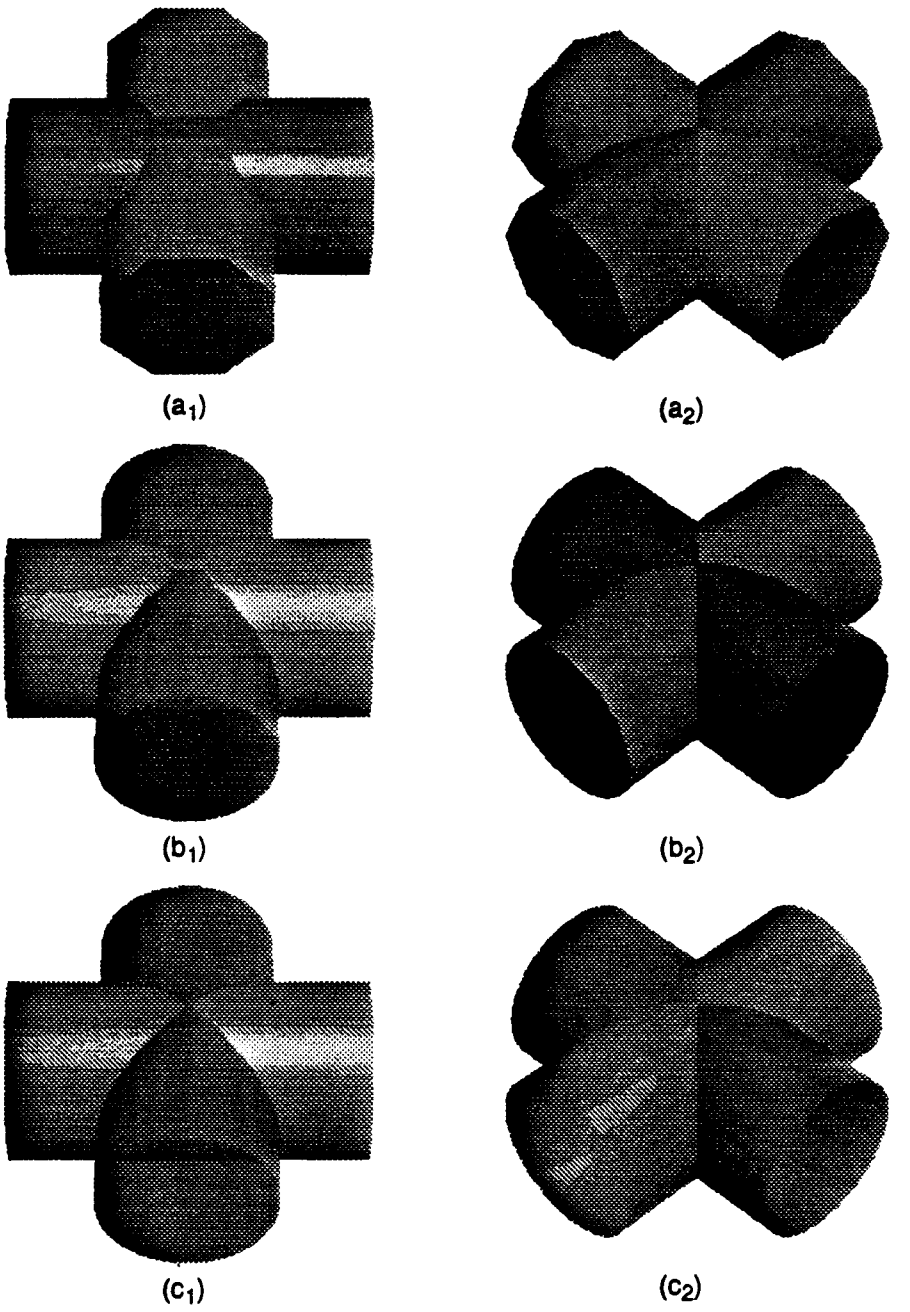


Figure 5.8 : La sphère et le rendu en facettes
(images réalisées à l'aide du simulateur décrit au chapitre 4)

5.3.2 Les intersections

L'emploi d'un Z-buffer lors du rendu permet d'effectuer l'élimination des parties cachées simplement, mais sur les machines classiques, la primitive employée pour la conversion est la facette plane : la silhouette de l'intersection de deux objets facettisés est définie par un ensemble de segments de droites. Pour obtenir une bonne approximation des intersections lors de l'affichage, il est alors impératif d'employer un grand nombre de facettes. Lors de l'affichage, l'utilisation des quadriques permet alors d'améliorer l'aspect courbe des silhouettes des intersections d'objets (Figure 5.9).



(a_1) et (a_2) sont les intersections des cylindres approchés par huit facettes quadrilatères

(b_1) et (b_2) sont les intersections des cylindres approchés par vingt facettes quadrilatères

(c_1) et (c_2) sont les intersections des vrais cylindres

Figure 5.9 : Intersections de deux cylindres suivant l'approximation
(images réalisées à l'aide du simulateur décrit au chapitre 4)

Van Kleij estime que le rendu d'une quadrique à l'aide de 20 facettes n'est pas de très bonne qualité, ce que confirment les figures précédentes. Cependant il est difficile d'estimer le nombre de facettes remplacées par une quadrique lors de l'affichage d'autres objets. En effet, les études de Damhen et de Guo pour l'approximation de surfaces de haut niveau (voir chapitre 3), porte sur une scène déjà facettisée. Il faut donc attendre la définition d'un modeleur permettant de définir des objets en quadrique pour se prononcer quant au nombre effectif de facettes remplacées.

5.4 Bilan

Dans une première partie, nous évaluons un pipe-line de Processeur Objet Patch quadrique. Nous montrons les capacités des Processeurs à être intégrés dans une machine de rendu, ainsi qu'un des grands avantages de la primitive quadrique, qui permet un coût de préparation assez faible. Nous utilisons le Processeur Objet Patch quadrique pour cette évaluation, car des deux processeurs objet définis, celui-ci sera certainement le premier que nous réaliserons.

La seconde partie permet de déterminer les avantages d'utilisation des quadriques dans les machines de rendu géométrique. Au vu des résultats obtenus dans ce chapitre, nous alors démontrons que la quadrique est une primitive envisageable pour l'affichage.

5.4.1 Puissance théorique d'un pipeline de P.O.Patch

La puissance intrinsèque d'un système composé de Processeurs Patch Quadrique organisés en pipeline selon une approche objet serait limitée par le nombre effectif de processeurs pouvant être utilisés [Karp93]. Si l'on suppose que le système travaille avec un découpage de l'écran en zones, et que le chargement des coefficients et la conversion sont parallèles, le nombre maximum de Processeurs Objet du pipeline est défini par :

$$No_{max} = \frac{L \times H \times To}{Co}$$

avec L, H les dimensions des zones écran, To le temps de conversion et Co le temps de chargement.

Pour le calcul de Co, on peut considérer que les 3 coefficients d'une expression linéaire sont chargés en 1,5 cycles (pour un bus de données sur 64 bits) et que les 6 coefficients d'une expression quadratique le sont en 6 cycles.

Le chargement des coefficients qu'un patch quadrique nécessitent alors 20 cycles (8 expressions linéaires, 1 expression quadratique et 3 valeurs pour la normale). Si l'on considère une vitesse de chargement de 50ns (pipeline alimenté par un processeur i860XP), nous obtenons Co = 1000ns.

Si les unités de conversion fonctionnent à 33 MHz, le nombre maximal de Processeurs Objet Patch dans un pipeline est alors d'environ 490 pour un découpage écran en zones 128×128 pixels et d'environ 120 pour un découpage en 64×64.

La puissance de conversion du pipeline, sur une zone écran 128×128, est alors de 490×2000=980000 patchs quadriques par seconde (960000 patchs quadriques/seconde pour une zone écran 64×64). Si l'on suppose que les patchs ont une boite écran de 80×80, le taux de duplication est alors de 2,64 (5 pour un découpage en zones écran 64×64) et le pipeline a alors une puissance d'environ 370000 patchs quadriques par seconde (192000 patchs quadriques/seconde avec des zones écran 64×64). Nous avons remarqué que le P.O.Patch étant 7 fois plus complexe qu'un processeur facette, un patch quadrique doit alors remplacer au minimum 7 facettes, ce qui signifie dans le cas du pipeline décrit ci-dessus, que la puissance serait d'environ 2590000 équivalents facettes par seconde.

Par ailleurs, l'utilisation d'un tel système permet de soulager considérablement le travail sur la machine de préparation, puisque la préparation des 370000 patchs quadriques avec le calcul des matrices de changement de repère et un découpage en zones, nécessite environ 500 MFlops alors que le coût de préparation de 2590000 facettes est de 1600 MFlops.

5.4.2 Avantages de la quadrique

Les résultats obtenus par van Kleij montrent qu'une quadrique remplace au minimum 20 facettes, or l'emploi de Processeurs de conversion de quadriques devient envisageable à partir d'un seuil nettement inférieur. La primitive quadrique peut alors être utilisée avantageusement dans les machines de rendu en remplacement de la primitive facette.

Lors de la définition et de l'utilisation d'une machine de rendu en facettes, un des problèmes actuellement rencontrés est la puissance nécessaire sur la machine de préparation pour effectuer les opérations de l'avant conversion (changements de repère, définition des coefficients utilisés par les unités de conversion, ...). Le besoin élevé de puissance sur la machine de préparation (600 MFlops pour 1000000 facettes/s) nécessite alors de définir une architecture de processeurs pouvant répondre aux exigences des unités de conversion (pour ne citer qu'un exemple, les Reality Engine de Silicon Graphics utilisent 8 processeurs i860XP pour la phase de préparation des données [Sili92]). En prenant les conditions des évaluations pour le découpage en zones écran (facettes de boites écran 20×20 sur des zones 32×32 et quadriques de boites écran 80×80 sur des zones 128×128), le coût de gestion d'une quadrique est 3 fois inférieur à celui-ci des 7 facettes remplacées. Si l'on suppose qu'une quadrique remplace 7 facettes, le P.O.Patch étant 7 fois plus cher que le processeur facette, les différents points abordés dans les évaluations permettent de conclure que la primitive quadrique est plus avantageuse pour le rendu géométrique que la primitive facette, puisqu'elle permet de diminuer les différents coûts de l'avant conversion. Or le nombre de facette remplacées par une quadrique est certainement supérieur à 7, la puissance requise pour la préparation des quadriques dans une machine de rendu permet une simplification de la structure de la machine de préparation.

Une autre conséquence de l'utilisation des quadriques pour l'affichage est une diminution du nombre de données devant transiter entre la machine de préparation et les unités de conversion. Le nombre de données (entiers sur 24 bits) à fournir pour la conversion est de 7×3=21 pour une facette et de 39 pour une quadrique (8*3+3 entiers sur 24 bits et 6 entiers sur 48 bits). Si l'on reprend le rapport de une quadrique pour 7 facettes évoqué ci-dessus, l'emploi des quadriques permet de diviser par 3 les communications (ou la taille des chemins de données) entre les unités de préparation et les unités de conversion.

En utilisant peu de Processeurs Objet Patch quadrique, il est possible de définir un système relativement puissant, puisque, dans les conditions énoncées précédemment, 100 processeurs peuvent permettre d'afficher environ 75750 patchs quadriques/s (soit plus de 530000 équivalents facettes) en ne nécessitant que 100 Mflops pour la préparation.

Conclusion

La tendance actuelle est d'employer un nombre de facettes toujours plus grand afin d'améliorer la qualité des images, ce qui augmente le nombre de primitives à afficher dans les scènes. Une solution pour éviter l'augmentation du nombre d'objets traités est de remplacer la facette par une primitive d'affichage plus complexe. Nous avons, dans ce travail, étudié les possibilités offertes par les quadriques dans le domaine du rendu géométrique.

La première approche employée nous a conduit à mettre en œuvre une première méthode de conversion des objets en pixels. A l'usage, nous nous sommes aperçus que cette solution restreint la géométrie des objets aux quadriques coupées par deux plans parallèles. Nous avons alors décidé de suivre une seconde approche, utilisant les principes du traitement des arbre C.S.G. La définition d'une méthode générale de coupe des quadriques par des plans quelconques, a ainsi permis de définir un Processeur Objet de conversion des Quadriques, qui ne limite ni le nombre de plans utilisés, ni leur orientation.

Une des qualités naturelles de la primitive quadrique étant la possibilité de l'employer comme surface ou volume, nous avons adapté la méthode d'application des ombres portées de Crow, au rendu utilisant les quadriques. Nous avons défini pour cela une méthode de construction des volumes d'ombre des objets quadriques.

Pour simplifier l'organisation du processeur de conversion et ainsi faciliter une réalisation effective, nous avons défini un nouveau processeur affichant uniquement des patchs quadriques, que nous avons nommé Processeur Objet Patch.

La conversion des quadriques ayant été montrée, nous avons évalué et comparé les coûts d'utilisation des deux primitives, afin de déterminer si la quadrique a un intérêt réel pour l'affichage rapide de scènes. Nous nous sommes ensuite penchés sur les conséquences de l'emploi des quadriques lors de la production d'images.

Les études menées montrent que la quadrique est une primitive crédible pour les machines de rendu géométrique :

Les processeurs quadrique étant d'une complexité matérielle de moins de dix fois celle d'un processeur facette, ils sont réellement exploitables dans une machine de rendu rapide. La réalisation d'un programme de simulation des Processeurs Objets Quadrique a démontré la faisabilité d'une telle machine et a mis en évidence que l'emploi d'un parallélisme au niveau de l'image est possible avec la structure choisie pour les processeurs.

La possibilité de convertir des quadriques volumiques ou surfaciques permet deux types d'approche lors de la modélisation des scènes. Il est évident que l'affichage à l'aide de quadriques offre des possibilités difficilement accessibles par un rendu en facette. Par ailleurs, les Processeurs Objets définis peuvent également être employés dans une machine d'affichage direct d'arbre C.S.G.

La qualité des quadriques de pouvoir représenter des volumes permet l'application des ombres portées par la méthode de Crow, celles-ci intervenant de manière prépondérante pour le réalisme des images

obtenues. Une machine à base de Processeurs Quadrique est donc à même de produire des images plus réalistes.

Actuellement, un des problèmes rencontrés par les machines de rendu en facettes est la définition d'une machine de préparation puissante afin d'alimenter rapidement les unités de conversion. La primitive quadrique est d'autant plus intéressante que son utilisation permet une diminution importante de la puissance nécessaire pour les calculs effectués en préparation. La machine de préparation est ainsi moins sollicitée, ce qui permet de simplifier sa structure. De plus le flot de communication entre les unités de préparation et les unités de conversion étant moins important, la réalisation d'une machine de rendu est plus facile.

L'utilisation des Processeurs Objets Quadriques nécessite l'emploi d'un post-processeur d'éclairement. De nombreux travaux laissent à penser que le post-éclairement sera utilisé dans quelques années par les machines du commerce. L'utilisation des Processeurs Objets quadrique en rendu géométrique est donc réaliste.

Les études sur la quadrique étant relativement récentes, il reste encore des problèmes à résoudre. Les principales voies à explorer sont :

L'utilisation d'objets volumiques en rendu. Les volumes sont certainement des objets intéressants, puisqu'ils offrent des possibilités inaccessibles au rendu surfacique. Cependant leur emploi ne permet pas une définition simple des objets complexes en modélisation et la conversion d'un volume est plus coûteuse que celle d'une surface. Poursuivre les travaux sur l'utilisation de l'aspect volumique des objets en rendu géométrique semble une voie prometteuse.

La définition d'un modeleur permettant de représenter les objets d'une scène à l'aide de quadriques. C'est actuellement l'étude la plus importante, si l'on veut employer la quadrique en rendu géométrique. Le premier chemin envisagé est l'utilisation de patchs (ou de macro-patchs) quadriques comme primitives de modélisation, qui permet d'éviter la conversion classique des primitives de modélisation en primitives d'affichage. La seconde voie qu'il faut explorer est l'approximation des surfaces de haut niveau (Nurbs, Bézier, ...) par des quadriques, puisque les modeleurs actuels emploient ces primitives.

L'anti-aliassage et l'application de textures. Ces deux problèmes sont moins cruciaux, mais il est nécessaire de s'y attacher pour que le rendu à base de quadriques puisse un jour supplanter les machines à facettes. L'anti-aliassage des quadriques peut être effectué par sur-échantillonnage, qui est la méthode actuellement la plus utilisée par les machines de rendu en facettes. L'application des textures est un problème plus complexe, puisque les méthodes employées habituellement pour les facettes ne peuvent pas être utilisées avec les quadriques.

Pour conclure, nous reprendrons le résultat de l'évaluation effectuée pour un pipeline de processeurs patch quadriques, qui démontre clairement les avantages de la quadrique pour le rendu géométrique. Il faut donc que les études des différents points encore non résolus à ce jour se poursuivent, pour confirmer que la primitive quadrique apportera un jour sa contribution à l'évolution des machines de rendu géométrique.

ANNEXES :

Annexe A : Le Processeur Objet Polyèdre

Annexe B : Le simulateur

Annexe C : Utilisation du Programme d'Animation

Annexe D : Recueil d'Images

Annexe A
Le Processeur Objet Polyèdre

L'algorithme de coupe par un plan utilisée pour la conversion des objets quadriques en pixels nous a conduit à définir la structure d'un processeur de conversion de volumes polyèdriques convexes. Nous présentons dans cette annexe la conversion d'un volume polyèdrique et les avantages liés à son utilisation en rendu géométrique.

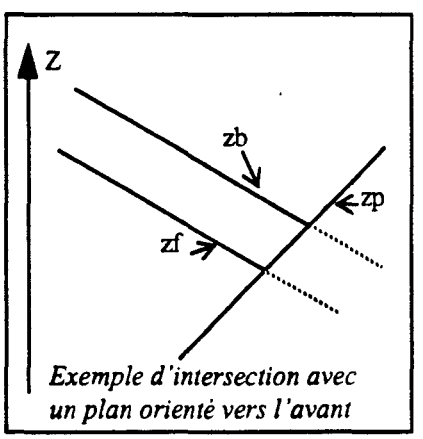
A.1 Les Polyèdres

On peut considérer un polyèdre convexe comme l'intersection de demi-espaces. L'avantage de cette structure réside dans le fait qu'un demi-espace est délimité par un plan, dont la profondeur est définie par une expression du premier degré. Le nombre de P.E.1 (Processeur Elémentaire calculant une expression du premier degré) utilisés pour la définition du volume est équivalent au nombre de faces du polyèdre.

A.1.1 L'affichage

En tout pixel (x, y), le volume convexe est construit en restreignant successivement l'espace initial par les demi-espaces définis par les faces du polyèdre.

Soit *zf* et *zb*, les profondeurs avant et arrière du volume en cours de construction, et *zp* la profondeur et *Np* la normale du plan utilisé pour une étape du processus. Suivant l'orientation de ce plan, *zf* et/ou *zb* seront modifiées par l'algorithme de calcul d'intersection (Figure A.1).



Algorithme d'intersection avec un demi-espace

si le plan frontière du demi-espace est perpendiculaire à l'écran
alors si (zp<0) alors zf=zb=Zmax (valeur de fond d'écran)
si le plan est orienté vers l'avant
alors si (zp>zb) alors zf=zb=Zmax
sinon si (zp>zf) alors zf=zp
Nf=Np
si le plan est orienté vers l'arrière
alors si (zp<zf) alors zf=zb=Zmax
sinon si (zp<zb) alors zb=zp
Nb=Np

Figure A.1 : Rappels sur la coupe par un plan

Pour la première itération, les profondeurs avant et arrière sont Z_{min} et Z_{max}, valeurs de front et de fond

d'écran. L'application de cet algorithme pour chaque demi-espace permet d'obtenir le volume polyèdrique.

A.1.2 Le Processeur Objet Polyèdre

La méthode de coupe générale permet d'utiliser un seul P.E.1. et un module effectuant l'intersection par face du polyèdre. Du fait de cette modularité, le Processeur Objet Polyèdre peut convertir des polyèdres convexes avec beaucoup de faces. En fait le nombre de faces n'est limité que par la définition matérielle du P.O.P.

Le P.O.P. permettant de convertir des polyèdres à 6 faces comprend 6 P.E.1 et 6 modules de coupe par un plan (Figure A.2).

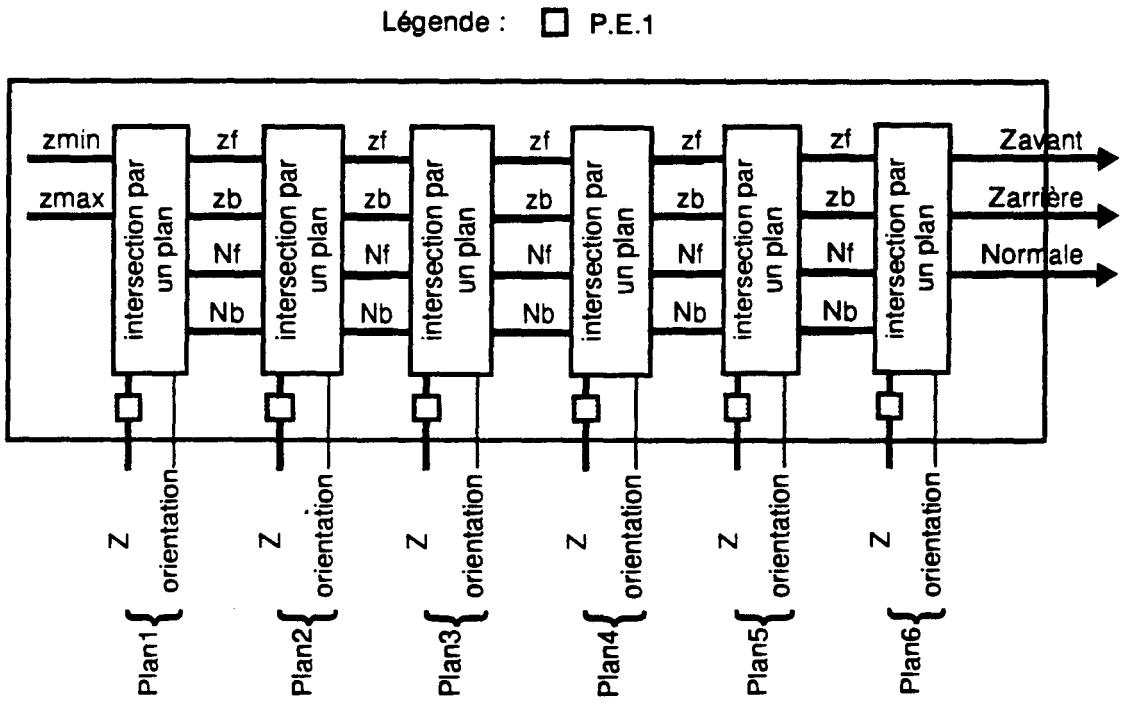


Figure A.2 : Le Processeur Objet Polyèdre

Le processeur utilisant des expressions pour la conversion des polyèdres, il est possible de l'utiliser avec un parallélisme au niveau de l'écran.

A.2 Evaluations

Ce paragraphe présente l'évaluation de la préparation des coefficients pour le Processeur de conversion des Polyèdre à six faces.

Le changement de repère est effectué à l'aide d'une matrice M_r qui représente l'ensemble des transformations appliquées à l'objet que l'on veut visualiser. La perspective est déterminée par la donnée de D, la distance de l'observateur à l'écran. La mise en perspective de l'objet étant effectuée avec le changement de repère, il est nécessaire de calculer une matrice M_{rp}, qui est le produit de M_r avec la matrice de mise en perspective M_p (112 opérations).

Le changement de repère de l'objet n'utilise pas la matrice M_{rp} mais la matrice inverse M_{rp}^{-1}. Or l'inversion d'une matrice comprend le calcul du déterminant (95 opérations) et de la matrice des cofacteurs M_{rp}^{*} (272 opérations), ainsi que la division de chaque élément de cette matrice par le déterminant (24 opérations), soit au total 391 opérations. Un plan peut être défini par une matrice colonne M_{plan} et donc placé dans le repère écran par un simple produit de matrice $M_{plan}' = M_{rp}^{-1} \times M_{plan}$, en 28 opérations. L'équation obtenue en utilisant M_{rp}^{*} à la place de M_{rp}^{-1} permet de définir l'équation du plan dans le nouveau repère, mais la normale est alors définie au signe du déterminant près. Il est donc nécessaire de calculer le déterminant de la matrice M_{rp}, si l'on veut connaître l'orientation des plans de coupe. Le coût de changement de repère des six plans du polyèdres est de $6 \times 28 = 168$ opérations.

La profondeur d'un plan en chaque pixel (x, y) est une expression du premier degré $z = -\frac{a}{c}x - \frac{b}{c}y - \frac{d}{c}$. Si on calcule en premier $-\frac{1}{c}$, la préparation nécessite $6 \times (9+3) = 72$ opérations. Les normales aux plans sont définies directement à partir des coefficients des équations, donc seule une normalisation (24 opérations) doit être effectuée. La préparation des coefficients du polyèdre est effectuée en $72 + 6 \times 24 = 216$ opérations.

La conversion en entier coûte 36 opérations pour les coefficients de profondeur et 36 opérations pour les composants des normales.

Au total, l'ensemble des opérations utilisées pour la préparation d'un polyèdre à six faces nécessite 600 opérations, ce qui est moins de deux fois le coût de préparation d'une facette. L'affichage direct d'un polyèdre à six faces nécessite quatre fois moins de calculs en préparation que l'affichage de l'ensemble des facettes le composant.

A.3 Conclusion

L'utilisation du polyèdre comme primitive n'est pas, a priori, un besoin lié à la représentation d'une scène. En effet, il ne s'agit pas d'un élément naturel de modélisation. Mais, la génération des ombres des facettes amène au calcul de volumes polyèdriques, dont le nombre est proportionnel à celui des facettes, ainsi qu'à celui des sources lumineuses. La diminution du coût de préparation par l'utilisation du Processeur Objet Polyèdre, se traduit par une baisse de la complexité matérielle de la machine hôte, lors du rendu d'une scène.

Cette primitive est intéressante puisqu'elle permet la définition directe de volumes, le P.O.P. peut donc être facilement utilisé pour le rendu avec ombres portées ou pour l'affichage d'arbres C.S.G.

Annexe B
Le simulateur

L'organisation du simulateur utilisé pour les tests sur les Processeurs Objets et la simulation du Processeur Objet Quadrique sont détaillées dans cette annexe.

B.1 L'organisation du simulateur

Nous décrivons dans ce paragraphe la seconde organisation du simulateur (chapitre 4), telle qu'elle est implantée sur un réseau de Transputers (Multicluster II). La définition du simulateur a été effectuée sous l'environnement de programmation Mtool en langage OCCAM. Le simulateur est connecté à un programme d'animation (voir annexe C) qui assure la gestion des images. Afin d'éviter un temps de communications important, le simulateur reçoit les données du programme d'animation, mais ne lui transmet rien.

B.1.1 L'organisation générale

Le simulateur emploi 1 Transputer pour la préparation des coefficients et 17 Transputers pour le rendu. Il est composé deux parties (Figure B.1):

- Un programme dont le nom est préfixé par le mot clé EXE, qui est exécuté sur le nœud d'entrée du réseau de Transputers. Ce programme reçoit les données du programme d'animation décrit dans le chapitre 4, calcule les coefficients et autres données nécessaire au rendu, puis les transmet au programme de rendu.
- Un programme de rendu dont le nom est préfixé par le mot clé PROGRAM, qui permet de définir la configuration du système et les processus employés, ainsi que la répartition de ceux-ci sur les 17 Transputers utilisés.

Afin de rendre la programmation plus facile et de donner une certaine homogénéité aux processus du rendu, nous avons créé des bibliothèques de processus et de procédures. Le première de ces bibliothèques (*libimo*) contient tous les processus de conversion d'objets que nous avons utilisés lors de la création des images, ainsi que les autres processus et procédures employés par ceux-ci. Cette bibliothèque permet aux Transputers Objets la conversion de n'importe quel objet, dès la réception de ses coefficients. Les autres bibliothèques permettent les transmissions des données dans le pipeline formé par les Transputers Objets. Les transmissions peuvent être effectué d'un Transputer vers un autre Transputer (*libcoef*) ou vers deux Transputers (*libcoefdup*).

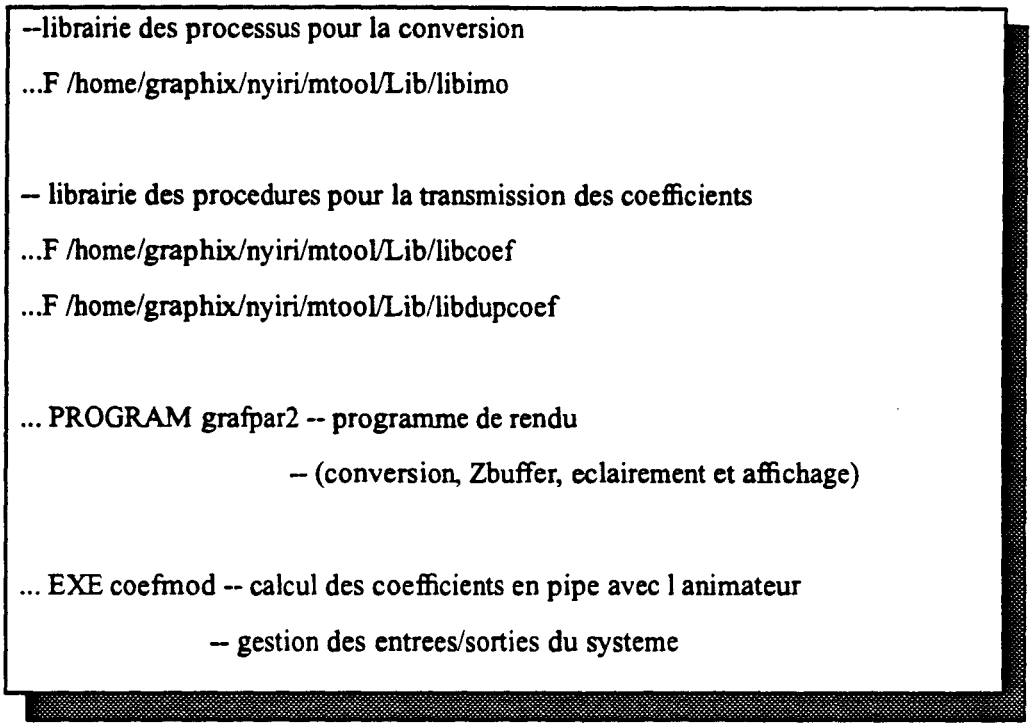


Figure B.1 : L'organisation générale du simulateur

Pour débuter la simulation, on configure du réseau et on envoie les processus définis dans le *PROGRAM grafpar2*, puis on exécute le programme *EXE coefmod*. Lorsque le programme d'animation envoie les données, le processus d'affichage commence : les coefficients des objets sont calculés, puis transmis au réseau pour le rendu.

B.1.2 L'organisation du Transputer de préparation

Le Transputer du nœud d'entrée du réseau effectue la préparation des données pour l'affichage des images et exécute certaines actions liées au traitement de la scène.

Le traitement de l'image

La construction d'une image commence par une définition initiale des données et des traitements à effectuer, puis par l'envoi des caractéristiques des sources lumineuses présentes dans la scène. Le programme traite et transmet ensuite pour la conversion en pixels, les objets de la scène puis les volumes de lumière et enfin les volumes d'ombres.

```
SEQ
  ... initialisation des données et des traitements
  ... traitement des sources
  ... traitement des objets
  ... traitement des volumes de lumiere
  ... traitement des ombres
write.full.string(screen,"Fin SCENE*c*n")
```

Figure B.2 : le traitement de l'image

Pour la conversion en pixels, nous avons défini des procédures (préfixées par SC pour la compilation séparée), qui calculent les coefficients suivant le type de l'objet à afficher (Figure B.3). Elles permettent l'utilisation des quadriques à trois plans de coupe, des facettes planes (triangulaires, triangulaires avec interpolation des normales aux sommets et quadrilatères) et des polyèdres à six faces pour le rendu, ainsi que l'emploi des ombres portées à partir de volumes d'ombre quadriques et polyèdriques. Les coefficients des volumes utilisés par les spots sont définis par la procédure de calcul pour les volumes d'ombre, puisque le traitement des volumes de lumière change seulement lors de la phase d'éclairement.

Lorsque le programme de préparation reçoit les données pour l'affichage d'un objet, il exécute la procédure correspondant au type de l'objet qui effectue automatiquement la transmission des coefficients au programme de rendu.

```
... SC quadrique
... SC quadriques ombres
... SC polyhedre et polyhedres ombres
... SC facettes triangulaires
... SC facette quadrilatere
... SC mise en perspective
```

Figure B.3 : les procédures de calcul des coefficients

Les données pour le Processus Objet Quadrique

La préparation des coefficients pour une quadrique, défini dans le module *SC quadrique*, comporte plusieurs parties (Figure B.4).

```
-- changement c positif
... changement de signe

-- definition des coeficients
... coefficients de z et normale
... definition coef plans
... determination face ou cote

IF
 deuxc = 0.0(REAL64)
 SEQ
 ... coef nul
 TRUE
 SEQ
 ... coef z quadrique + ou - racine

IF
 (p1c*mulm)> 16000000.0(REAL64) -- variation de 2 pixels
 SEQ
 ... coef planf perpendiculaire
 TRUE
 SEQ
 ... planf

IF
 (p2c*mulm)> 16000000.0(REAL64) -- variation de 2 pixels
 SEQ
 ... coef planb perpendiculaire
 TRUE
 SEQ
 ... planb

IF
 (p3c*mulm)> 16000000.0(REAL64) -- variation de 2 pixels
 SEQ
 ... coef plan3 perpendiculaire
 TRUE
 SEQ
 ... plan3

... coeffficient multiplicatif normales

... normales sur_quadrique
... normale planf
... normale planb
... normale plan3

... couleur
... type d Objet
... Normale en Sortie
```

Figure B.4 : Les coefficients de conversion d'une quadrique

La première action effectuée est de transformer l'équation de la quadrique pour rendre positif le coefficient c du terme en z^2 afin d'ordonner les deux profondeurs de la surface quadrique. Ensuite la procédure définit les coefficients pour le calcul des profondeurs et des normales à la surface quadrique et aux plans de coupe. Un test permet de déterminer ensuite l'orientation du volume (si l'objet est volumique). Les coefficients utilisés pour le calcul des profondeurs sont testés afin de déterminer si la

surface peut être définie aux pixels ou s'il faut la transformer (en une limite de demi-plans par exemple).

Tous les coefficients de profondeurs et de normales sont normalisés puis convertis en entier avant la transmission. Les dernières données envoyées au programme de rendu sont la couleur de l'objet, le type d'objet (volume ou surface) et la normale (avant ou arrière) qu'il faut transmettre en un pixel après la conversion.

La gestion des actions

Les communications entre l'animateur et le simulateur ne s'effectuant que dans un sens, seul le transputer du nœud d'entrée peut déterminer la fin d'un traitement (par exemple la fin du rendu d'une scène). L'animateur envoie donc des commandes au programme de préparation afin que ce dernier exécute soit une action d'entrée/sortie pour le réseau, soit une action particulière liée à l'animation (Figure B.5).

```
{{{ traiter commande
read.int(keyboard,action,ff)
IF
  (action=0)
    ... traiter image
  (action=3)
    ... enregistrer image sur scope
  (action=4)
    ... fin de simulation
  (action=5)
    ... init Scope
  (action=6)
    ... filtre image
  (action=7)
    ... changer duree enregistrement
  (action=9)
    ... sauvegarde image dans fichiers
  ((action=10) OR (action < 0))
    SKIP -- flush du fichier d'entree
  TRUE
    write.full.string(screen,"Erreur de commande*c*n")
}}}
```

Figure B.5 : la gestion des commandes

Les actions d'entrées/sorties pour le réseau sont l'affichage d'une image, la sauvegarde dans trois fichiers des couleurs rouges vertes et bleues de chaque pixels ou l'application d'un filtre sur l'image affichée.

Lors de la création d'une animation, il est possible d'enregistrer chaque image sur un magnétoscope. La gestion de l'enregistrement est effectuée par le programme de simulation.

La fin de la simulation est exécutée par l'envoi d'une commande sur le réseau de Transputers, chaque processus devant être achevé pour que le programme de rendu s'arrête.

B.1.3 L'organisation du rendu

Le rendu est effectué à l'aide de 17 Transputers (Figure B.6) sur lesquels sont répartis les processus de l'affichage (Figure B.7).

Le Transputer Maître est situé sur le premier Transputer du réseau et exécute le processus *imoD*. Il répartit les objets pour la conversion et gère les communications du système avec le nœud d'entrée. Il dispose d'une liaison avec le nœud d'entrée (pour la lecture des coefficients des objets), le premier Transputer de conversion (pour l'envoi des coefficients sur le pipe-line de répartition) et avec le Transputer d'affichage (pour la gestion de l'image finale).

Les neuf Transputers Objets sont regroupés par trois, afin de convertir un objet en pixels sur tout l'écran (celui-ci étant divisé en trois groupes de lignes entrelacées). Ils exécutent les processus *imoDP* (ou *imoDPF*), *imoIP* ou *imoFP* suivant les lignes écran qu'ils doivent gérer. L'ensemble de ces Transputers permet donc de convertir trois objets en parallèle.

Les trois Transputers exécutant les processus *zbufcol* effectuent l'élimination des parties cachées et la mémorisation des couleurs aux pixels lors de la conversion des objets et transmettent les normales aux processus d'éclairement. Les processus définissent ensuite les pixels qui sont éclairés ou à l'ombre pour l'application des spots lumineux et des ombres portées. Enfin ils transmettent les données de couleurs au processus d'éclairement.

Les trois processus *calstock* reçoivent et mémorisent lors de la conversion les normales aux pixels. Lorsque la conversion est terminée, ces trois processus calculent l'éclairement des pixels transmis par les processus zbufcol. Après éclairement, les couleurs des pixels sont envoyées au processus d'affichage.

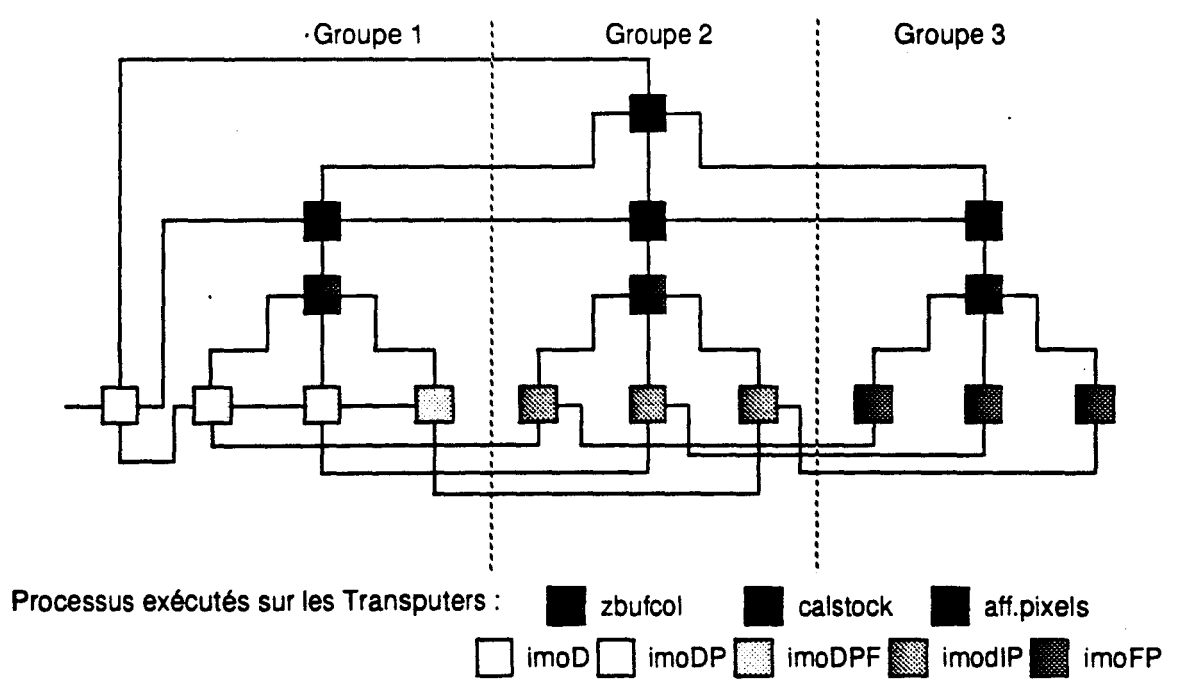


Figure B.6 : Les processus du rendu

```
... SC graf2 -- affichage pixels
... SC imoD -- repartition objets
... SC imoDP -- objets sur ligne 3L sur pipe repartition
... SC imoDPF -- objets sur ligne 3L en final du pipe repartition
... SC imoIP -- objets sur ligne 3L+1 sur pipe objet
... SC imoFP -- objets sur ligne 3L+2 en final sur pipe objet
... SC calstock -- calcul de l eclairement de pixel + stockage normale
... SC zbufcol -- calcul de zbuffer + test volume d ombre + stock couleur

VAL INT al IS 3: -- nombre de divisions de l'écran
VAL Mode IS 1 : -- 0 : TV ; 1 : MONITOR ; 2 : MONITOR BROADCAST
... declaration des liens physiques
[2]CHAN OF INT64 hosttop0,p0tograf: -- maitre et noeud d entree
[3][2][2]CHAN OF INT64 pitopp: -- sur pipe entre po objet
[3][2]CHAN OF INT64 pitopis: -- sur pipe repartition
[3][3]CHAN OF ANY ptozbuf: -- po vers zbuf
[3]CHAN OF ANY zbuftocal: -- entre zbuf et calstock
[3][2]CHAN OF INT caltogds: -- entre calstock et gds
PLACED PAR

-- Processeur Maitre

PROCESSOR 0 T8
PLACE hosttop0[0] AT link0.in:
PLACE hosttop0[1] AT link0.out:
PLACE pitopis[0][1] AT link2.in:
PLACE pitopis[0][0] AT link2.out:
PLACE p0tograf[1] AT link3.in:
PLACE p0tograf[0] AT link3.out:
imogD(hosttop0[0], hosttop0[1], p0tograf[1], p0tograf[0],
pitopis[0][1], pitopis[0][0],3,al)

-- Processeurs Objets sur Pipe Repartition

PROCESSOR 1 T8 .
PLACE pitopis[0][0] AT link0.in:
PLACE pitopis[0][1] AT link0.out:
PLACE pitopis[1][1] AT link1.in:
PLACE pitopis[1][0] AT link1.out:
PLACE pitopp[0][0][1] AT link2.in:
PLACE pitopp[0][0][0] AT link2.out:
PLACE ptozbuf[0][0] AT link3.out:
imogDP(pitopis[0][0],pitopis[0][1],pitopis[1][1],pitopis[1][0],
pitopp[0][0][1],pitopp[0][0][0],ptozbuf[0][0],1,al,0)
     .....

-- en Intermediaire sur pipe Objet

PROCESSOR 4 T8
PLACE pitopp[0][0][1] AT link0.out:
PLACE pitopp[0][0][0] AT link0.in:
PLACE pitopp[0][1][0] AT link2.out:
PLACE pitopp[0][1][1] AT link2.in:
PLACE ptozbuf[0][1] AT link3.out:
imogIP(pitopp[0][0][0],pitopp[0][0][1],
pitopp[0][1][1],pitopp[0][1][0],ptozbuf[0][1],
4,al,1)
```

```
.....
-- en Final sur pipe Objet

PROCESSOR 7 T8
PLACE pitopp[0][1][1] AT link0.out:
PLACE pitopp[0][1][0] AT link0.in:
PLACE ptozbuf[0][2] AT link3.out:
imogFP(pitopp[0][1][0],pitopp[0][1][1],ptozbuf[0][2],7,al,2)
.....
-- Processeurs pour ZBuffer et stockage Couleur

PROCESSOR 10 T8
PLACE ptozbuf[0][0] AT link1.in:
PLACE ptozbuf[1][0] AT link2.in:
PLACE ptozbuf[2][0] AT link3.in:
PLACE zbuftocal[0] AT link0.out:
zbufcol(ptozbuf[0][0],ptozbuf[1][0],ptozbuf[2][0],zbuftocal[0])
.....
-- Processeurs de calcul d eclairement et stockage normale

PROCESSOR 13 T8
PLACE zbuftocal[0] AT link2.in:
PLACE caltogds[0][0] AT link0.out:
PLACE caltogds[0][1] AT link0.in:
eclair.stock(caltogds[0][1],caltogds[0][0],zbuftocal[0],Mode,al,0)
.....
-- Processeur d affichage

PROCESSOR 1200 T8
PLACE caltogds[0][0] AT link0.in:
PLACE caltogds[0][1] AT link0.out:
PLACE caltogds[1][0] AT link1.in:
PLACE caltogds[1][1] AT link1.out:
PLACE caltogds[2][0] AT link2.in:
PLACE caltogds[2][1] AT link2.out:
PLACE p0tograf[0] AT link3.in:
PLACE p0tograf[1] AT link3.out:
aff.pixels (p0tograf[0],p0tograf[1],caltogds[0][0],
caltogds[1][0],caltogds[2][0],
caltogds[0][1],caltogds[1][1],caltogds[2][1],Mode)
```

Figure B.7 : Le réseau du rendu

B.2 La simulation des Processeurs Objets

Nous avons défini des processus élémentaires qui calculent des expressions du premier et du second degré et des procédures pour le calcul de la racine carrée. Ces processus ont ensuite été employés dans les processus objets, qui simulent le fonctionnement des processeurs de conversion.

B.2.1 Les processus élémentaires

Les processus élémentaires calculent des expressions du premier et du second degré de manière incrémentale. Un signal leur est envoyé par le processus objet qui les utilise à chaque début de conversion d'un objet, début de ligne, changement de pixels ou fin de traitement de l'objet.

Le processus élémentaire du premier degré

Un processus qui calcule une expression du premier degré fonctionne selon la méthode du "forward differencing" (Figure B.8). Ainsi lorsque le processus reçoit un signal de début de ligne ou de changement de pixels, il utilise la valeur précédente pour calculer la valeur de l'expression à la nouvelle ligne ou au nouveau pixel.

Ce processus utilise une procédure de masquage des valeurs afin d'exécuter des calculs sur un nombre de bits fixé à 24 (les processeurs élémentaires du premier degré utilisent 24 bits).

Le processus élémentaire du second degré

Un processus élémentaire du second degré utilise la même méthode de fonctionnement que le processus élémentaire du premier degré, pour évaluer en chaque pixel une expression du second degré (Figure B.9).

La procédure Mask2 permet d'effectuer les calculs sur 48 bits afin de simuler le processeur élémentaire du second degré.

```
PROC Pe1 (CHAN OF INT64 from.proc,CHAN OF INT64 to.proc)
{{{
INT64 maskA1,maskB1,maskC1 :
INT64 MaxB,MinB :
{{{ procedure de masquage
PROC Mask1 (INT64 value)
SEQ
value:=value/\maskA1 -- mask de la valeur
IF
((value/\maskB1)=0(INT64)) -- test signe
SKIP
TRUE
value:=value\/maskC1 -- si valeur negative propager le signe
:
}}}
INT64 d,e,f :
INT64 P,Q,V :
BOOL nfin:
SEQ
  nfin := TRUE
  maskA1:=#0000000000FFFFFF(INT64) -- mask sur 24 bits
  MaxB:=#00000000007FFFFF(INT64)
  MinB := (-MaxB)
  WHILE nfin
    INT64 com :
    SEQ
      from.proc ? com
      IF
        com=(INT64 0) -- debut d'objet
          SEQ
            maskC1:=~maskA1
            maskC1:=((maskC1)/2(INT64)) -- negatif sur maskA1 bits
            maskB1:=~(maskA1><maskC1) -- bit de signe
            from.proc ? d
            from.proc ? e
            from.proc ? f
            Q:=(f)
            Mask1(Q)
            P:=(d)
            Mask1(P)
            V:=Q
        com=(INT64 1) -- debut de ligne
          SEQ
            Q:=(Q PLUS e)
            Mask12(Q)
            V:=Q
        com=(INT64 2) -- pixel
          SEQ
            to.proc ! V
            V:=(V PLUS P)
            Mask12(V)
        com=(INT64 3) -- fin traitement
          SEQ
            nfin := FALSE
}}}
:
```

Figure B.8 : Le processus élémentaire du premier degré

```
PROC Pe2 (CHAN OF INT64 from.proc,CHAN OF INT64 to.proc)
{{{
INT64 maskA2,maskB2,maskC2 :
... procedure de masquage
INT64 a,b,c,d,e,f :
INT64 A,B,I,J,P,Q,V :
BOOL nfin :
SEQ
 nfin := TRUE
 maskA2:=#0000FFFFFFFFFFFF(INT64) -- mask sur 48 bits
 WHILE nfin
  INT64 com :
  SEQ
   from.proc ? com
   IF
    com=(INT64 0) -- debut d'objet
     SEQ
       ... calcul des masks
       from.proc ? a
       from.proc ? b
       from.proc ? c
       from.proc ? d
       from.proc ? e
       from.proc ? f
       A:=(a PLUS a)
       B:=(b PLUS b)
       I:=(e MINUS b)
       Mask2(I)
       Q:=(f)
       Mask2(Q)
       P:=(d MINUS a)
       Mask2(P)
       V:= Q
       J:= P
    com=(INT64 1) -- debut de ligne
      SEQ
        I:=(I PLUS B)
        Mask2(I)
        Q:=(Q PLUS I)
        Mask2(Q)
        P:=(P PLUS c)
        Mask2(P)
        J:=P
        V:=Q
    com=(INT64 2) -- pixel
      SEQ
        to.proc ! V
        J:=(J PLUS A)
        Mask2(J)
        V:=(V PLUS J)
        Mask2(V)
    com=(INT64 3) -- fin traitement
      SEQ
        nfin := FALSE
}}}
:
```

Figure B.9 : Le processus élémentaire du second degré

Les procédures de calcul des racines carrées

Suivant le nombre de bits significatifs que l'on veut obtenir sur le résultat de l'évaluation d'une racine carrée (24 ou 12), nous avons défini deux procédures, qui permettent de tester le nombre de bits utiles lors du rendu des scènes (Figure B.10).

```
PROC rac.a2(VAL INT64 valeur1,INT64 rac.app) -- racine carree sur 24 bits
 INT nb.bit :
 INT64 Mask :
 VAL INT bit.choix IS 24 :
 VAL INT64 mask IS #FFFFFF(INT64):
 SEQ
  IF
   valeur1 < 0(INT64)
     rac.app := 0(INT64)
   TRUE
    rac.app:= INT64 ROUND (DSQRT(REAL64 ROUND valeur1))
  {{{ mask
  long.bit64(rac.app,nb.bit)
  Mask := mask
  IF
   nb.bit > bit.choix
     SEQ
      Mask := Mask<<(nb.bit-bit.choix)
      rac.app := rac.app/\Mask
   TRUE
    SKIP
 }}}
:

PROC rac.a(VAL INT64 valeur1,INT64 rac.app) -- racine carree sur 12 bits
 INT nb.bit :
 INT64 Mask :
 VAL INT bit.choix IS 12 :
 VAL INT64 mask IS #FFF(INT64):
 SEQ
  IF
   valeur1 < 0(INT64)
     rac.app := 0(INT64)
   TRUE
    rac.app:= INT64 ROUND (DSQRT(REAL64 ROUND valeur1))
  {{{ mask
  long.bit64(rac.app,nb.bit)
  Mask := mask
  IF
   nb.bit > bit.choix
     SEQ
      Mask := Mask<<(nb.bit-bit.choix)
      rac.app := rac.app/\Mask
   TRUE
    SKIP
 }}}
:
```

Figure B.10 : Le calcul des racines carrées

B.2.2 La simulation des Processeurs Objets

Afin de pouvoir afficher des scènes contenant différents types d'objets, nous avons défini les méthodes qui simulent le fonctionnement des différents processeurs objets de conversion. Parmi celles-ci, nous retrouvons la simulation du Processeur Objet Quadrique que nous décrivons ensuite.

Les Processus Objets simulés

La simulation des Processeurs de conversion (Figure B.11) permet de démontrer la faisabilité de ceux-ci et d'étudier les qualités visuelles des primitives utilisées.

Dans le but de comparer le rendu en facettes et le rendu en quadriques, nous avons défini des processus de conversion des primitives facettes triangulaires (avec interpolation des normales au sommet), facettes quadrilatères et quadriques (volumiques ou surfaciques). Seule la seconde génération de Processeur Objet Quadrique est simulée, la première génération de P.O.Q. n'ayant plus d'intérêt. Le Processeur Objet Patch quadrique n'a pas non plus été simulé ici, puisque une autre personne de l'équipe a été chargée de son étude et que ce processeur n'est qu'une version simplifiée du P.O.Q.

Nous avons défini aussi un processus de conversion de polyèdre afin de tester les qualités de l'algorithme de coupe par des plans déjà utilisé pour la définition du P.O.Q.

Enfin nous avons défini des processus de conversion des volumes d'ombres quadriques et polyèdriques, afin de valider les concepts et les algorithmes utilisés pour l'application des ombres portées par la méthode de Crow. Ces processus de génération des volumes d'ombres ne calculent pas les normales aux surfaces et emploient donc une procédure de coupe par un plan simplifiée.

Tous les Processus Objets emploient pour la conversion les Processus Elémentaires du premier et/ou du second degré. Ceux-ci sont exécutés en parallèle grâce à la commande PAR (Figure B.11). Les communications entre les Processus Elémentaires et les Processus Objets s'effectuent par l'intermédiaire de canaux virtuels, qui simulent les liens physiques à l'intérieur des Processeurs Objets.

```
PROC fac.quad (CHAN OF INT64 from.host,CHAN OF ANY link.zbuf,
 INT ScrX,ScrY, VAL INT al, incl)
 SEQ
 ...
:
PROC fac.triang (CHAN OF INT64 from.host,CHAN OF ANY link.zbuf,
 INT ScrX,ScrY, VAL INT al, incl)
 SEQ
 ...
:
PROC quadri2 (CHAN OF INT64 from.host,CHAN OF ANY link.zbuf,
 INT ScrX,ScrY,BOOL clipping, VAL INT al,incl)
 ... declarations des variables
 ... liens avec les PE
 ... procedures de coupe par les plans
 PAR
 ... les Pe1 et Pe2 du processeur quadrique
 ... quadrique
:
PROC quadri.ombre (CHAN OF INT64 from.host,CHAN OF ANY link.zbuf,
 INT ScrX,ScrY,BOOL clipping, VAL INT al, incl)
 ... declarations des variables
 ... liens avec les PE
 ... procedures de coupe par les plans
 PAR
 ... les Pe1 et Pe2 du processeur quadrique
 ... quadrique
:
PROC polyhedre (CHAN OF INT64 from.host,CHAN OF ANY link.zbuf,
 INT ScrX,ScrY, VAL INT al,incl)
 ... declarations des variables
 ... liens avec les PE
 ... procedures de coupe par les plans
 PAR
 ... les Pe1
 ... constructeur
:
PROC polyhedre.ombre (CHAN OF INT64 from.host,CHAN OF ANY link.zbuf,
 INT ScrX,ScrY, VAL INT al,incl)
 ... declarations des variables
 ... liens avec les PE
 ... procedures de coupe par les plans
 PAR
 ... les Pe1
 ... constructeur
```

Figure B.11 : Le Processus Objet défini dans le simulateur

Le Processus Objet Quadrique

Le processus Objet de conversion des quadriques a été défini à partir du schéma fonctionnel du Processeur Objet Quadrique. Il comporte une procédure de coupe par un plan, qui reproduit le fonctionnement de l'algorithme utilisé pour les calculs de l'intersection du volume (ou de la surface) quadrique avec des demi-espaces.

- La procédure de coupe par un plan

```
PROC trait.plan(INT64 zf,nxf,nyf,nzf, zb,nxb,nyb,nzb, zp,tp,nxp,nyp,nzp, typeO,
BOOL snzf, snzb)
 VAL ZMAX IS #7FFFFFFF(INT64) :
 SEQ
  IF
    (tp = 0(INT64)) AND (zp<0(INT64)) -- plan perpendiculaire a l ecran
      SEQ
        zf := ZMAX
        zb := ZMAX
    (tp < 0(INT64)) AND (zp<=zf) -- plan avant devant quadrique
      SKIP
    (tp < 0(INT64)) AND (zb<zp) -- plan avant derriere quadrique
      SEQ
        zf := ZMAX
        zb := ZMAX
    (tp < 0(INT64)) AND (zf<=zp) -- plan avant interieur quadrique
      SEQ
        IF
          typeO<>3(INT64) -- volume
            SEQ
              zf := zp
              nxf := nxp
              nyf := nyp
              nzf := nzp
              snzf := TRUE
          TRUE -- surface
            SEQ
              zf := zb
              nxf := nxb
              nyf := nyb
              nzf := nzb
              snzf := snzb
    (tp > 0(INT64)) AND (zb <= zp) -- plan arriere derriere quadrique
      SKIP
    (tp > 0(INT64)) AND (zp < zf) -- plan arriere devant quadrique
      SEQ
        zf := ZMAX
        zb := ZMAX
    (tp > 0(INT64)) AND (zp <= zb) -- plan arriere interieur quadrique
      SEQ
        IF
          typeO<>3(INT64) -- volume
            SEQ
              zb := zp
              nxb := nxp
              nyb := nyp
              nzb := nzp
              snzb := FALSE
          TRUE -- surface
            SEQ
              zb := zf
              nxb := nxf
              nyb := nyf
              nzb := nzf
              snzb := snzf
    TRUE
      SKIP
:
```

Figure B.12 : La procédure de coupe par un plan

La procédure admet en entrée la profondeur avant et arrière de l'objet et la profondeur et le type du plan délimitant le demi-espace. Elle prend aussi comme paramètres les normales de l'objet et la normale du plan. Après l'exécution de l'algorithme, la procédure renvoie les nouvelles valeurs de profondeurs et de normales de l'objet coupé par l'intermédiaire des variables d'entrée (Figure B.12).

- Le traitement de la quadrique

Afin de simuler le déroulement d'une conversion en "scan-line", la conversion d'une quadrique s'effectue à l'aide de deux boucles imbriquées, la première représentant les changements de lignes et la seconde les changements de pixels (Figure B.13).

```
{{{ traiter quad
... coef quadrique
IF
  ((YMAX>0)AND(XMAX>0))AND(YMIN<ScrY)
    SEQ i = Ydeb FOR Nbl
      SEQ
        {{{ traiter ligne
        SEQ j = XMIN FOR (XMAX-XMIN)
          SEQ
            ... Pixel et Lecture des Pes
            ... definition des valeurs
            IF
              Vzs<0(INT64) --pixels hors contour quadrique
                SEQ
                  Zqf := ZMAX
                  Zqb := ZMAX
              TRUE
                SKIP
            ... coupe par plans
            ... conversion signe des normales
            ... test zavant en sortie --Zbuffer
            ... test zarriere en sortie -- Zbuffer
            ... normale en sortie
            IF
              (Zfol<>ZMAX) -- envoi des valeurs utiles
                SEQ
                  link.zbuf ! (INT32 j);(INT32 i);(INT32 Zfol);(INT32 Zbol) (INT32
Nxfol);(INT32 Nyfol);(INT32 Nzfol);(INT32 signNz
              TRUE
                SKIP
        }}}
        SEQ t=0 FOR al
          SEQ
            ... debut ligne pour PEs
    TRUE
      SKIP
... fin objet pour PEs
}}}
```

Figure B.13 : Le Processus Objet Quadrique

Le processus de conversion commence par la lecture des coefficients de la quadrique (et autres données nécessaires à l'affichage), qui sont transmis aux Processus Elémentaires utilisés. Lors de chaque changement de ligne ou de pixel, le processus envoie aux Processus Elémentaires le signal correspondant.

En chaque pixel, les valeurs des expressions du premier et du second degré sont transmises par les PE. Après l'appel de la procédure d'évaluation de la racine carrée, le processus détermine les valeurs de profondeurs de la quadrique et des plans de coupe. Il effectue alors le test sur le contour de la quadrique, puis exécute la procédure de coupe par un plan pour chaque plan de coupe (et pour chaque demi-objet si la quadrique est volumique). Finalement, si la quadrique est présente au pixel traité (afin de limiter les communications), le processus transmet au processus Z-buffer les valeurs de profondeurs, les normales et autres données utilisées pour le rendu.

Le simulateur nous a permis l'étude des Processeurs Objets Quadrique. Nous avons également pu comparer le rendu en facettes et en quadriques et tester les divers algorithmes que nous avons définis. Bien que notre étude ait été axée sur les processeurs objets, la réalisation d'un programme de rendu nous a permis de mieux comprendre les problèmes rencontrés lors de l'utilisation du parallélisme dans les machines d'affichage.

La définition du simulateur nous a aussi fait découvrir les joies de la programmation en OCCAM et la simplicité de gestion des communications entre processus parallèles. Enfin, nous avons apprécié l'environnement de programmation Mtool et tout particulièrement le debugger de programme parallèle.

Annexe C
Utilisation du Programme d'Animation

La réalisation d'un programme de rendu nous a conduit à définir un programme d'animation, afin de construire rapidement et simplement des images et des animations. Cette annexe contient une description de l'utilisation de cet animateur par l'intermédiaire de son interface.

C.1 L'animateur

L'animateur que nous avons développé permet de construire facilement des images et des séquences simples. L'interface comporte toutes les informations sur la scène et sur l'animation en cours de réalisation. Les entrées/sorties permettent de mémoriser une scène ou une animation pour une utilisation ultérieure. Le programme d'animation communique avec le simulateur que nous avons développé par l'intermédiaire d'un socket unix, permettant ainsi l'exécution des deux programmes sur des machines différentes. Lorsque l'on désire afficher une image ou une séquence d'images, l'animateur transmet les données au programme de rendu, qui peut alors effectuer la visualisation.

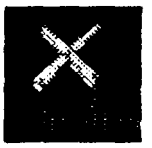
C.1.1 Description générale de l'interface

L'interface du programme d'animation (Figure C.1) peut être divisée en deux parties, la première comportant la gestion et la création de la scène et la seconde définissant une structure pour la gestion de la séquence.

La gestion de la scène est effectuée à l'aide de trois listes représentant les objets, les caméras et les sources. Dans chacune de ces listes, le nom sélectionné permet de déterminer l'objet, la caméra ou la source lumineuse qui sera manipulé dans la séquence. Deux boutons associés à la liste des objets permettent de se déplacer dans la hiérarchie des objets. La scène est créée à partir des définitions successives de tous ses éléments à l'aide du bouton *Creer*.

La séquence est gérée à l'aide de trois listes qui représentent les images, les objets, caméras et sources manipulés et les actions menées. La sélection d'un nom dans une des listes permet de se positionner à un endroit donné de l'animation pour y effectuer une modification. L'exécution de l'animation est contrôlée par un bouton *Séquence*, qui permet aussi l'accès aux paramètres de l'affichage.

Les messages concernant le déroulement du programme sont affichés en bas et à gauche de la fenêtre de l'interface. Enfin le bouton *Fichier* est utilisé pour le chargement ou la sauvegarde des scènes et des séquences.



L'animateur iconifié

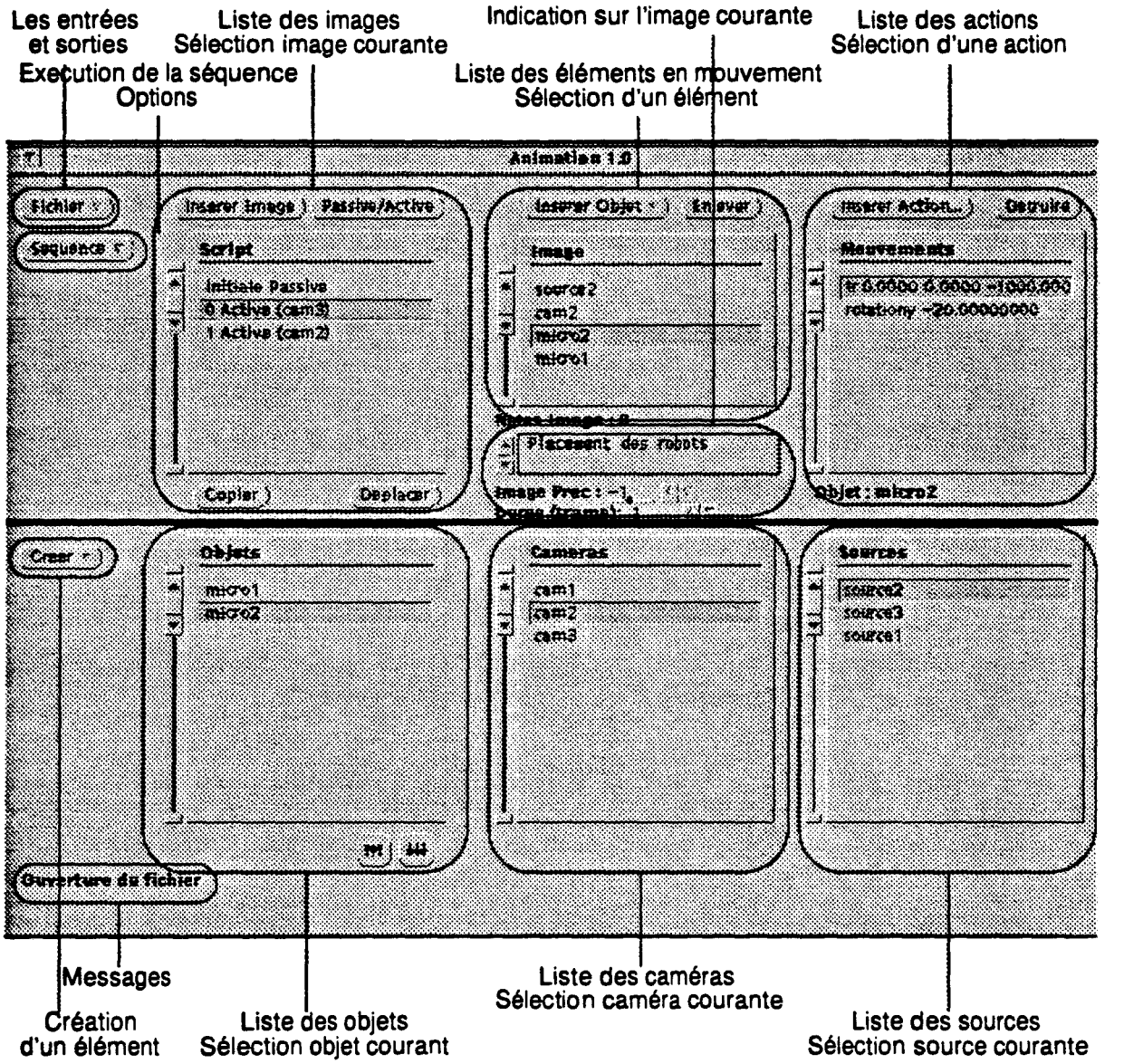


Figure C.1 : L'interface de l'animateur

C.1.2 Les entrées/sorties

Les actions d'entrée/sorties sont effectuées par l'appui sur le bouton *Fichier* (Figure C.2).

L'action *Charger* permet la lecture d'un fichier comportant soit une description des éléments de la scène, soit la définition d'une animation sous la forme d'une séquence d'images.

Nous avons séparé la sauvegarde des éléments de la scène de la sauvegarde de la séquence afin de

permettre l'application de plusieurs animations sur une même scène. L'action *Sauver Script* permet de mémoriser les différentes images de la séquence et l'action *Sauver Objets* enregistre les objets, caméras et sources d'une scène ainsi que le positionnement initial de ceux-ci.

L'action *Nouveau Script* efface la séquence en cours afin d'en redéfinir une autre, soit à l'aide de l'interface, soit par le chargement d'un fichier.

La destruction de la séquence et de la scène est effectué par *Nouvelle Anim*, une nouvelle animation peut alors être définie.

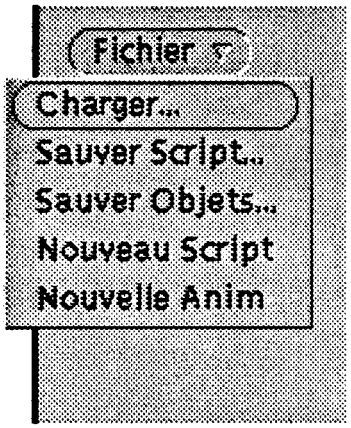


Figure C.2 : Les entrées/sorties

Les sauvegardes des scènes et des animations sont définies dans un fichier, sous la forme d'un ensemble de commandes suivies de données.

Les commandes sont de deux types :

- la création des éléments de l'animation, qui est préfixée par le mot clé *set*. Un élément de la scène est défini à partir de la donnée de son nom et de son type. suivie de données numériques qui dépendent de la création en cours. Une image de la séquence est construite par la donnée de l'image précédente dans la liste des images et dans le déroulement de la séquence. La caméra associée à une image doit être sélectionnée avant la création d'une image (*make nom select*).
- la manipulation des éléments de la scène, qui est préfixée par le mot clé *make*. La sélection de l'objet manipulé est effectuée par son nom et la nature de l'action définie ensuite, détermine les données nécessaires. Il est possible de définir une translation, une rotation autour d'un des axes ou un changement d'échelle sur tous les éléments de la scène. Les autres actions ne sont applicables que sur certains types d'objets particuliers. Une remarque importante : les actions sont attachées à l'image courante qu'il faut sélectionner au préalable (*make image n° select*), l'image courante par défaut étant l'image n° -1, qui définit le placement initial des éléments de la scène.

C.2 La création d'une scène

Une scène est créée à l'aide du bouton *Creer* (Figure C.3) par les définitions successives de ses objets, caméras et sources. Il faut noter que la définition d'une image oblige la création d'au moins une caméra et que l'affichage nécessite au moins la présence d'une source dans la scène.

Le type d'élément est défini par la sélection de *Objet*, *Source* ou *Camera*, ce qui permet d'ouvrir la fenêtre de création voulue.

Lors de la création, le repère local de l'élément est positionné sur le repère global de la scène.

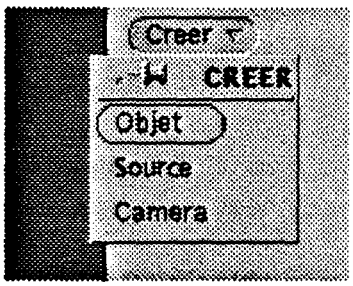


Figure C.3 La création d'une scène

C.2.1 La création d'un objet

La fenêtre de création d'un objet comporte trois parties (Figure C.4) :

- le champ *Nom* défini un nom d'objet, qui doit être différent pour chaque élément de la scène.
- une liste des différents objets géométriques pouvant être définis. La sélection d'un type d'objets ouvre alors la fenêtre de définition des paramètres nécessaires.
- le champ *Couleur* qui comporte le numéro de matière de l'objet. Actuellement les matières sont prédéfinies.

L'appui sur le bouton *Creer* de la fenêtre permet la création effective de l'objet.

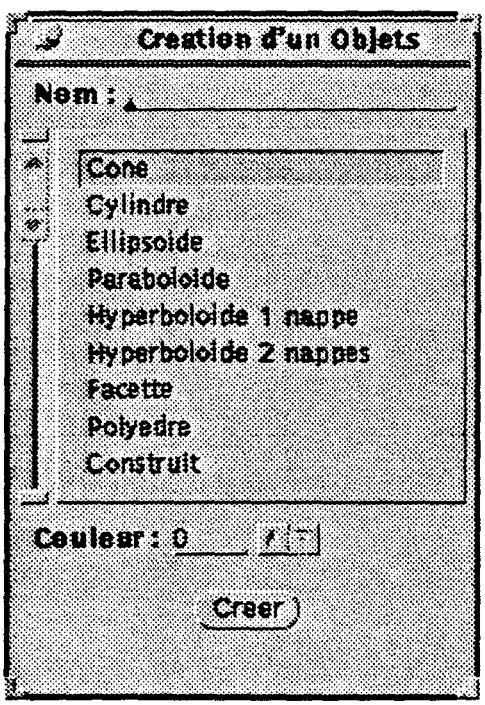


Figure C.4 : La création d'un objet

La création d'une quadrique

La sélection d'un type de quadrique dans la liste de création d'objets ouvre automatiquement la fenêtre de création d'un objet quadrique (Figure C.5).

Cette fenêtre permet de définir les paramètres de la surface de l'objet (*Rx*, *Ry* et *Rat*), le repère de l'objet étant le repère canonique de la quadrique. Le nombre de plans de coupe varie entre deux et trois, chaque plan de coupe étant déterminé par la donnée d'un point de sa surface (*X*, *Y*, *Z*) et par sa normale (*Nx*, *Ny*, *Nz*); le demi-plan utilisé pour la construction de l'objet se situe du côté opposé à la normale au plan.

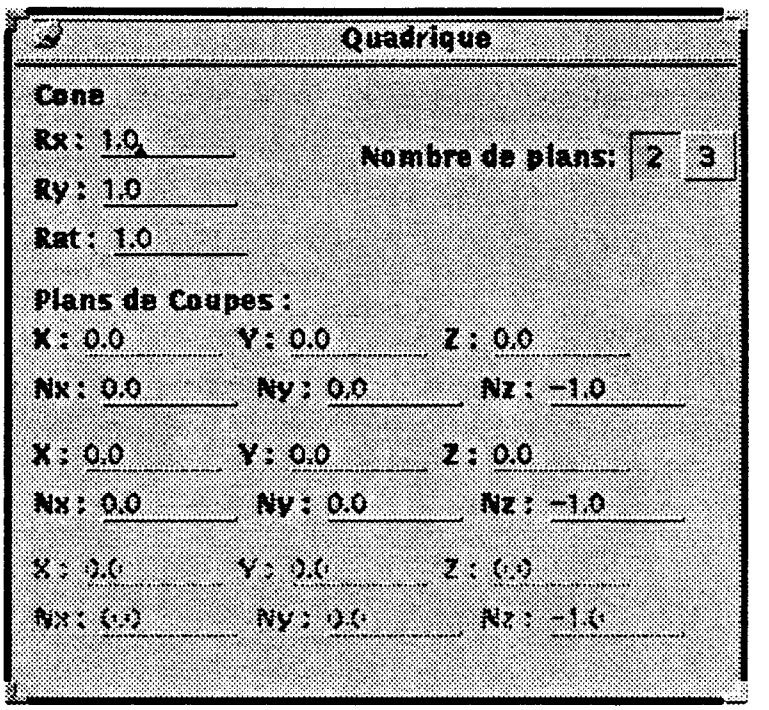
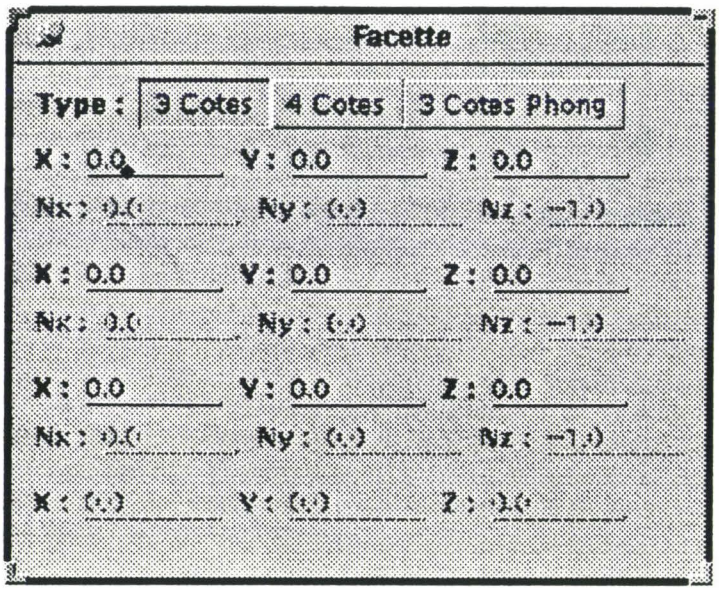


Figure C.5 : Paramètres pour la création d'une quadrique

La création d'une facette

La fenêtre de création d'une facette (Figure C.6) permet de choisir le type de facette que l'on veut définir. Si l'on choisit une facette à *3 Cotes* (ou à *4 Cotes*), les points (*X*, *Y*, *Z*) sont les 3 (ou 4) sommets. Pour la création d'une facette triangulaire avec définition des normales (*Nx*, *Ny*, *Nz*) en chacun des 3 sommets (*X*, *Y*, *Z*), il faut utiliser le type *3 Cotes Phong*.

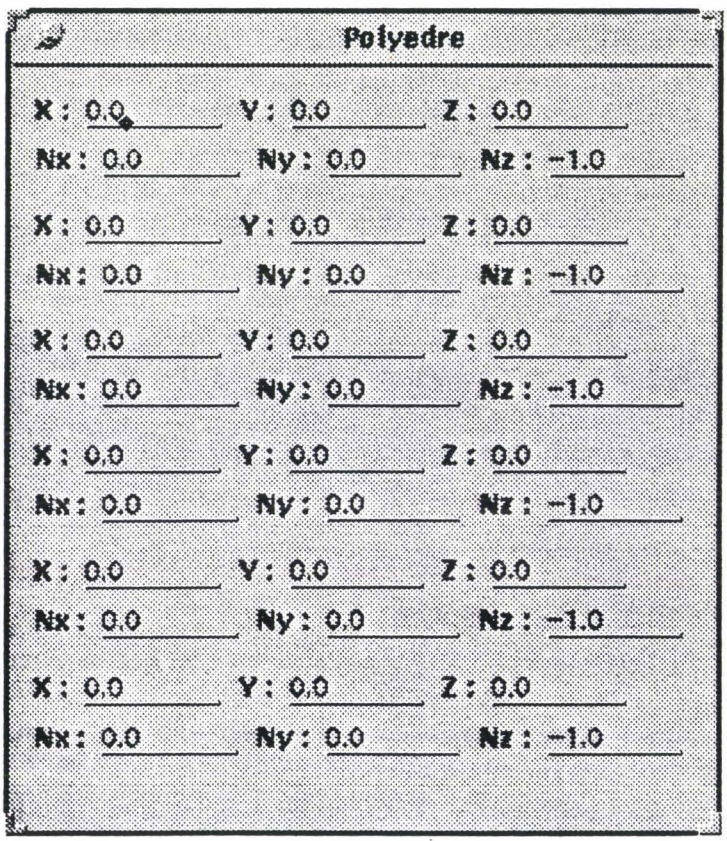


X : 0.0 Y : 0.0 Z : 0.0
Nx : 0.0 Ny : 0.0 Nz : -1.0
X : 0.0 Y : 0.0 Z : 0.0
Nx : 0.0 Ny : 0.0 Nz : -1.0
X : 0.0 Y : 0.0 Z : 0.0
Nx : 0.0 Ny : 0.0 Nz : -1.0
X : 0.0 Y : 0.0 Z : 0.0

Figure C.6 : Les paramètres des facettes

La création d'un polyèdre

La création d'un polyèdre à 6 faces nécessite de définir 6 demi-espaces définis à partir de 6 plans (Figure C.7). Comme pour les demi-espaces de construction des objets quadriques, un plan frontière est déterminé par la donnée d'un point de sa surface (*X*, *Y*, *Z*) et par sa normale (*Nx*, *Ny*, *Nz*)



X : 0.0 Y : 0.0 Z : 0.0
Nx : 0.0 Ny : 0.0 Nz : -1.0
X : 0.0 Y : 0.0 Z : 0.0
Nx : 0.0 Ny : 0.0 Nz : -1.0
X : 0.0 Y : 0.0 Z : 0.0
Nx : 0.0 Ny : 0.0 Nz : -1.0
X : 0.0 Y : 0.0 Z : 0.0
Nx : 0.0 Ny : 0.0 Nz : -1.0
X : 0.0 Y : 0.0 Z : 0.0
Nx : 0.0 Ny : 0.0 Nz : -1.0
X : 0.0 Y : 0.0 Z : 0.0
Nx : 0.0 Ny : 0.0 Nz : -1.0

Figure C.7 : Les paramètres du polyèdre à 6 faces

La création d'un objets construit

Un objet construit (ou complexe) étant un groupe d'objets, sa définition nécessite que les objets le composant soient créés avant. La construction de l'objet complexe consiste alors à sélectionner les objets du groupe dans la liste des objets de la scène déjà créés (Figure C.8).

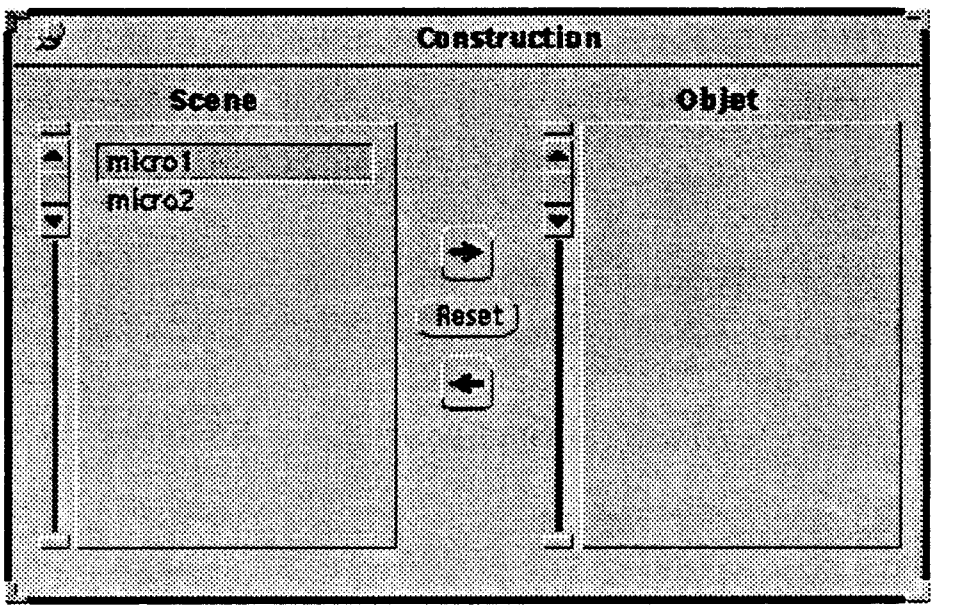


Figure C.8 : Définition d'un objet construit

C.2.2 La création d'une source lumineuse

La création d'une source est effectuée par l'intermédiaire d'une fenêtre (Figure C.9), qui permet de définir tous les éléments nécessaires. Comme pour un objet, une source est connue par son nom qui doit être différents de ceux des autres éléments de la scène. L'intensité d'une source dans chacune des couleurs de base (rouge, verte et bleue) est définie par un entier compris entre 0 et 255.

Une source *Directionnelle* est orienté suivant l'axe $\vec{z}$ de son repère local et la position initiale d'une source *Ponctuelle* est l'origine du repère. Un spot (type *Volumique*) est défini par une source ponctuelle placée à l'origine du repère local et par un volume quadrique, dont les paramètres sont *Rx*, *Ry* et *Rat* pour la surface et le plan d'équation $-z = 0$ pour la limite du demi-espace. Le type de la surface quadrique peut être choisi parmi le *cône*, le *cylindre*, le *paraboloïde* et l'*hyperboloïde* à une nappe.

Le bouton *Creer Source* permet la création effective de la source dans la structure des éléments de la scène.

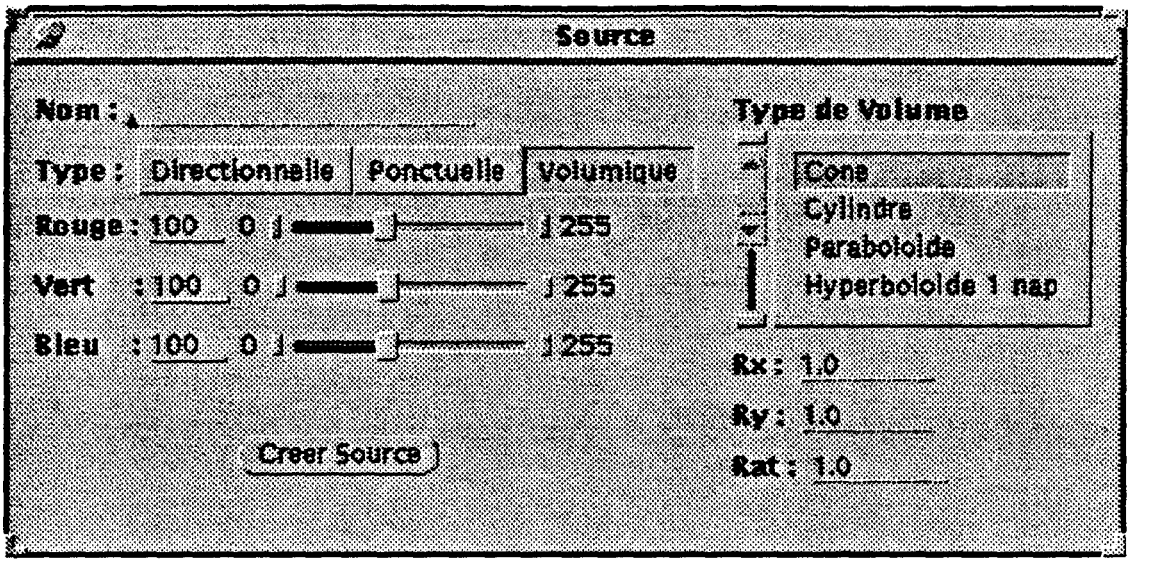


Figure C.9 : La création d'une source lumineuse

C.2.3 La création d'une caméra

Une caméra est définie par son nom et par sa focale, qui représente la distance entre l'observateur et l'écran de projection pour la transformation perspective appliquée à la visualisation. La focale est comprise entre 200 et 10000, puisqu'elle doit être strictement supérieure à 0 et parce que 10000 est une valeur suffisante. Une caméra est orientée suivant l'axe $\vec{z}$ de son repère local lors de sa création par *Creer Camera*.

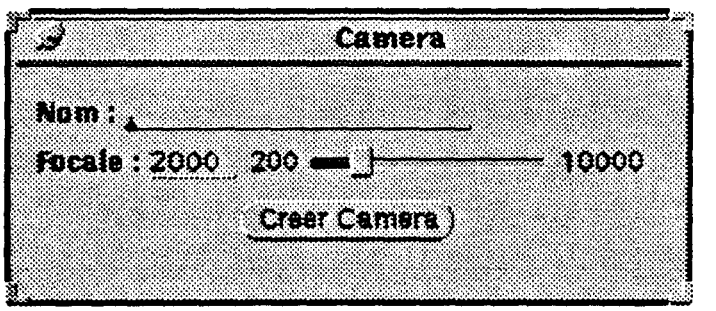


Figure C.10 : La création d'une caméra

C.3 La génération d'une séquence

La définition d'une séquence est réalisée par la création de toutes les images une par une. Une image est définie par l'ensemble des éléments de la scène qui subissent une modification par rapport à une image précédente. Afin de permettre une construction par images clés, une image peut être insérée à un endroit donné dans la liste des images. Il est alors nécessaire de définir pour chaque image, l'image la précédant dans la construction de l'animation, qui peut être différente de l'image précédente dans le déroulement de la séquence.

C.3.1 La création d'une image

La manipulation d'une image s'effectue à l'aide de trois actions (Figure C.11):

- *Inserer Image.* Cette action permet de créer une nouvelle image en sélectionnant l'image précédente dans le déroulement de la séquence. L'image précédente dans la création de l'animation est l'image précédente dans la liste, mais si l'on désire changer le numéro de celle-ci, on utilise ensuite le champ *Image Prec.*

- *Copier.* Lorsque deux images comportent les mêmes modifications d'objets, on peut dupliquer l'une des deux pour créer la seconde. Il suffit pour cela de sélectionner l'image en question, d'appuyer sur *Copier* puis de sélectionner l'image précédente pour l'insertion dans la liste : l'image obtenue a les même caractéristiques que l'image source. L'action de copie permet aussi une simplification de la création des images, lorsque deux images sont peu différentes au niveau de la modification des objets.

- *Deplacer.* Si une image a été insérée à une mauvaise position, il est possible de la déplacer à un autre endroit de la liste. Il suffit de sélectionner l'image à déplacer, d'appuyer sur *Deplacer*, puis de sélectionner la nouvelle image précédente dans la liste. Le déplacement ne modifie pas les caractéristiques de l'image.

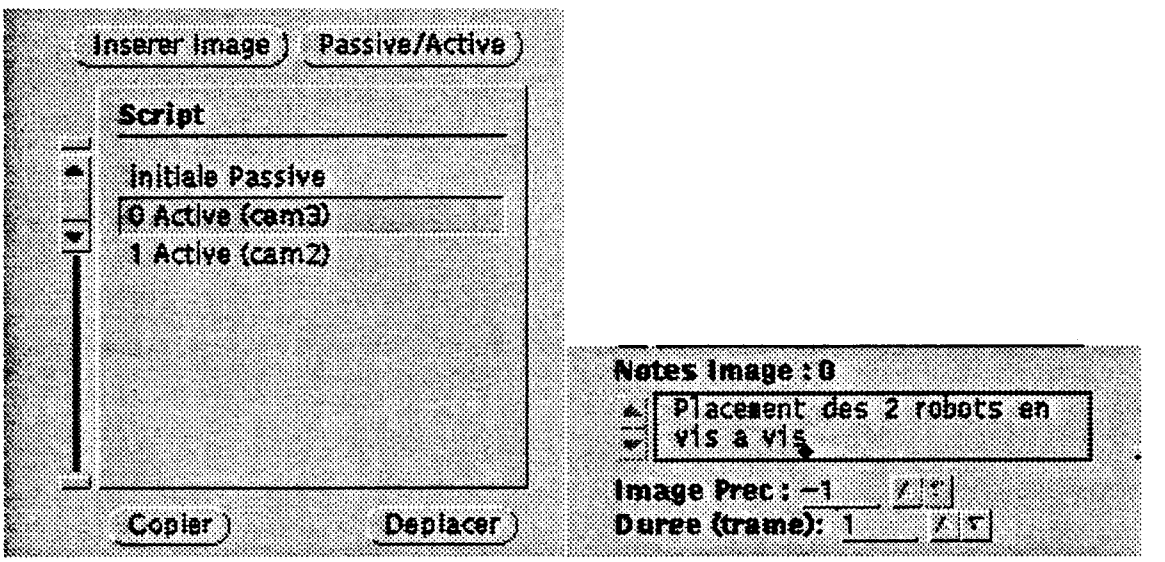


Figure C.11 : La manipulation des images

Par l'intermédiaire du bouton *Passive/Active*, on désactive (ou on réactive) une image de la séquence afin qu'elle n'apparaisse pas (ou qu'elle apparaisse à nouveau) dans l'animation.

Afin d'indiquer le contenu d'une image, on utilise le champ Notes Images. Ce champ permet lors d'une utilisation ultérieure de la séquence de connaître facilement le déroulement de l'animation.

Enfin, nous avons ajouté le champs *Duree*, qui permet de déterminer la durée d'une image lors d'un enregistrement de l'animation.

C.3.2 La désignation d'un élément en mouvement

A chaque image est associée une liste des éléments de la scène qui sont modifiés (Figure C.12). Pour insérer un nouvel élément dans la liste, on utilise *Inserer Objet*, qui permet de choisir l'élément parmi

l'objet courant, la caméra courante ou la source courante sélectionnés dans les listes de gestion de la scène. L'appui sur le bouton *Enlever* permet de supprimer un élément de cette liste, afin de ne plus le modifier.

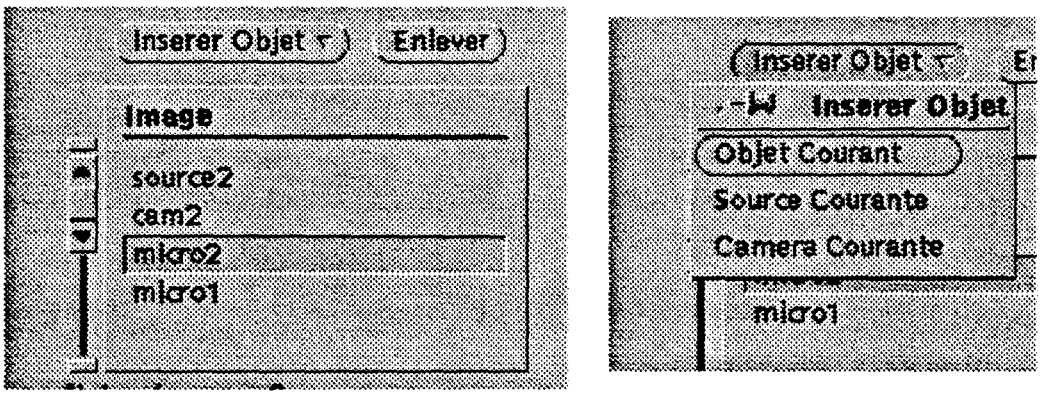


Figure C.12 : les éléments modifiés pour une image

C.3.3 La définition d'une action

A chaque objet modifié dans une image, on associe une liste des actions effectuées (Figure C.13). Lors de la définition d'une nouvelle action (*Inserer Action*), celle-ci est insérée après la sélection dans la liste. Pour supprimer une action, il suffit de la sélectionner puis d'appuyer sur le bouton de destruction *Detruire*.

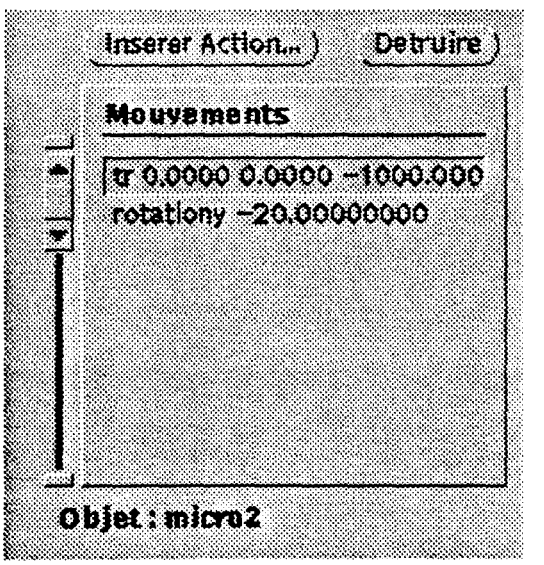


Figure C.13 : les modification sur un objet

Suivant l'élément de la scène que l'on veut modifier, il est possible d'effectuer des actions différentes dans la fenêtre de création d'une action (Figure C.14).

Les actions de translation (*Translation*), de rotation autour d'un des axes (*RotationX*, *RotationY*, *RotationZ*) et de changement d'échelle (*Echelle*) sont définies pour tous les éléments, puisqu'elles agissent sur le repère local de l'élément sélectionné.

Ensuite suivant l'élément, d'autres actions sont possibles :

- Il est possible de changer le statut de visibilité d'un objet en appuyant sur l'un des deux boutons *Montrer* ou *Cacher.*
- La modification des intensités d'une source est effectuée en indiquant les nouvelles valeurs pour chacune des couleurs rouge, verte et bleue, puis en appuyant sur *Intensité.*
- La seule action spécifique à une caméra est le changement de sa focale par *Changer Focale.*

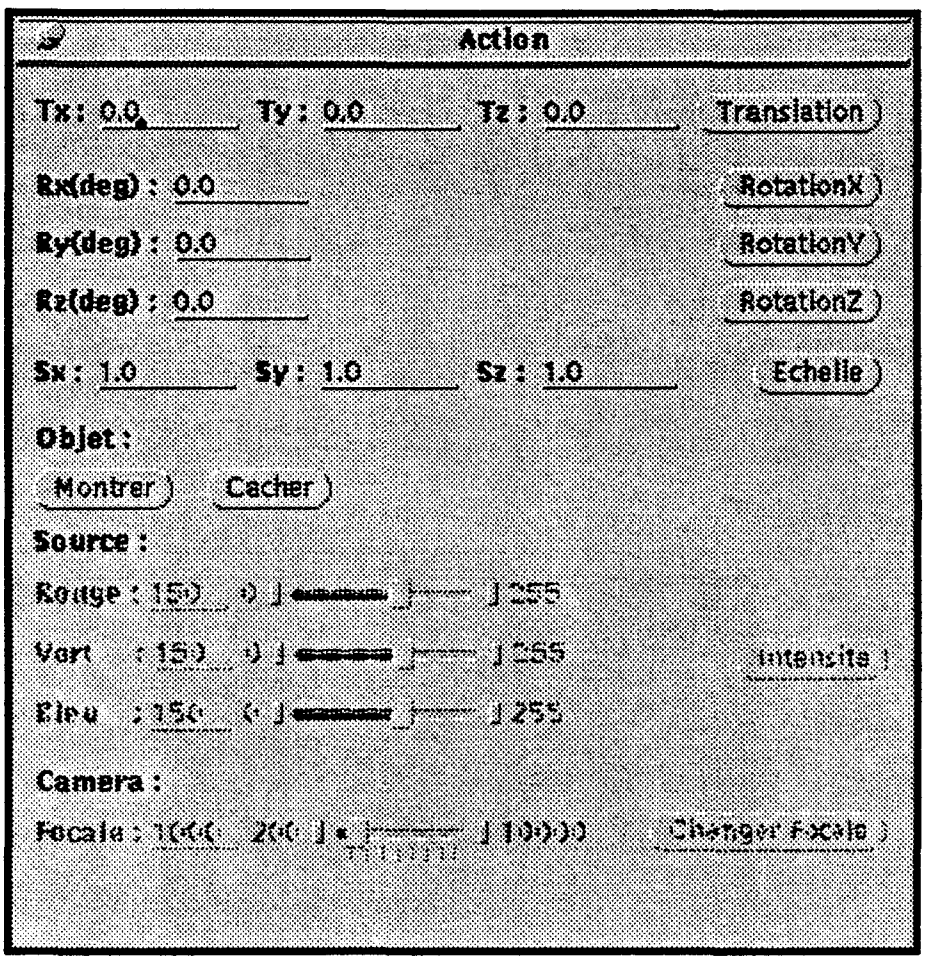


Figure C.14 : Définition d'une action sur un objet

C.4 La réalisation d'une animation

La scène et la séquence étant définies, l'animation peut être réalisée avec différentes options, qui définissent l'environnement de l'affichage.

C.4.1 L'exécution de la séquence

L'animation réalisée en affichant successivement toutes les images de la séquence. Le choix est laissé à l'utilisateur d'afficher une image à la fois (*Image par Image*) permettant ainsi de contrôler chaque image, ou d'afficher toute la séquence (*Animation*) (Figure C.15). La réalisation de l'animation commence à l'image sélectionnée dans la liste des images, puis se poursuit jusqu'au retour à l'image initiale.

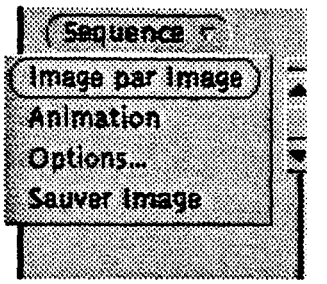


Figure C.15 : la réalisation de l'animation

La sélection de *Options* ouvre une fenêtre contenant les paramètres d'affichage. L'action *Sauver Image* transmet l'ordre au programme de rendu de sauver l'image actuellement affichée dans un fichier.

C.4.2 Les options

Les options d'affichage (Figure C.16) la définition de la couleur du fond d'écran par un numéro (*Couleur du fond*), l'activation ou la désactivation de l'application des ombres portées (*Ombres*), ainsi que le numéro de l'image à afficher (*Numero de l'image*). Nous avons ajouté la possibilité qu'une séquence soit simplement affichée sur un écran (*Afficher*) ou soit enregistrée sur un magnétoscope (*Enregistrer*).

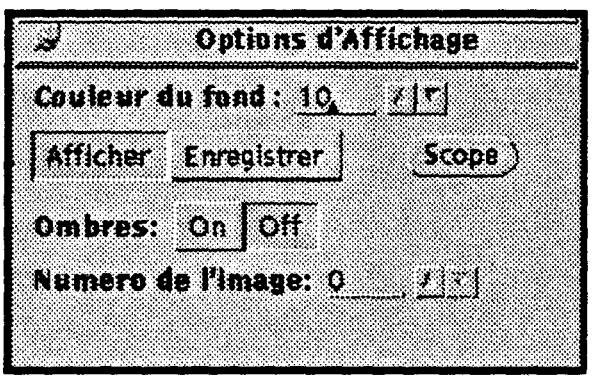


Figure C.16 : les options de l'affichage

L'appui sur le bouton *Scope* fait apparaître une fenêtre (Figure C.17) permettant de définir le point de début de l'enregistrement de l'animation sur le magnétoscope.

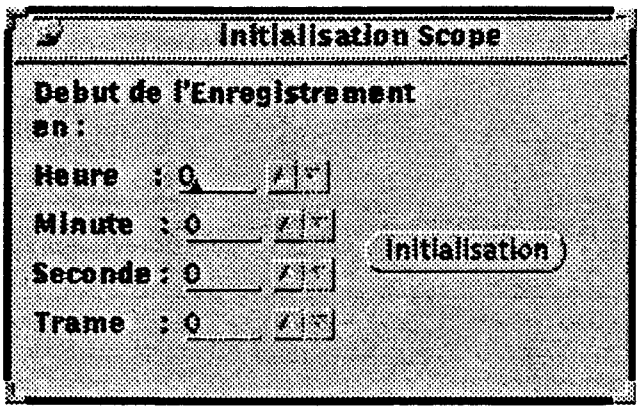
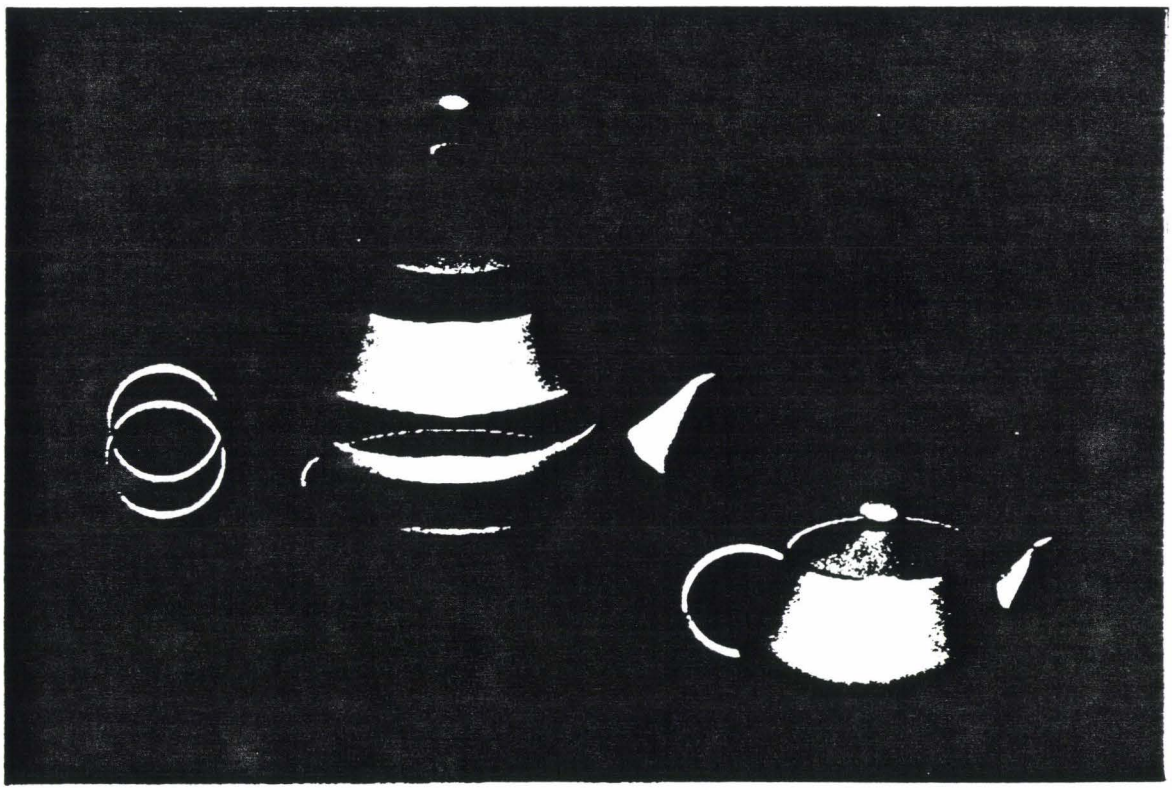


Figure C.17 : L'initialisation du magnétoscope

Grâce à l'animateur décrit ci-dessus, nous avons pu réaliser un grand nombre d'images afin d'effectuer des essais sur le programme de rendu que nous avons défini. La définition des objets à l'aide de l'animateur étant assez difficile, nous avons ressenti l'utilité d'un véritable programme de modélisation à base de quadriques.

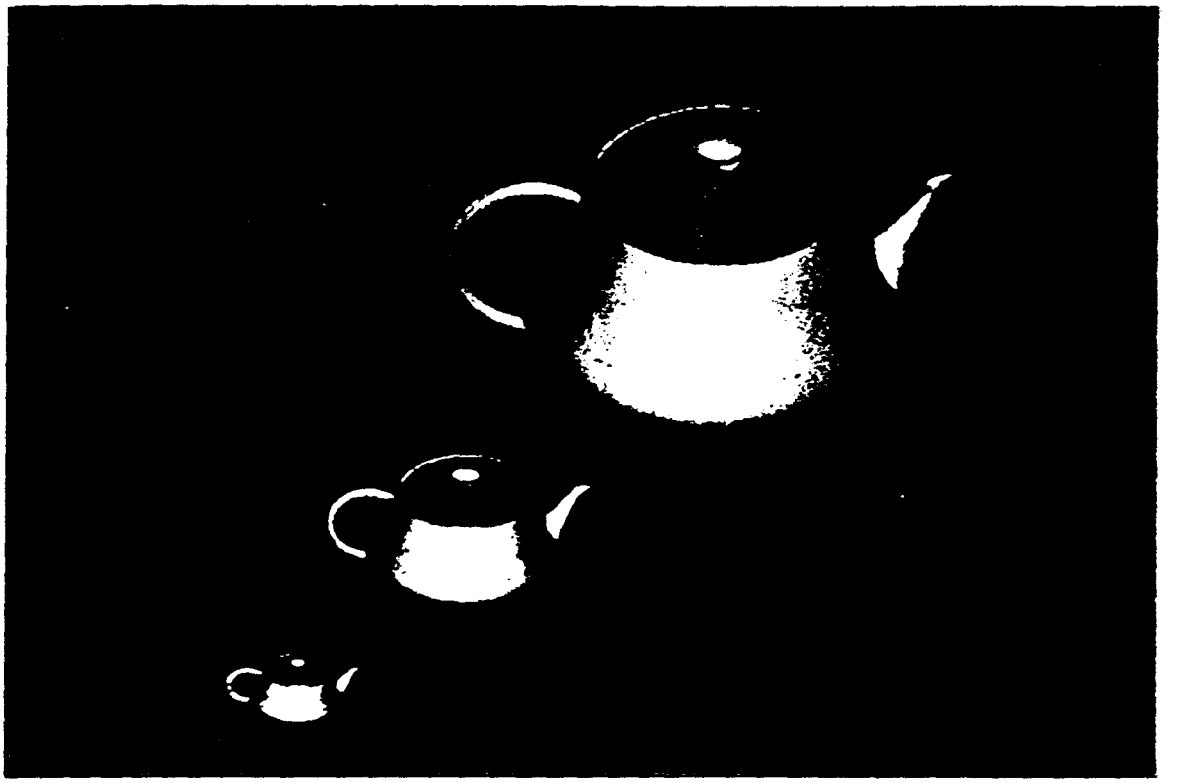
Annexe D
Recueil d'Images

Cette annexe contient quelques images réalisées à l'aide du programme d'animation et du simulateur. Ces images démontrent les qualités visuelles de la primitive quadrique et le potentiel d'une machine de rendu utilisant cette primitive. Les différentes scènes ont été définies à l'aide de l'animateur, donc sans véritable modeleur, ce qui explique que l'assemblage de surfaces quadriques ne soit pas effectué en continuité de tangence. Enfin, le rendu a été effectué avec application de la perspective mais sans anti-aliassage.

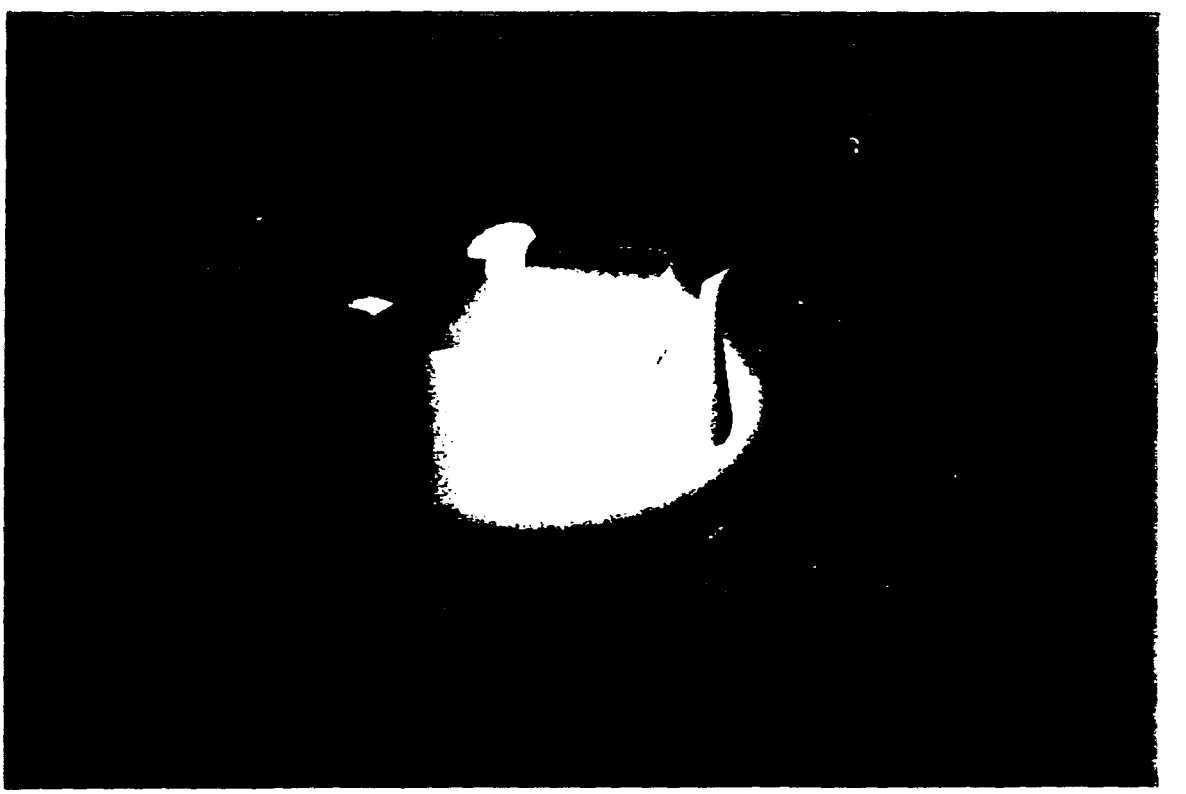


La théière (teapot) a été construite à l'aide de neuf quadriques surfaciques (quatre ellipsoïdes, deux cônes et trois hyperboloïdes à une nappe) et une quadrique volumique (le fond de l'objet). Cette image montre les possibilité offertes par la primitive quadrique.

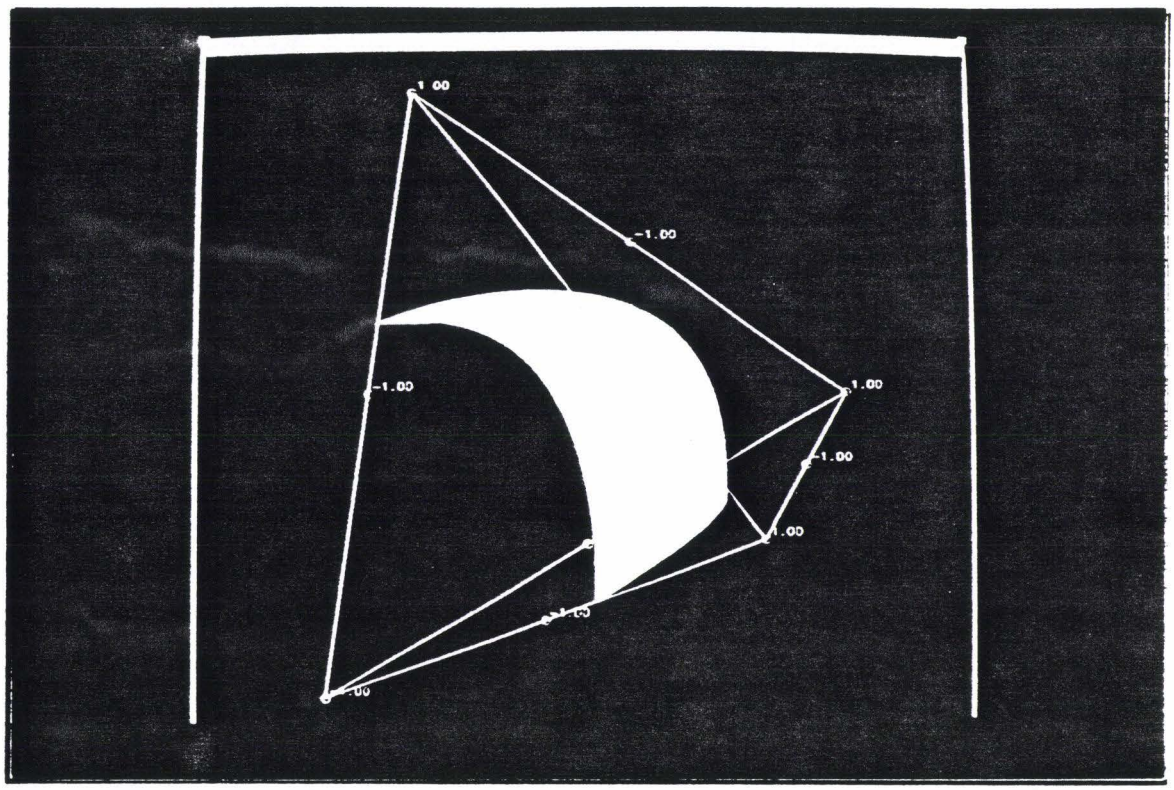
Nous remercions Sylvain Karpf qui a défini cette théière.



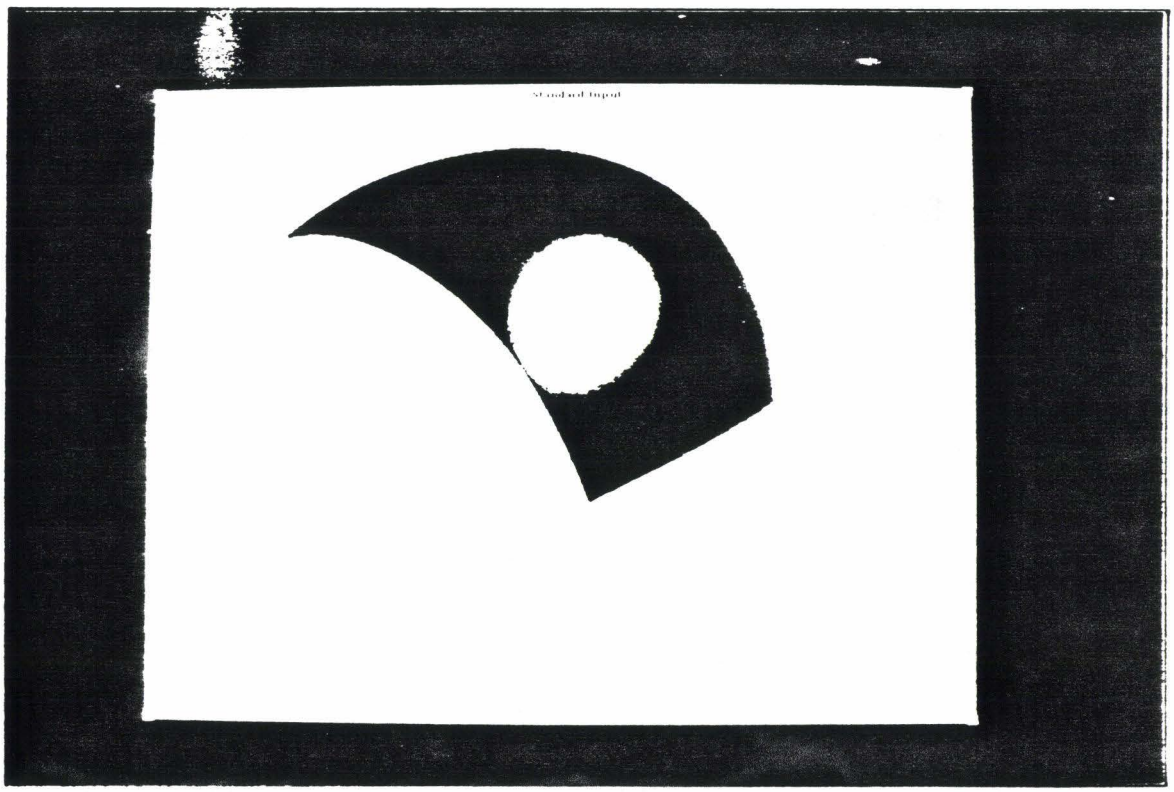
Changement d'échelle appliqué à l'objet.



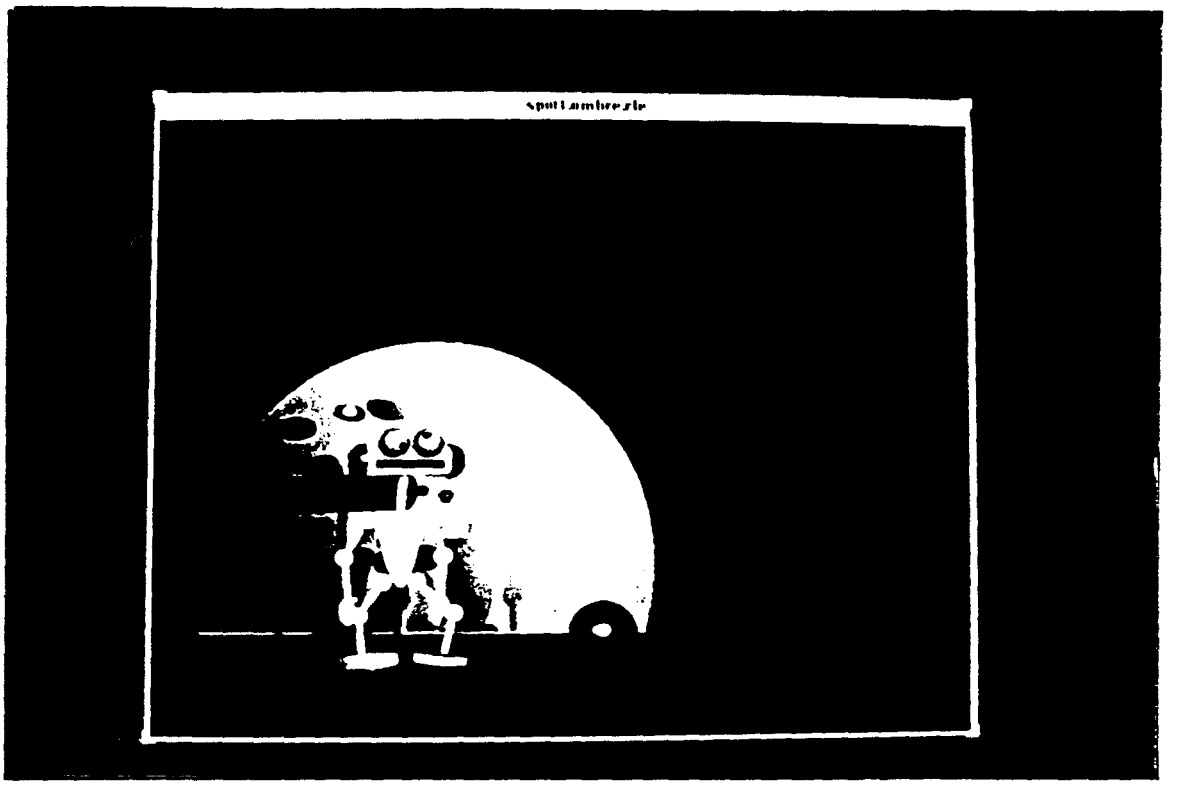
Autre vue de la théière, qui montre cependant les problèmes de raccord dûs à la modélisation.



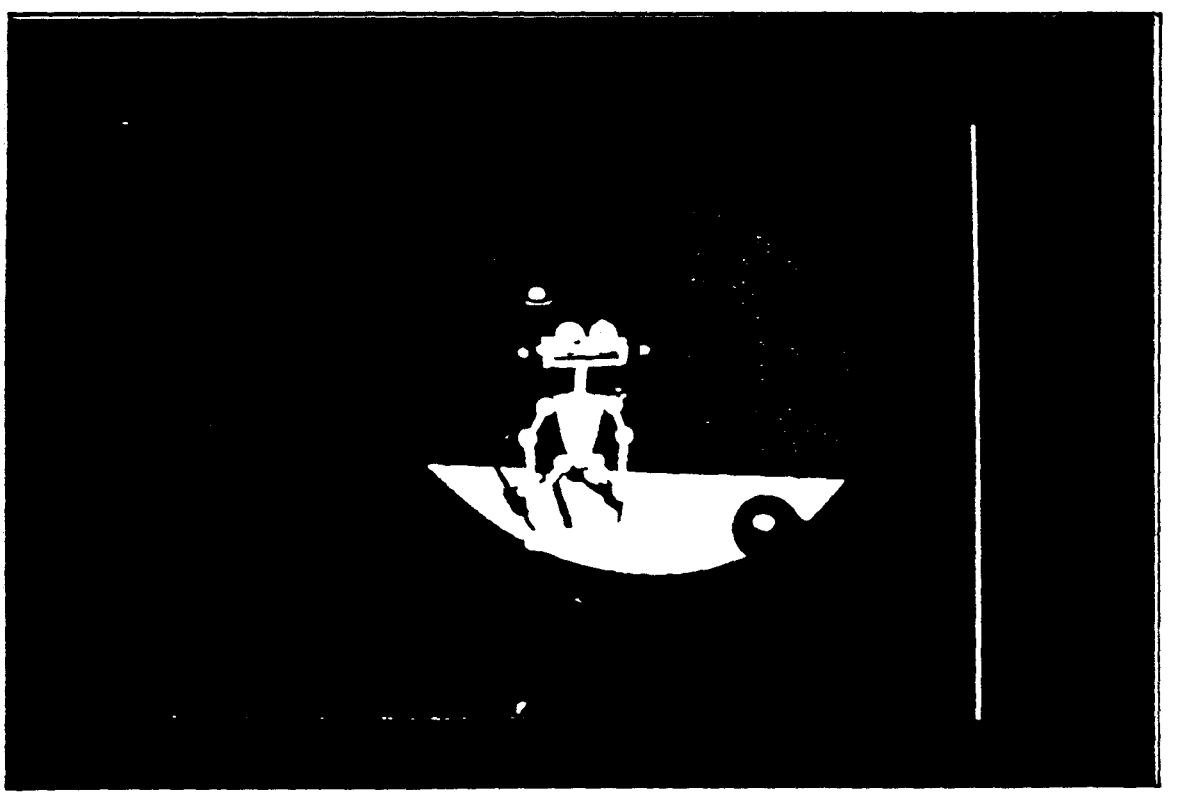
Nous tenons à remercier Max Froumentin pour la réalisation de cette image, qui montre un patch quadrique défini dans un tétraèdre. Cette image a été réalisée sur une station SUN.

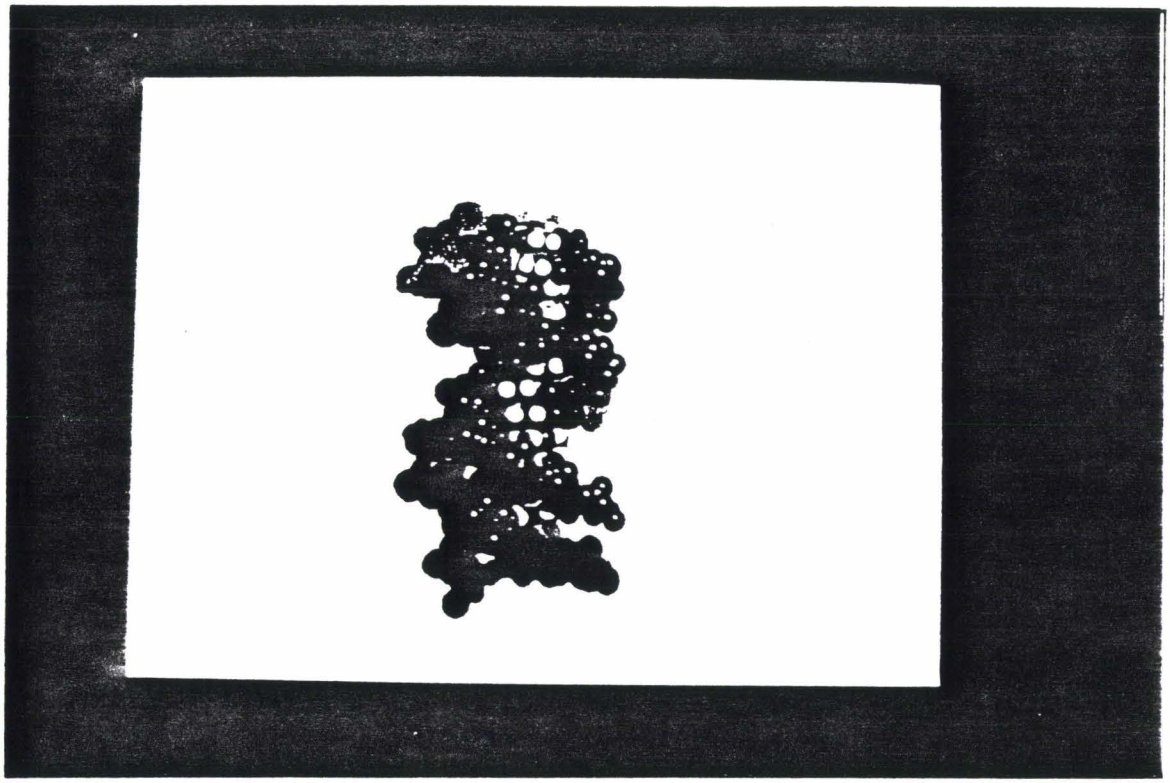


Voici le rendu du patch quadrique à l'aide du simulateur.

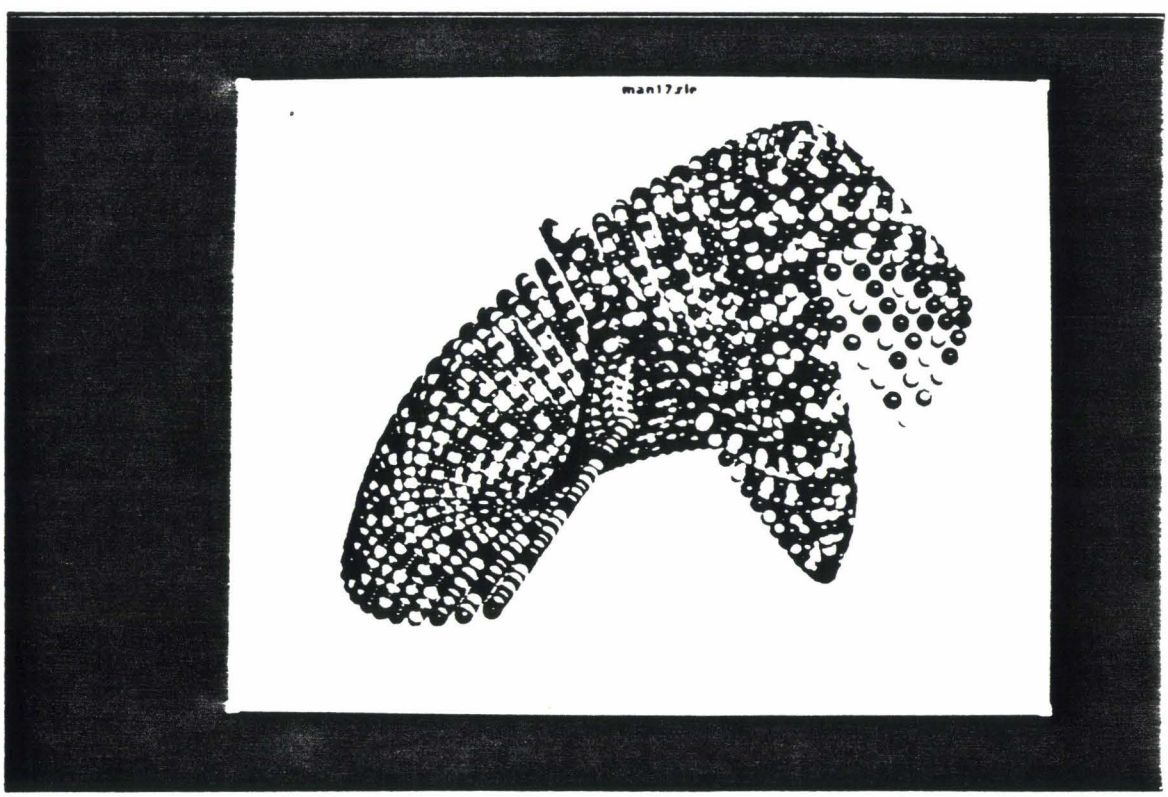


Le robot Micro1 est construit à l'aide de 39 quadriques. Les scènes contiennent au total 42 primitives (deux facettes ayant servi à la définition du sol et du fond et une quadrique définissant la sphère), une source lumineuse ponctuelle, un spot défini par un volume conique et 161 volumes d'ombre (159 volumes quadriques et deux volumes polyèdriques) pour l'application des ombres portées par la méthode de Crow. Nous remercions D. Degrande qui a défini Micro1.





Ce morceau de molécule d'ADN est défini par 483 sphères représentant les atomes. La scène contient en plus une facette définissant un fond et deux sources lumineuses ponctuelles. Le nombre de volumes d'ombre est de 966 volumes quadriques et un volume polyèdrique.



La représentation de la surface de Mandala a été réalisée à l'aide de 4000 sphères.

Bibliographie

[Akel88] K. Akeley & T. Jermoluk
High-Performance Polygon Rendering
ACM Computer Graphics, vol. 22 num. 4, august 1988, pp 239-246

[Bier86] E. Bier & K. Sloan Jr
Two-Part Texture Mappings
IEEE Computer Graphics and Applications, vol. 9, 1986, pp 40-53

[Brun90] J.-M. Brun
Surfacique et volumique. Deux Types de Modélisation ou deux Approches de la Modélisation
Revue de CFAO et d'Infographie, vol. 5 num. 3, 1990, pp17-33

[Catm80] E. Catmul & A. Smith
3-D Transformations of Images in Scanline Order
ACM Computer Graphics, vol. 14 num. 3, july 1980, pp 279-285

[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 I, janvier 1991

[Chai92] C. Chaillou
Architecture des Systèmes pour la Synthèse d'Images
Dunod Informatique, 1992

[Chan89] L.S. Chang, M. Shantz & R. Rocchetti
Rendering Cubic Curves and Surfaces with Interger Adaptive Forward Differencing
ACM Computer Graphics, vol.23 num. 3, july 1989, pp 157-166

[Clau91] U. Claussen
Real Time Phong Shading
Advanced in Computer Graphics Hardware V, Springer Verlag, 1991

[Cowg63] D. Cowgill
Logic Equations for a Built-In Root Method
IEEE Transactions on Electronic Computers, april 1964, pp 156-157

[Crow77] F. Crow
Shadow Algorithms for Computer Graphics
ACM Computer Graphics, vol. 11 num. 3, july 1977, pp 242-248

[Damh89] W. Damhen
Mathematical Methods in CAGD
Academic Press, 1989, pp 181-193

[Deer88] M. Deering, S. Winner & al
The triangle Processor and Normale Vector Shader : a VLSI System for High Performance Graphics
ACM Computer Graphics, vol. 22 num. 4, august 1988, pp 21-30

[Elli91] J.L. Ellis, G. Kedem & al.
The Ray Casting Engine and Ray Representations
Proceedings ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, june 1991, pp 255-267

[Eyle88] J. Eyles, J. Austin, H. Fuchs, T. Greer & J. Poulton
Pixel-Planes 4: A Summary
Advanced in Computer Graphics Hardware II, Springer Verlag, 1988

[Fari92] G. Farin
Curves and Surfaces for CAGD
Academic Press, 1992

[Fole90] J. Foley, A Van Dam, S. Feiner & J. Hughes
Computer Graphics : Principles and Practice (second edition)
The Systems Programming Series, Addison Wesley 12110, 1990

[Frou93] M. Froumentin & C. Chaillou
Déformation interactive de patchs quadriques
Actes des journées AFIG-GROPLAN, décembre 1993

[Fuch85] H. Fuchs, J. Golfeather, J. Hultquist & 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 & 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

[Gaga91] A. Gagalowicz
Special Modelings
Eurographics 91 Technical Report, Tutorial Note, 1991

[Ghaz87] D. Ghazanfarpour & B. Peroche
A Fast Antialiasing Method with a Z-Buffer
Proceedings Eurographics 87, august 1987, pp 503-513

[Gold86] J. Goldfeather
Fast Constructive Solid Geometry Display in the Pixel-Power Graphics System
ACM Computer Graphics, vol. 20 num. 4, august 1986, pp 107-116

[Gour71] H. Gouraud
Continuous Shading of Curved Surfaces
IEEE Transaction on Computers, vol. C-20 num. 6, june 1971, pp 623-629

[Guo93] Guo B.
Representation of Arbitrary Shapes using Implicit quadrics
The Visual Computer, vol. 9 num. 5, 1993, pp 267-277

[Hash90] R. Hashemian
Square Rooting Algorithms for Interger and Floating Point Numbers
IEEE Transactions on Computer, vol. 19 num. 8, august 1990, pp 1025-1029

[Heck86] P. S. Heckbert
Survey of Texture Mapping
IEEE Computer Graphics and Applications, vol. 6, 1986, pp 56-67

[Hegr85] G. Hégron
Synthèse d'Image : Algorithmes Elémentaires
Dunod Imformatique, 1985

[Inmo89] Inmos
The Transputer Databook
Technical Report, 1989

[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 & M. Meriaux
Real-Time Display of Quadric Objects in the IMOGENE Machine
Proceedings ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, june 1991, pp 269-277

[Karp93] S. Karpf
Architecture Massivement Parallèles pour la Synthèse d'Images Temps Réel
Thèse de Doctorat, Université de Lille, janvier 1993

[Kede89] G. Kedem & J.L. Ellis
The Ray Casting Machine
Parallel Processing for Computer Vision and Display, Addison Wesley, p 378-401

[Kirk90] D. Kirk & D. Voorhies
The Rendering Architecture of the DN10000VS
ACM Computer Graphics, vol. 24 num.4, august 1990, pp 299-307

[Klei93] R. van Kleij
Display of Solid Models with Quadratic Surfaces
PhD Thesis, Delft University, april 1993

[Knit93] G. Knittel
a VLSI Design for Fast Vector Normalization
Proceedings Eight Eurographics Workshop on Graphics Hardware, september 1993, pp 1-14

[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

[Lapo93] H. Laporte, E. Nyiri, M. Froumentin & C. Chaillou
Direct Visualization of Quadrics
Proceedings Eight Eurographics Workshop on Graphics Hardware, september 1993, pp 44-55

[Lefe92] V. Lefevere & C. Chaillou
Low Cost Hardware for Real-Time Phomg Lighting
Proceedings Graphics Interface 92 Workshop on Local Illumination, may 1992, pp 23-30

[Lefe94] V. Lefevere
Architecture Spécialisée pour l'éclairement de Phong en Temps-Réel
Thèse de Doctorat, Université de Lille I, à soutenir en février 1994.

[Levi76] J. Levin
A Parametric Algorithme for Drawing Pictures of Solid Objects Composed of Quadric Surfaces
Communications of the ACM, vol. 19 num. 10, 1976, pp 555-563

[Lewi89] J. P. Lewis
Algorithms for Solid Noise Synthesis
ACM Computer Graphics, vol. 23 num 3, july 1989, pp 263-270

[Luca91] M. Lucas
Parallélisme et Synthèse d'Images
TSI, vol. 10 num. 3, 1991, pp 171-202

[Ma86] Song-De Ma
Modélisation et Synthèse de Texture, application à l'Infographie
Thèse de Doctorat, Université Pierre et Marie Curie, Paris VI, juin 1986

[Meri84] M. Mériaux
Contribution à l'Imagerie Informatique : Aspects Algorithmiques et Architecturaux
Thèse d'Etat, Université de LilleI, 1984

[Mill86] J. R. Miller
Sculptured Surfaces in Solid Models : Issues and Alternative Approaches
IEEE Computer Graphics and Applications, vol. 6, 1986, pp 37-47

[Mill88] J. R. Miller
Analysis of Quadric-Surface-Based Solid Models
IEEE Computer Graphics and Applications, vol. 8, 1988, pp 28-42

[Moln91] S. Molnar
Image-Composition Architecture for Real-Time Image Generation
PhD Thesis, University of North Carolina, october 1991

[Nyir92a] E. Nyiri & C. Chaillou
Aspect Logiciel du Projet IMOGENE
Actes de MICAD 92, Editions Hermès, pp 201-217

[Nyir92b] E. Nyiri, C. Chaillou & M. Froumentin
Le Polyèdre et la Quadrique comme Primitives d'Affichage
Actes des Journées GROPLAN 92, novembre 1992, pp 31-37

[Peac85] D.R. Peachey
Solid Texturing of Complex Surfaces
ACM Computer Graphics, vol. 19 num. 3, july 1985, pp 279-286

[Perl85] K. Perlin
An Image Synthetiser
ACM Computer Graphics, vol. 19 num. 3, july 1985, pp 287-296

[Perl89] K. Perlin
Hypertexture
ACM Computer Graphics, vol.23 num. 3, july 1989, pp 253-262

[Preu91] A. Preux
Le C.S.G. et le Temps Réel
Mémoire de DEA, Université de Lille, septembre 1991

[Preu92] A. Preux & C. Chaillou
Un Algorithme Economique pour l'Anti-aliassage des Bords de Polygones
Actes des Journées GROPLAN 92, novembre 1992, pp 23-30

[Phon75] B.T. Phong
Illumination for Computer Generated Pictures
Communications ACM, vol. 18 num. 18, june 1975, pp 311-317

[Potm82] M. Potmesil
Generating Three-Dimensional Surface Models of Solid Objects from Multiple Projections
PhD Thesis, Rensselaer Polytechnic Institute Troy, New York, august 1982

[Powe77] M. J. D. Powel & M. A. Sabin
Piecewise Quadriatric Approximations on Triangles
ACM Transations of Mathematical Software, vol. 3 num 4, 1977, pp 316-325

[Sili92] Silicon Graphics
Reality Engine in Visual Simulation
Technical Report, 1992

[Schn88] B.O. Schneider & U. Claussen
PROOF : an Architecture for Rendering in Object Space
Advanced in Computer Graphics Hardware III, Springer Verlag, 1988

[Slat92] M. Slater
A Comparison of three Shadow Volume Algorithme
The Visual Computer, vol. 9 num. 1, 1992, pp 25-38

[Wang92] L. R. Wanger, J. A. Ferwerda & D.P. Greenberg
Perceiving Spacial Relationships in Computer-Generated Images
IEEE Computer Graphics and Applications, may 1992, pp 44-57

[Will78] L. Williams
Casting Curved Shadows on Curved Surfaces
ACM Computer Graphics, vol. 12 num. 3, july 1978, pp 270-274