NUMERO D'ORDRE : 1775 ANNEE : 1996

USTL

LIFL Laboratoire d'Informatique Fondamentale de Lille

CNRS

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

pour obtenir le titre de

DOCTEUR EN INFORMATIQUE

par

Hervé LAPORTE

ETUDE LOGICIELLE ET MATERIELLE D'UN SYSTEME DE VISUALISATION TEMPS-REEL BASE SUR LA QUADRIQUE

Thèse soutenue le 27 Juin 1996, devant la commission d'examen :

Président :	V. Cordonnier	LIFL
Directeur de thèse :	C. Chaillou	LIFL
Rapporteurs :	R. Caubet	IRIT
	B. Arnaldi	IRISA
Examinateurs :	M. Mériaux	Université de Poitiers
	P. Chenin	LMC-IMAG

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. 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

M. MIGEON Michel	EUDIL
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie appliquée

PROFESSEURS - 1ère CLASSE

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

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

PROFESSEURS - 2ème CLASSE

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

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

M. 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

Table des matières

Remerciements

Merci à Christophe Chaillou qui a su me conseiller et supporter mes états d'âmes au cours de mes quatre années de thèse. Son aide précieuse m'a permis d'aller au bout de cette thèse et d'en faire une expérience enrichissante et réussie.

Merci à René Caubet et Bruno Arnaldi qui ont accepté de rapporter ma thèse. Leurs remarques pertinentes et constructives sur mon travail m'ont permis d'améliorer la qualité de mon manuscrit.

Merci à Vincent Cordonnier qui m'a fait l'honneur de présider mon jury. Rappelons qu'il est aussi à l'origine du projet IMOGENE dont découle cette thèse.

Merci à Patrick Chenin et Michel Mériaux qui ont bien voulu juger ce travail. Michel Mériaux a également été mon directeur de thèse pendant trois ans.

Merci à Henri Glanc qui a reproduit ce manuscrit avec toute la célérité et la bonne humeur qu'on lui connaît.

Merci au Coyote, au Wookie, à Gregounet, à Philou, à Samclone et tous les autres sans qui le labo ne serait pas un lieu où il fait bon travailler.

Enfin, merci à mes parents, ma famille et tout particulièrement Gwen qui m'ont soutenu et m'ont offert les meilleures conditions possibles pour réussir.

Introduction

Le domaine de la synthèse d'image est né de la convergence de deux phénomènes complètement différents. D'une part l'apparition de machines capables de manipuler des quantités considérables d'informations en un temps de plus en plus court. D'autre part la puissance de l'image comme vecteur de communication.

On considère généralement la thèse d'I. Sutherland sur le logiciel graphique SKETCHPAD, comme l'acte de naissance de la synthèse d'image. Depuis celle-ci a évolué et progressé constamment tant sur le plan logiciel que matériel.

Ces progrès ont permis à la synthèse d'image de s'implanter dans de nombreux domaines d'applications. La CAO (notamment mécanique) et les simulateurs de vols (tout d'abord militaires puis civils) ont été longtemps les domaines les plus porteurs. Aujourd'hui la synthèse est utilisée aussi bien dans l'art (cinéma, clips vidéos, spots publicitaires), en architecture et urbanisme, dans les simulateurs (de conduite, médicaux, etc) et de plus en plus en réalité virtuelle et dans les jeux vidéos.

Il y a encore quelques années, rares étaient ceux qui pouvaient s'offrir des machines capables non seulement de calculer et d'afficher des images mais de plus de le faire vite. Cette possibilité était réservée aux simulateurs de vol dont le coût était de l'ordre de dix millions de francs. Aujourd'hui une console de jeu offre quasiment les mêmes fonctionnalités qu'un simulateur (avec toutefois des performances moindres) pour deux mille francs.

Entre les deux, environ vingt ans de recherches universitaires et industrielles ont permis de trouver des solutions aux problèmes de la visualisation temps-réel. Celle-ci a nécessité des développements spécifiques aussi bien au niveau de l'algorithmique que du matériel. Un des points clés pour le calcul des images en temps-réel est l'usage de la facette comme primitive de visualisation.

L'utilisation de la facette a permis l'essor de la synthèse temps-réel de part sa simplicité. Néanmoins, et malgré de nombreuses études, la facette introduit certains défauts dans les images produites. En particulier les objets affichés, qui se composent d'une multitude de facettes, ont un aspect anguleux.

Pour éliminer ces défauts nous recherchons une alternative à la facette. Notre objectif est de proposer une nouvelle primitive de visualisation qui soit dénuée des défauts de la facette. Notre choix s'est porté sur la quadrique qui est la primitive courbe la plus simple. La démarche qui nous y a conduit fait l'objet du chapitre un.

Cette remise en cause implique de profonds changements dans l'organisation et l'algorithmique du processus de visualisation. Pour bien cerner les difficultés et mettre en évidence les problèmes nouveaux nous établissons un état de l'art sur la quadrique dans le chapitre deux. Nous commençons par donner des notions mathématiques de base. Ensuite nous abordons des questions de modélisation puis de visualisation. Enfin nous revenons sur les études déjà menées ou en cours au LIFL. Dans le bilan nous établissons un cahier des charges d'un système de visualisation temps-réel basé sur la quadrique.

Le chapitre trois regroupe les solutions algorithmiques que nous avons apportées au cours de cette thèse au problème de la visualisation temps réel d'objets constitués de quadriques. Les trois points principaux sont la définition d'une primitive de visualisation unique et générique, la proposition d'un algorithme de placage de texture et l'intégration des fonctionnalités usuelles dans un système cohérent.

Une mise en oeuvre matérielle et logicielle fait l'objet du chapitre quatre. La conception d'un processeur de rendu est d'abord présentée. Une évaluation du coût fournit des informations précieuses sur la complexité et donc la faisabilité du processeur. Ensuite une étude d'une machine complète est proposée. Enfin un logiciel a été développé à des fins de simulation et de validation du processeur de rendu et de la machine dans laquelle il s'intègre.

Le chapitre cinq fait le bilan de cette étude. Il présente nos conclusions sur l'ensemble des recherches qui ont été réalisées au LIFL sur la quadrique. Enfin les perspectives ouvertes par cette étude sont abordées.

Cadre général

Le processus de synthèse d'images consiste à produire une *image* à partir de la description d'une *scène* en fonction du *système de visualisation*. En infographie une image est une grille de points élémentaires appelés *pixels*. Suffisamment petits, un grand nombre de pixels permet de donner l'illusion d'un dessin continu. Une scène est l'ensemble des objets qui se trouve dans le "monde" que l'on veut visualiser. Le système de visualisation est l'équivalent informatique de l'oeil. Approximativement il est déterminé par la position de l'observateur et la direction dans laquelle celui-ci regarde.

Le processus de synthèse d'images se décompose en deux étapes qui sont la *modélisation* et le *rendu*. La première étape a pour but de décrire de façon abstraite (mathématique) les objets qui composent une scène. La *modélisation géométrique* concerne la description géométrique des objets, à savoir la forme, la taille, la position relative, etc. La *modélisation photométrique* consiste à décrire comment la lumière éclaire la scène. Pour cela on utilise un *modèle d'éclairement* qui est la mise en équations de la façon dont les flux lumineux transitent dans une scène.

La deuxième étape est la phase de calcul de l'image. Le rendu est la détermination d'une couleur pour chacun des pixels. Pour cela il faut tout d'abord déterminer quel est l'objet présent au pixel considéré. Chaque objet est donc converti en l'ensemble des pixels qu'il recouvre, c'est la phase de *conversion objet-pixels*. Dans la mesure où plusieurs objets peuvent être présents au même pixel, seul l'objet effectivement vu doit être pris en compte. C'est la phase d'*élimination des parties cachées*. Ensuite la couleur du pixel est déterminée en fonction des caractéristiques de l'objet et du modèle d'éclairement utilisé pour rendre compte des interactions lumineuses.

Nous décrivons brièvement les méthodes classiques de modélisation géométrique et les principales méthodes de rendu. Ensuite nous nous intéressons plus particulièrement au *rendu temps-réel*. Nous définissons ce qu'est un *accélérateur graphique* et nous présentons les techniques qui y sont utilisées.

Bien que considérées comme deux domaines distincts, la modélisation géométrique et la visualisation sont étroitement liées. Les méthodes utilisées en modélisation ont des conséquences sur les algorithmes de visualisation. A l'inverse les possibilités et les limites en visualisation engendrent des contraintes sur la modélisation. Nous proposons une analyse de ces diverses répercussions qui fait apparaître la notion de *primitive de rendu* (ou *primitive de visualisation*) qui se distingue de celle de primitive de modélisation.

Nous mettons alors en évidence les faiblesses introduites par cette distinction. En particulier les objets affichés sont souvent des approximations des objets modélisés. Par ailleurs les problèmes induits par l'utilisation de la facette en tant que primitive de visualisation sont exposés. Nous en déduisons qu'une solution à ces problèmes consiste à remettre en cause la facette comme primitive de visualisation. Une discussion montre qu'une alternative pertinente est la quadrique.

1.1 Modélisation géométrique

Un modèle géométrique permet de décrire l'ensemble des objets constituant une scène. Une étude exhaustive de ces modèles nécessite deux étapes. Tout d'abord une étude des différentes primitives que l'on peut utiliser. Ensuite une étude des différentes façons de combiner ces primitives pour obtenir des objets complexes. Notre but ici est simplement de présenter les méthodes de modélisation les plus répandues. Nous les regroupons en fonction de la dimension des primitives utilisées : représentations surfacique et volumique.

1.1.1 Représentation surfacique

Un objet est représenté par sa surface [Farin92]. Différents modèles ont été proposés, selon le type de surface utilisée pour représenter les objets. Les trois principales catégories sont les représentations à facettes polygonales, les carreaux de surfaces et les surfaces implicites.

Facettes polygonales

Ce type de représentation correspond bien à certains objets comme une boîte, un immeuble ou un placard. Toutefois les objets arrondis ne peuvent être qu'approchés. Un objet est représenté par un ensemble de facettes polygonales qui peut être organisé selon différentes méthodes. La plus simple, appelée *représentation explicite*, consiste à énumérer tous les sommets de chaque facette, mais comme un sommet appartient en général à plusieurs facettes, il y a redondance de l'information. La représentation par pointeurs sur une liste de sommets évite que les sommets soient répétés plusieurs fois (Figure 1.1). Une autre méthode, appelée *winged-edge* [Baumgart72] est souvent utilisée pour les informations structurelles qu'elle offre sur l'objet.

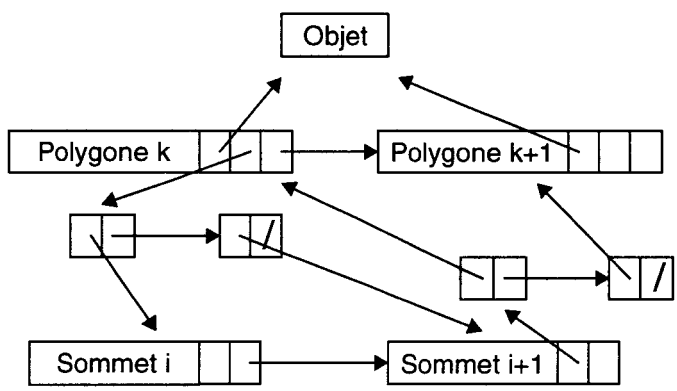


Figure 1.1 ***Représentation par pointeur sur liste de sommets***

Carreaux de surfaces

Un objet est défini par un ensemble de carreaux élémentaires, c'est-à-dire un ensemble de morceaux de surfaces complexes (Figure 1.2). Ces surfaces sont définies paramétriquement à partir de points de contrôle affectés d'une fonction de pondération, polynomiale ou non. De ces fonctions de pondération découlent des propriétés intéressantes : possibilité de contrôle local ou global de la forme de la surface, degré de continuité entre carreaux, interpolation des points de contrôle, nombre de paramètres à manipuler pour définir la surface etc. Ces avantages se payent par un temps de calcul plus long pour l'affichage et les opérations de modélisations.

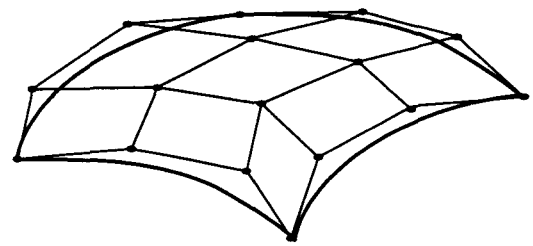


Figure 1.2 ***Une surface complexe avec seize points de contrôle***

Un des pionniers du domaine est Bézier [Bézier70] [Bézier74] qui a développé les courbes et surfaces qui portent son nom dans le but de modéliser des carrosseries automobiles. Plus tard une autre famille de courbes est devenue prépondérante dans le domaine de la modélisation surfacique : la famille des splines dans laquelle on citera les B-splines, les β-splines et les NURBS [Farin92].

Les surfaces paramétriques souffrent d'un certain nombre de défauts. Certaines opérations telles que l'extrusion (qui définit un volume à partir d'une surface qui suit un chemin) ne sont pas closes. D'autres, comme l'intersection ou le *blending* (qui définit une surface qui se raccorde en tangence avec deux surfaces données) sont parfois difficiles à calculer. L'usage des surfaces paramétriques en modélisation géométrique est aujourd'hui encore un domaine de recherche très actif. Néanmoins on voit apparaître depuis quelques temps une nouvelle approche pour pallier les problèmes de ces surfaces : il s'agit des surfaces implicites.

Les surfaces implicites

Les surfaces implicites [Sederberg85] [Bloomenthal90] constituent un domaine de recherche assez récent. Elles évitent la plupart des inconvénients liés aux surfaces paramétriques. En revanche il est beaucoup moins aisé de contrôler la forme de l'objet que l'on désire. Il existe deux grandes catégories de surfaces implicites, les surfaces algébriques et les modèles à squelette.

Les surfaces algébriques profitent des propriétés des polynômes. Ces surfaces sont en général d'un degré faible, ce qui simplifie les calculs. Elles contiennent toutes les surfaces paramétriques. De plus la fermeture des opérations telles que l'*offset* (qui définit une surface dont chaque point se trouve à une distance constante dans la direction de la normale à la surface donnée) ou l'intersection[1] est assurée. Notons que les quadriques sont des surfaces implicites simples puisque de degré deux.

Les modèles à squelette offrent un moyen très intuitif et facile pour définir des formes globuleuses. Au départ on a des primitives de très haut niveau qui sont ensuite mélangées (on parle de *blending*) pour obtenir des formes plus complexes. Les éléments du squelette sont en général le point, le segment de courbe paramétrique (ou plus simplement le segment de droite) et le polygone. La primitive implicite est définie par une fonction implicite, dite *de potentiel*, qui pour des raisons de simplicité est souvent une fonction de la distance au squelette. Les primitives sont ensuite mélangées pour obtenir un objet complexe (cf. Figure 1.3).

1. On entend par là que l'intersection de deux surfaces algébriques est toujours une courbe algébrique

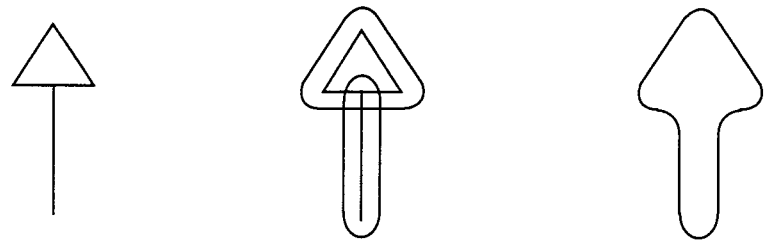


Figure 1.3 ***A partir d'un squelette (gauche), on définit deux primitives implicites (centre) pour obtenir une surface (droite). Illustration d'après [Bloomenthal90].***

1.1.2 Modélisation volumique[1]

Elle consiste en la modélisation d'objets ayant un volume. Il existe différentes méthodes de modélisation de solide ([Mantyla88], [Foley90] chapitre 12). Les deux plus importantes sont la B-Rep et le CSG. Par ailleurs notons qu'il est souvent intéressant de pouvoir passer d'une représentation à une autre.

Représentation B-Rep

La *Boundary Representation* ou *B-Rep* représente un volume par les surfaces qui le délimitent. La plupart du temps ces surfaces sont des polygones plans pour des raisons de simplicité mais on peut utiliser d'autres types de surfaces (dans le cas général cela devient plus compliqué voire impossible). Ainsi sur l'exemple de la Figure 1.4 on utilise des morceaux de cylindre et un disque.

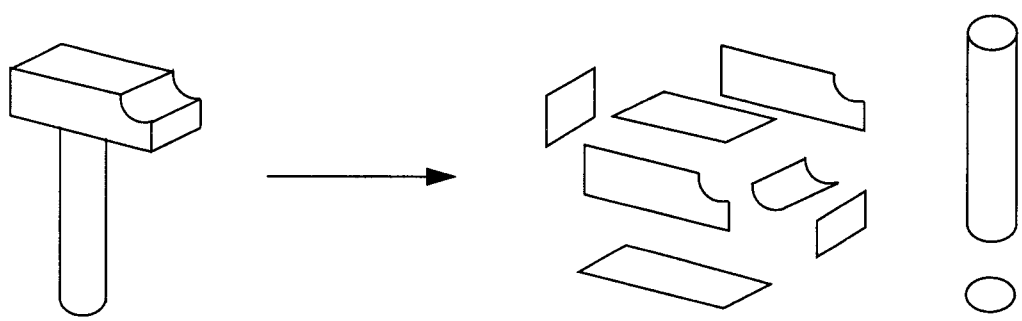


Figure 1.4 ***B-rep d'un objet solide***

Représentation CSG

En modélisation CSG (Constructive Solid Geometry) un objet est défini à partir d'un ensemble de solides de base combinés par les opérateurs booléens régularisés (typiquement l'union, l'intersection et la différence). On a alors deux représentations équivalentes de l'objet, sous forme d'arbre ou sous forme d'expression booléenne (Figure 1.5).

1. Solid modeling en anglais

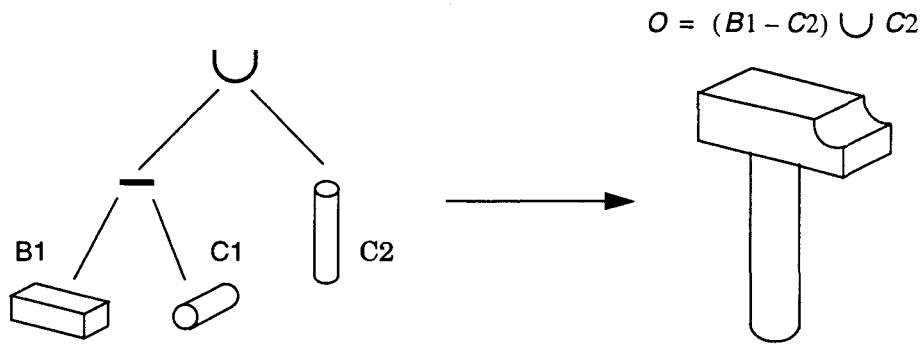


Figure 1.5 ***Un exemple d'objet CSG représenté sous forme d'arbre***

L'avantage majeur qui a fait le succès de ce type de modélisation est qu'il était bien adapté à la conception d'objets mécaniques tels que les pièces de moteurs, pompes hydrauliques, boites de vitesses etc. En effet les objets de base correspondaient aux blocs de métaux non façonnés et les opérateurs booléens correspondaient aux actions que les machines outils étaient capables de réaliser.

Passage d'une représentation CSG à une B-Rep

Si différentes représentations des solides existent c'est qu'elles offrent chacune leurs avantages et leurs inconvénients. Pouvoir passer de l'une à l'autre est donc une façon de s'affranchir des inconvénients et de ne retenir que les avantages. En modélisation cela permet de mélanger des objets modélisés dans plusieurs représentations, en général B-rep et CSG. En ce qui nous concerne, nous verrons au paragraphe 1.3 que le passage d'une représentation CSG à une représentation B-rep est utile pour le rendu.

1.2 Visualisation

La phase de *visualisation* ou *rendu* consiste à produire des images de la scène. Aujourd'hui la production d'images de synthèse se fait selon deux axes. Le *rendu réaliste* consiste à rechercher la meilleure qualité d'image. Pour cela, des algorithmes très complexes sont mis en oeuvre. Les applications se trouvent dans les domaines où l'on cherche à avoir une représentation fidèle de la "réalité" comme par exemple en architecture, ou une impression visuelle forte comme pour le cinéma.

En *rendu rapide*, l'objectif principal est de produire des images très rapidement afin d'assurer une fluidité d'animation. C'est le domaine des simulateurs ou des jeux vidéo (l'intersection de ces deux domaines est d'ailleurs non nulle). Bien sûr, la rapidité est obtenue au détriment de la qualité.

Nous présentons tout d'abord les différentes techniques pour produire une image. Dans la mesure où notre étude s'inscrit dans le cadre du rendu rapide nous allons nous intéresser plus particulièrement à l'une de ces techniques, le *rendu projectif*. En effet celui-ci est actuellement la seule méthode de rendu utilisée pour calculer des images en *temps-réel*.

Ce terme est défini et nous signalons ses implications principales. Nous définissons ce qu'est un *accélérateur graphique* et décrivons les moyens mis en oeuvre habituellement dans ces machines pour la production d'images en temps-réel.

1.2.1 Techniques de rendu

Il n'existe que deux grandes méthodes pour visualiser une scène, le rendu projectif et le lancer de rayon. Nous expliquons tout d'abord le principe du lancer de rayons (le rendu projectif est présenté au paragraphe suivant). Ensuite nous décrivons brièvement deux autres classes d'algorithmes, qui ne sont pas à proprement parler des méthodes de rendu mais qui sont en général étudiées à ce titre : il s'agit de la radiosité et du "rendu" volumique.

Le lancer de rayon

Le lancer de rayon [Whitted80] simule le parcours de la lumière dans une scène en suivant le chemin inverse des rayons lumineux. On lance depuis l'observateur un rayon lumineux, dit primaire, par pixel de l'écran. Le premier objet de la scène rencontré par ce rayon sera l'objet visible, on effectue ainsi l'élimination des parties cachées. L'éclairement doit être alors calculé. Pour cela, des rayons, dits secondaires, sont lancés. Un rayon réfléchi permet d'obtenir des effets de miroir. Un rayon réfracté permet d'obtenir des effets de transparences. Un rayon vers les sources lumineuses (un par source ponctuelle) permet d'obtenir des effets d'ombres portées. Ces rayons peuvent à leur tour rencontrer un objet et générer de nouveaux rayons.

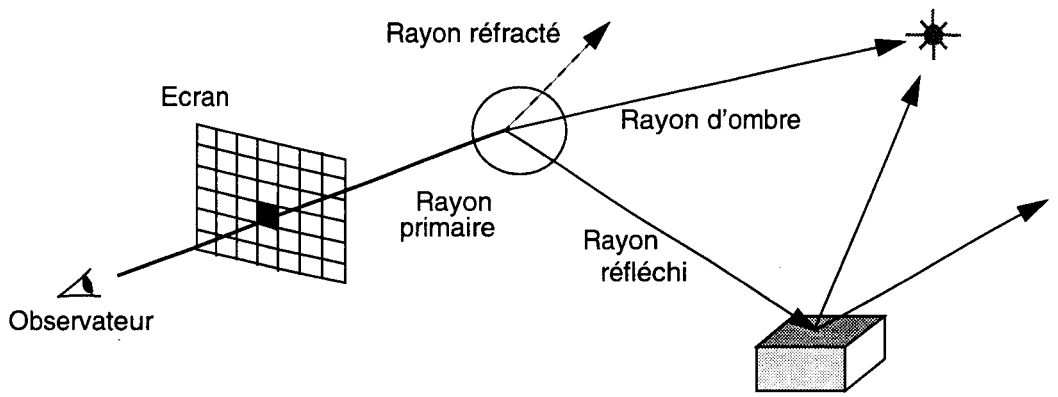


Figure 1.6 *Le lancer de rayon*

Cet algorithme nécessite un nombre particulièrement important de calculs d'intersections objet-rayon. En effet pour déterminer quel objet un rayon rencontre il faut calculer l'intersection du rayon avec tous les objets pour savoir si il y a effectivement intersection. A titre d'exemple prenons une scène d'environ dix mille objets et un écran de mille fois mille pixels. Rien que pour les rayons primaires on aura donc dix mille objets fois un million de rayons, c'est-à-dire dix milliards d'intersections. Heureusement, il existe des solutions pour accélérer les calculs. On peut soit diminuer le nombre d'intersections à calculer par des mécanismes de boîte englobante ou de partitionnement de l'espace, soit paralléliser les calculs. Toutes ces méthodes et d'autres sont décrites dans [Glassner89].

La radiosité

La radiosité [Goral84] est une méthode d'illumination globale, c'est-à-dire qu'elle permet de calculer l'éclairement "en tout point" d'une scène en tenant compte des interactions lumineuses entre les objets de la scène. Elle est basée sur la propriété physique de conservation de l'énergie dans un espace clos. La scène est tout d'abord décomposée en un ensemble de facettes. Ensuite on calcule, pour chaque couple de facettes, la quantité d'énergie lumineuse diffuse échangée.

Pour obtenir une image il faut alors effectuer le rendu, soit par rendu projectif soit par lancer de rayon. Dans ce dernier cas, on peut ajouter une composante spéculaire pour gérer les reflets.

Les volumes de calculs à effectuer sont considérables. Toutefois la qualité de l'image ainsi obtenue est excellente, on parle souvent de qualité photographique, c'est-à-dire que l'image de synthèse ne peut pas se distinguer de l'image réelle.

La première méthode implémentant efficacement cette technique de radiosité est due à Cohen et al. en 1985 [Cohen85]. Depuis de nombreuses recherches sont menées soit pour accélérer les calculs, soit pour accroître la qualité le l'image [Sillion94].

Le rendu volumique

Appelé également rendu voxel, le rendu volumique consiste à calculer une image à partir d'un ensemble de points tridimensionnels appelés voxels, soit par rendu projectif soit par lancer de rayon [Stolte95]. Historiquement, ce type de rendu s'est développé dans le cadre de l'imagerie médicale puisque certains appareils fournissent directement des données de type voxel (scanner, RMN[1]). Aujourd'hui de nombreux travaux sont en cours sur le rendu voxel. Un aperçu de ces recherches est proposé dans [Kaufman91] et [Sims96].

1.2.2 Le rendu projectif

Le principe du rendu projectif est de projeter sur l'écran tous les objets de la scène (Figure 1.7). Il se compose d'un ensemble de tâches réalisées les unes après les autres. C'est pour cette raisons que l'on parle de pipeline de rendu. Toutefois l'ordre dans lequel elles sont effectuées varie selon le pipeline choisi. Nous ne présentons ici que les tâches proprement dites. Le pipeline de Gouraud et celui de Phong sont présentés aux paragraphes 1.2.4 et 3.1.1.

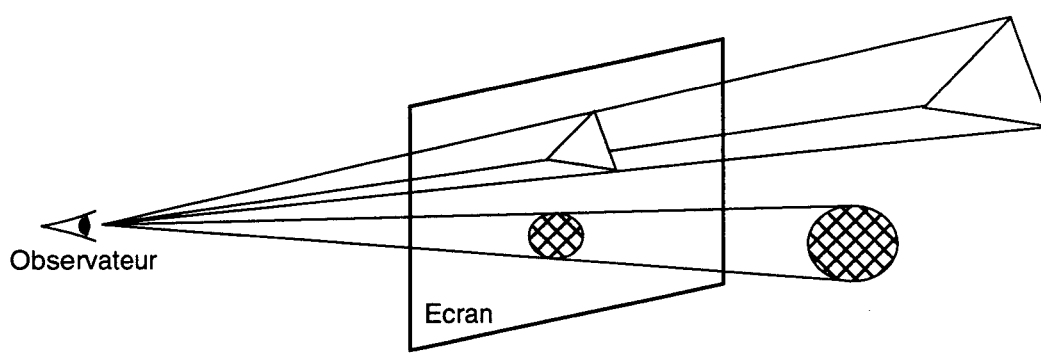


Figure 1.7 ***Rendu projectif : projection des objets sur l'écran***

- Traversée de la scène : elle consiste à récupérer (sur disque) l'ensemble des objets de la scène qui sont susceptibles de contribuer à l'image.
- Transformation géométrique : un certain nombre de transformations géométriques sont appliquées à chaque primitive afin de la faire passer de son propre repère (de modélisation) au repère monde.
- Fenêtrage : certains objets de la scène ne sont pas dans le champs de vision de l'observateur. S'ils sont complètement en dehors du champs de vision, ils doivent être éliminés. S'ils sont partiellement dans le champs de vision, ils sont découpés afin de ne garder que les parties éventuellement visibles.
- Projection sur l'écran : tous les objets de la scène sont passés dans le repère écran. La projection est généralement soit perspective (plus proche de la vision humaine), soit

1. Résonnance Magnétique Nucléaire.

orthogographique (utilisée souvent en CAO car elle ne déforme pas les objets). Une étude détaillée des différents types de projections est fournie dans [Rogers90] pp. 133-206.

- Conversion objet-pixel : elle consiste à déterminer l'ensemble des pixels recouverts par la projection des objets de la scène. Il existe deux types de méthodes. On suppose que l'objet que l'on veut convertir est un triangle. Pour le suivi de contour on détermine de proche en proche les pixels de bords droit et gauche du triangle. Tous les pixels entre les deux appartiennent au triangle considéré. Dans la méthode par équations le triangle est représenté comme étant l'intersection des demi-plans définis par les droites supports des segments du triangle. Un test d'inclusion est réalisé pour tous les pixels.
- Elimination des parties cachées : plusieurs objets peuvent recouvrir le même pixel. Il faut donc être capable de déterminer quel objet est vu par l'observateur c'est-à-dire quel objet est devant. Cette tâche est réalisée par un algorithme d'*élimination des parties cachées* [Sutherland74]. Selon la technique choisie, cette phase peut avoir lieu avant ou après la projection.
- Eclairement et ombrage : l'éclairement consiste à appliquer les équations décrites dans le modèle d'éclairement choisi à tous les points de tous les objets. Mais dans la pratique une surfaces est continue et donc contient une infinité de points. L'éclairement est donc effectué uniquement sur certains points. Toutefois, comme il faut calculer une couleur en chaque pixel, on utilise une méthode d'ombrage. Celle-ci précise quelles parties de la formule d'éclairement sont calculées pour quels points et comment.

  Les modèles d'éclairements sont décrits en annexe 6.1. Pour l'ombrage, la méthode de Gouraud est décrite au paragraphe 1.2.4 et celle de Phong au paragraphe 3.1.1.

1.2.3 Temps-réel

La synthèse d'images temps-réel est un domaine particulier de l'infographie. En effet la prise en compte de l'aspect temps-réel est particulièrement contraignante et oblige à développer des solutions algorithmiques et matérielles spécifiques. Nous donnons une définition de ce terme et nous mettons en évidence les conséquences sur les temps de calcul.

Le terme de temps-réel est relatif à la vitesse de calcul et d'affichage des images. D'un point de vue intuitif, il signifie que l'image désirée est calculée et affichée suffisamment vite pour que l'utilisateur ait l'impression que tout se passe instantanément. Ce qui veut dire que le temps-réel nécessite une vitesse qui dépend de l'utilisateur ou plus exactement qui varie selon le type d'application.

On distingue deux types de temps-réel. Le temps-réel d'interaction, utile notamment en CAO, implique que le temps de calcul d'une image soit inférieur à une seconde. Pour le temps-réel d'animation les applications principales sont les simulateurs et les mondes virtuels. On convient généralement qu'il faut environ vingt cinq images par seconde[1], ce qui correspond à la fréquence de rafraîchissement d'un écran de télévision.

Dans ce dernier cas, les contraintes de temps sont alors très élevées. En effet, une image doit être produite toutes les quarante millisecondes. Sachant que pour construire une image, plusieurs millions de pixels sont calculés, un pixel doit être fourni environ toutes les dix nanosecondes.

1.2.4 Accélérateur graphique

La réponse proposée au problème du calcul d'images en temps-réel est l'*accélérateur graphique*. Les accélérateurs graphiques sont des machines constituées de matériel spécialisé en vue d'accélérer le calcul et l'affichage des images. Ils utilisent tous (à l'exception de la Ray Casting

1. En fait ce nombre peut varier de dix à soixante images par seconde

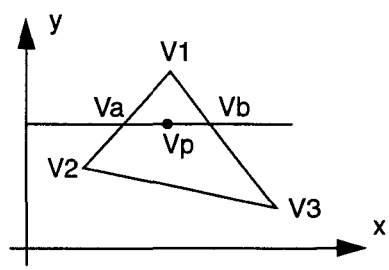
Machine, voir paragraphe 2.3.3) le rendu projectif. Par ailleurs ils mettent en oeuvre d'autres moyens classiques :

- L'usage de la facette permet la mise au point d'algorithmes suffisamment simples pour soutenir les contraintes de vitesse.
- Le pipeline de Gouraud est l'organisation classique que l'on retrouve dans la grande majorité des accélérateurs graphiques.
- Des processeurs dédiés ont été développés et intégrés au sein d'architectures spécifiques qui utilisent notamment le parallélisme.

L'usage de la facette

Historiquement, la facette s'est très vite imposée pour sa simplicité. De nombreux efforts de recherche ont été réalisés afin de trouver des méthodes efficaces pour son affichage. Tous les objets d'une scène sont facettisés, ce qui permet de ne traiter qu'un seul type d'objet à savoir la facette.

L'avantage majeur est la possibilité d'interpoler les valeurs à calculer lors de la conversion objet-pixels (Figure 1.8). Cette méthode simple permet de réduire au maximum les calculs. De plus, avec certaines précautions l'interpolation peut être réalisée avec des nombres entiers au moyen de méthodes incrémentales. Enfin, tous les algorithmes sont optimisés pour la facette, certains étant même spécifiques à celle-ci.



$$Va = V1 - (V1 - V2)\frac{y1 - ya}{y1 - y2}$$

$$Vb = V1 - (V1 - V3)\frac{y1 - yb}{y1 - y3}$$

$$Vp = Vb - (Vb - Va)\frac{xb - xp}{xb - xa}$$

Figure 1.8 *Interpolation linéaire d'une valeur sur une facette*

Le pipeline de Gouraud

En synthèse d'images temps-réel tous les accélérateurs graphiques aujourd'hui proposés sur le marché utilise le pipeline de Gouraud [Gouraud71]. Ce pipeline se compose de deux grandes parties, la préparation et le rendu (Figure 1.9). Ses caractéristiques principales sont :

- L'ombrage : l'ombrage de Gouraud consiste à calculer la couleur en chaque pixel par interpolation linéaire sur les trois composantes de couleur, rouge, vert et bleu. L'interpolation est réalisée à partir de la couleur en chaque sommet de la facette qui est calculée (lors de la phase d'éclairement) avec la formule d'éclairement de Lambert ou de Phong (cf. annexe 6.1).
- L'élimination des parties cachées : tous les accélérateurs graphiques utilisent un Z-buffer[1] [Catmull75]. Pour cela, on calcule pour tous les pixels recouverts par une primitive donnée la profondeur du point. Cette valeur est comparée à celle déjà stockée dans le tampon de profondeur. Si elle est plus petite, le point considéré est plus proche que celui déjà stocké, on le mémorise donc en lieu et place de l'ancien (ainsi que sa couleur).

1. ou tampon de profondeur. Le terme anglais est si répandu que nous utiliserons indifféremment l'un ou l'autre.

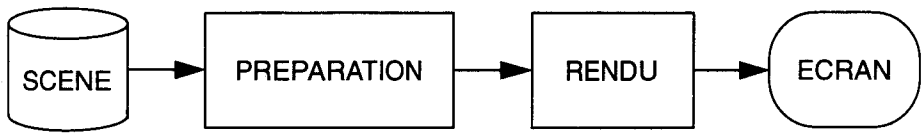


Figure 1.9 ***Le pipeline de rendu de Gouraud***

La préparation prend en charge les calculs géométriques et l'éclairement. Elle transforme les objets 3D en primitives 2D en conservant néanmoins une information de profondeur. Elle se décompose en cinq étapes présentées Figure 1.10.

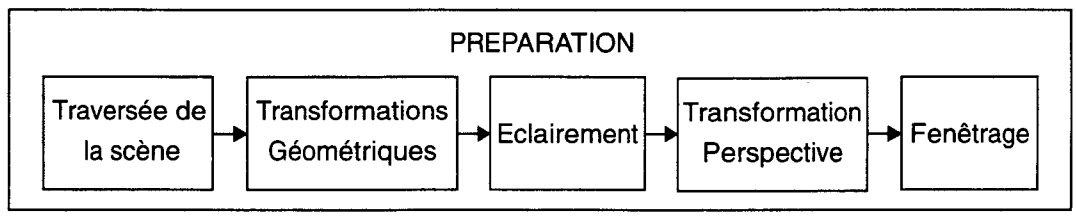


Figure 1.10 ***Phase de préparation***

Lors de la phase de rendu, l'image est calculée pixel par pixel. Pour cela il faut déterminer en chaque pixel quel objet est vu (s'il y en a un) et en déduire la couleur du pixel. On procède en trois étapes (Figure 1.11), plus éventuellement une étape pour le placage de texture (cf. paragraphe 3.5).

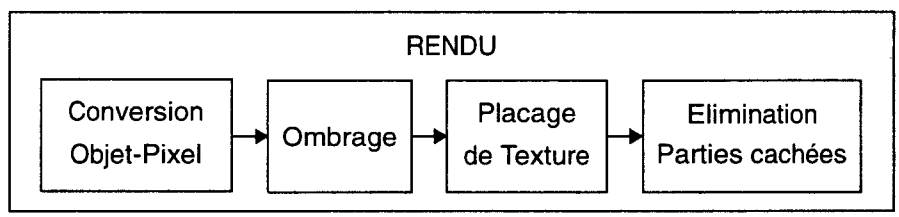


Figure 1.11 ***Phase de rendu***

Matériel spécifique

Au coeur des accélérateurs graphiques se trouvent les interpolateurs qui permettent la conversion objet-pixel. Ils sont utilisés pour déterminer la projection de l'objet sur les pixels de l'écran et pour chaque pixel recouvert, la profondeur et la couleur de l'objet (au minimum). Ces interpolateurs calculent une expression du premier degré en fonction des coordonnées du pixel. Rappelons qu'il est possible de réaliser ce calcul de façon incrémentale sur des nombres entiers. L'interpolateur se réduit alors à un simple additionneur entier. Le coût silicium est ainsi limité au maximum.

Toutes les parties du processus de synthèse d'image ont été étudiées et on trouve aujourd'hui des machines très performantes [SGI92] dans lesquelles tout est dédié. Cela concerne la phase de préparation (cf. chapitre quatre), celle de rendu ainsi que l'algorithme d'élimination des

parties cachées (Z-buffer). De plus on utilise de la mémoire spécifique pour accéder aux données plus rapidement en liaison avec des réseaux d'interconnections haut débit.

Pour accroître les performances plusieurs processeurs sont utilisés en parallèle. Un aperçu des possibilités est donné au paragraphe 4.3. Une étude plus complète du parallélisme dans les accélérateurs graphiques est réalisée dans [Karpf93].

L'association de la facette comme primitive d'affichage en rendu projectif et de processeurs dédiés a fait ses preuves. Les systèmes commerciaux actuels de synthèse d'images sont tous basés sur ces principes. Toutefois de nombreuses études sont encore poursuivies afin d'accroître la complexité et la qualité des images produites.

1.3 Relation entre modélisation et visualisation

La modélisation et la visualisation sont deux domaines dont les objectifs sont très différents. Il n'en reste pas moins que le premier fournit les données en entrée au deuxième et que le deuxième fournit une présentation visuelle du modèle. La difficulté principale est d'afficher ce que l'on modélise.

En effet les primitives de modélisation sont souvent relativement complexes afin de pouvoir rendre compte des objets de forme quelconque. En fonction de l'algorithme de rendu utilisé il n'est pas toujours facile, voire possible, d'afficher ces primitives. Nous montrons les difficultés existantes et les manières de les résoudre. Nous nous intéressons plus particulièrement au rendu projectif temps réel. Nous mettons ainsi en exergue la notion de *primitive de visualisation*.

En lancer de rayon toute primitive dont on peut calculer l'intersection avec une demi-droite peut être gérée. Le lancer de rayon permet donc de traiter la plupart des primitives de modélisation. Toutefois les calculs peuvent parfois être compliqués. De plus dans ce dernier cas l'algorithme d'intersection est sensible aux erreurs numériques et peut "rater" certaines intersections (cas de rayons rasant la primitive ou difficulté de détecter les singularités de certaines primitives).

En rendu voxel, toute primitive de modélisation doit être convertie en un ensemble de voxels (on parle de *voxelisation*). La voxelisation est une opération souvent longue et très souvent délicate. Par exemple comment voxeliser une surface plane continue en un ensemble discret de voxels volumiques ? Des algorithmes spécifiques de voxelisation sont développés pour le plan mais pour les primitives plus complexes le problème est en général ouvert. C'est pour cette raison que le rendu voxel reste principalement cantonné dans le domaine médical où les données en entrée sont déjà sous forme voxel.

En rendu projectif, seuls les polygones sont rendus efficacement. Certains algorithmes pour visualiser les primitives de modélisation existent mais ne sont pas efficaces. De plus, dès que l'on a besoin d'affichage rapide, il faut utiliser un accélérateur graphique qui de toute façon ne sait afficher que des facettes (et encore le plus souvent uniquement triangulaires). Les primitives de modélisation ne peuvent pas être affichées directement.

On voit donc apparaître une distinction entre *primitive de modélisation* et ce que nous appelons *primitive de visualisation* ou *primitive de rendu*. La première est l'entité de base utilisée pour décrire un objet. La deuxième est l'entité de base qui peut être affichée, indépendamment de toute considération de modélisation. La question qui vient alors à l'esprit est : comment passer de l'une à l'autre ? En d'autres termes comment combler l'écart entre primitive de modélisation et primitive de visualisation ?

Dans le cadre du rendu projectif temps-réel, la réponse aujourd'hui apportée réside dans les algorithmes de facettisation [Preparata88]. L'objectif est d'approcher par un ensemble de facettes polygonales (en général triangulaires) un objet décrit par des primitives de

modélisation. Ces algorithmes sont coûteux et font souvent perdre l'interactivité en modélisation. De plus ce n'est plus l'objet qui est rendu mais une approximation de l'objet.

Ces algorithmes sont aujourd'hui relativement bien maîtrisés pour les surfaces paramétriques. Toutefois les surfaces implicites souffrent du fait qu'elles sont difficiles à facettiser, ce qui rend difficile (et lent) leur affichage et interdit une utilisation directe des accélérateurs matériels. Pour les objets CSG il faut effectuer un passage CSG-B-rep puis un rendu classique éventuellement avec accélérateur. Pour éviter le passage d'une représentation à une autre, ce qui est pénalisant au niveau du temps d'affichage, des méthodes de *rendu CSG direct* ont été développées (cf. paragraphe 2.3.2) mais on ne peut plus utiliser d'accélérateur classique et donc L'affichage se fait par logiciel, ce qui est relativement lent. Dans un cas comme dans l'autre l'interactivité s'en trouve dégradée.

1.4 Une alternative à la facette : la quadrique

Dans le cadre du rendu rapide, l'écart entre primitive de modélisation et primitive de visualisation est habituellement comblé par une facettisation des objets. Toutefois cette technique est parfois délicate à mettre en oeuvre et toujours pénalisante pour l'interactivité.

De plus, l'emploi de la facette comme primitive de visualisation fait apparaître des défauts spécifiques. Ces défauts sont bien connus et malgré de très nombreuses études, certains persistent. Nous les décrivons puis nous montrons qu'un moyen pour les éviter est de remplacer la facette par une primitive de visualisation "courbe". Nous expliquons alors pourquoi la quadrique est une alternative pertinente.

1.4.1 Défauts dus aux facettes

Les faiblesses de la facette en synthèse d'images sont bien connues. La principale est l'approximation géométrique qui se traduit par un aspect anguleux des objets facettisés. Pour réduire ce défaut on augmente le nombre de facettes (Voir photos en annexe 6.4). Toutefois le problème n'est pas résolu, on ne fait que le reculer. De plus, facettiser finement accroit le nombre de primitives ce qui entraine une augmentation de la taille des bases de données et allonge les temps de calcul pour le rendu.

D'autres problèmes moins gênants existent. L'éclairement, notamment pour la composante spéculaire, souffre du fait que l'on utilise une interpolation de couleur. On peut noter dans certains cas des anomalies dues au fait que l'interpolation de couleur est effectuée après la transformation perspective. Le résultat de l'interpolation n'est pas indépendant de l'orientation de la facette. Des discontinuités dans l'éclairement peuvent apparaître à certains sommets partagés. Une solution à ces deux problèmes consiste à n'utiliser que des facettes triangulaires ce qui suppose un découpage des facettes polygonales (autre que les triangles) qui augmente d'autant plus le nombre de primitives. Enfin, dans certains cas les normales calculées en chaque sommet de facette ne représentent pas correctement la géométrie de la surface approchée, ce qui se traduit par des erreurs d'éclairement.

1.4.2 La quadrique

La principale limite de la facette est donc la qualité des contours des objets. Une solution est de changer de primitive de visualisation au profit d'une autre qui puisse être courbe. Une première idée est de prendre une primitive utilisée en modélisation. Ainsi on peut afficher directement à l'écran les objets que l'on modélise sans passer par une représentation intermédiaire qui entraine une perte de qualité. De plus on évite la phase de transformation des objets modélisés dans cette représentation intermédiaire (typiquement la phase de facettisation) qui souvent ruine l'interactivité. Toutefois cette solution reste peu envisageable pour un rendu temps-réel

car les primitives de modélisation sont trop complexes, ce qui implique des algorithmes compliqués qui ne peuvent pas être implémentés en temps-réel pour un coût raisonnable.

En effet une leçon à tirer de l'utilisation de la facette est que pour pouvoir concevoir des processeurs spécialisés efficaces, il faut que les algorithmes à implanter soient simples. De plus, pour des raisons techniques, la primitive doit pouvoir se manipuler sous sa forme implicite. Nous avons donc choisi la famille des quadriques car ce sont des surfaces courbes. On y trouve notamment la sphère, le cylindre et le cône. De plus ce sont les surfaces mathématiquement les plus simples : en effet l'équation d'une quadrique est du second degré en x, y, z.

1.5 Bilan

Aujourd'hui, pour produire des images en temps réel, on utilise un accélérateur graphique. Celui-ci n'accepte que la facette comme primitive de visualisation. Il y a alors un décalage entre ce qui est produit en modélisation et ce qui est rendu. En d'autres termes, il y a un écart entre primitive de modélisation et primitive de visualisation. Pour combler cet écart les algorithmes de facettisation sont habituellement utilisés.

L'emploi de la facette comme primitive de visualisation, même si elle s'explique très bien, entraîne néanmoins des défauts au niveau de la qualité de l'image produite. Principalement, les objets ont un aspect anguleux, facettisé.

Nous proposons dans cette thèse, de remettre en cause l'emploi de la facette comme primitive de visualisation. La facette est remplacée par une autre primitive, cette fois-ci "courbe". Pour des raisons de simplicité nous avons choisi la quadrique.

Avant de définir un système de visualisation temps-réel basé sur la quadrique il est nécessaire de bien connaître la quadrique. Objet mathématique simple, la quadrique a déjà été utilisée en informatique graphique. L'objet du chapitre suivant est donc d'établir un état de l'art sur la primitive quadrique en synthèse d'image.

Etat de l'art sur la quadrique

La première démarche pour concevoir un système graphique basé sur la quadrique consiste à mettre en évidence les questions essentielles. Par exemple, qu'est ce qu'une quadrique ? Peut-on modéliser avec les quadriques et comment ? Peut on afficher des quadriques et avec quels algorithmes ?

Les réponses à toutes ces questions peuvent être organisées en trois parties. Tout d'abord une étude mathématique. Notre but ici n'est pas de faire une étude exhaustive mais de donner les bases nécessaires afin de bien appréhender les problèmes et les solutions abordées dans le reste de ce mémoire. Ensuite nous présentons les différentes voies intéressantes pour la modélisation. Enfin nous décrivons les études antérieures sur l'affichage des quadriques.

Cet état de l'art ne serait pas complet si nous ne revenions pas sur les différentes études menées au LIFL. Après la thèse d'Eric Nyiri [Nyiri94], les recherches se sont scindées en deux thèmes : la modélisation avec la thèse de Maxime Froumentin [Froumentin96] et la visualisation qui fait l'objet de cette thèse.

2.1 Présentation mathématique des quadriques

Il existe plusieurs façons de représenter les quadriques. Nous allons nous intéresser plus particulièrement à la représentation algébrique car nous utiliserons celle-ci dans la suite de cet exposé. Toutefois il existe d'autres représentations qui offrent certains avantages. Les représentations géométriques et paramétriques sont bien connues et permettent de connaître facilement le type de quadrique. De récents travaux en CAO on fait apparaître d'autres représentations comme celle de Bernstein-Bézier qui sont utiles pour modéliser à partir des quadriques.

2.1.1 Représentation algébrique

C'est la représentation habituelle des quadriques. De plus c'est sous cette forme que nous allons les manipuler dans nos algorithmes de rendu, notamment pour exprimer les profondeurs en un pixel donné et la normale en un point de la quadrique.

Définition

Soit l'expression quadratique en x, y, z :

$$Q(x, y, z) = ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j. \qquad \textbf{(Eq. 2.1)}$$

Une quadrique notée Q est l'ensemble des points vérifiant :

$$Q(x, y, z) = 0 \qquad \textbf{(Eq. 2.2)}$$

Les dix coefficients $a, b, ..., j$ définissent la quadrique. Les quadriques sont un sous-ensemble des surfaces algébriques, définies par l'équation $P(x, y, z) = 0$ où P est un polynôme. Les surfaces algébriques sont elles-mêmes un sous-ensemble des surfaces implicites définies par l'équation $f(x, y, z) = 0$ où f est une fonction quelconque.

La normale (utile pour le calcul d'éclairement, cf. annexe 6.1) d'une quadrique s'exprime directement à partir de son équation. En effet les composantes de la normale de la quadrique $Q(x, y, z)$ en un point (x, y, z) sont les dérivées partielles de l'équation (Eq. 2.1). On a :

$$\begin{cases} N_x = \frac{\partial}{\partial x} Q(x, y, z) = 2ax + dy + ez + g \\ N_y = \frac{\partial}{\partial y} Q(x, y, z) = 2by + dx + fz + h \\ N_z = \frac{\partial}{\partial z} Q(x, y, z) = 2cz + ex + fy + i \end{cases} \quad \textbf{(Eq. 2.3)}$$

Notons que la normale s'exprime simplement en fonction des coefficients qui définissent la quadrique. Cette propriété influencera notre décision lors du choix de l'architecture de notre système.

Classification

Les quadriques sont une famille de surfaces tri-dimensionnelles. Les plus connues sont la sphère, qui est un cas particulier de l'ellipsoïde, le cylindre (elliptique) et le cône. On citera également le paraboloïde, l'hyperboloïde à une ou deux nappes, le paraboloïde hyperbolique et le cylindre parabolique ou hyperbolique (Figure 2.1 et Figure 2.2). Les autres quadriques sont des paires de plans. Pour une classification exhaustive des différents types de quadriques voir [Levin76] ou [Kleij93].

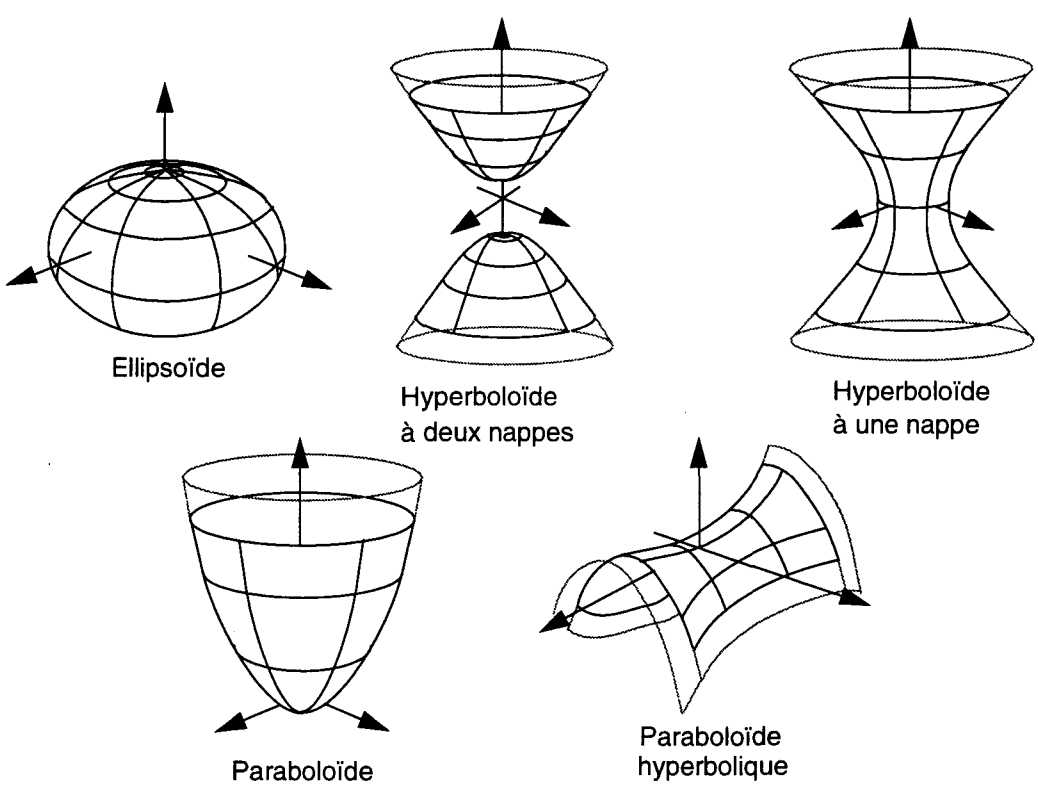


Figure 2.1 ***Quadriques non dégénérées***

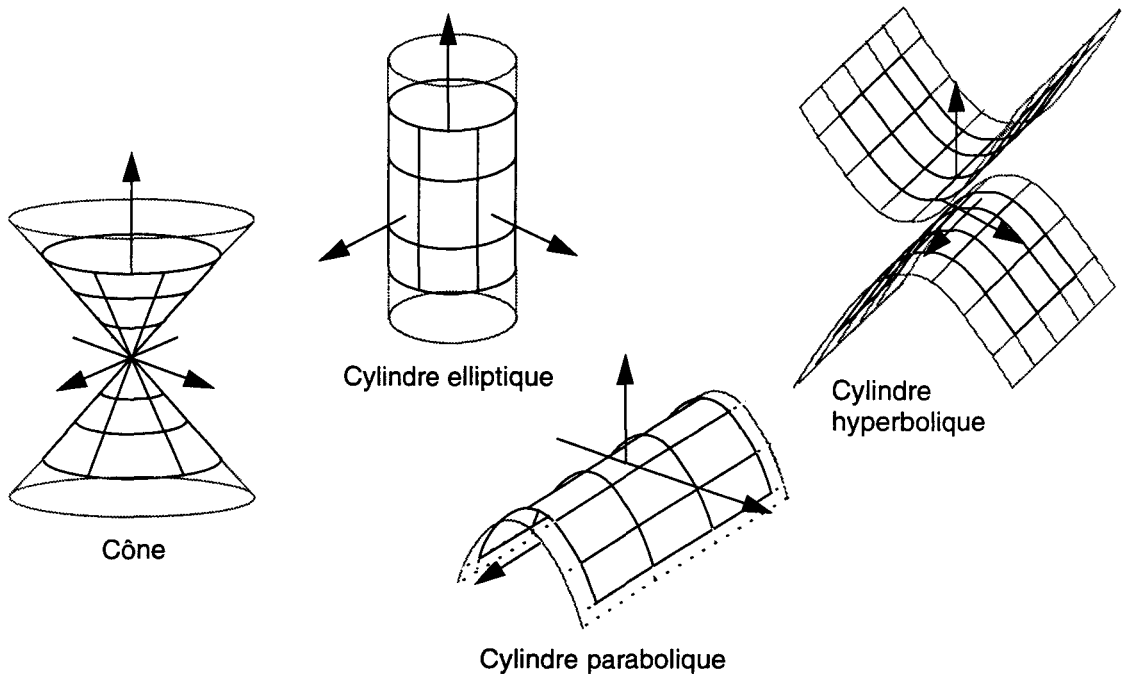


Figure 2.2 ***Quadriques dégénérées du premier ordre***

Forme matricielle et opération sur les quadriques

L'équation (Eq. 2.2) peut également s'écrire sous la forme:

$$ {}^tXQX = 0 \qquad \textbf{(Eq. 2.4)} $$

avec ${}^tX = (x, y, z, 1)$ et $Q = \begin{bmatrix} a & d/2 & e/2 & f/2 \\ d/2 & b & g/2 & h/2 \\ e/2 & g/2 & c & i/2 \\ f/2 & h/2 & i/2 & j \end{bmatrix}$

Cette notation bien que redondante permet de réaliser facilement les transformations géométriques sur les quadriques, au moyen du calcul matriciel. En effet, soit M une matrice de transformation et Q' la matrice transformée de Q par M. On écrira :

$$ Q' = M^{-1} \cdot Q \cdot M^{-1\,t} \qquad \textbf{(Eq. 2.5)} $$

Si l'on travaille dans un espace projectif en coordonnées homogènes toutes les transformations habituellement utilisées pour le rendu peuvent se représenter sous forme de matrices 4×4. Il s'agit de la translation, de la rotation, de la mise à l'échelle et de la transformation perspective. Toutes ces opérations sont closes sur la famille des quadriques ce qui nous assure de ne travailler que sur des quadriques quelles que soient les transformations réalisées.

2.1.2 Autres représentations

Représentation géométrique

Cette représentation, proposée par R. Goldman [Goldman83], a pour but d'éviter les défauts posés par les équations algébriques, en particulier ceux dus au codage avec des nombres flottants ayant une précision limitée. En effet dans certains cas il n'est pas possible de déterminer le type de la quadrique ou de savoir si deux quadriques sont égales à cause des erreurs d'arrondis.

Chaque quadrique est représentée par un ensemble de données : tout d'abord, un point de l'espace, qui permet de fixer la position de la quadrique ; ensuite deux vecteurs qui fixent l'orientation de la surface ; puis trois coefficients d'échelle en x, y et z ; enfin quatre bits qui codent le type de la quadrique.

Représentation paramétrique

Toutes les quadriques peuvent être paramétrées soit par des fonctions trigonométriques, soit par des polynômes rationnels. Toutefois on a une paramétrisation différente par type de quadrique (Table 2.1:). En effet il n'existe pas de paramétrisation générale qui puisse décrire tous les types de quadriques et uniquement les quadriques. De plus pour certaines d'entre elles l'espace des paramètres est infini ce qui empêche leur utilisation dans un cadre pratique.

Type de quadrique	Equations paramétriques	Type de quadrique	Equations paramétriques
Ellipsoïde	$x = a \times \cos\theta \times \cos\varphi$ $y = b \times \cos\theta \times \sin\varphi$ $z = c \times \sin\theta$	Cylindre elliptique	$x = a \times \cos\theta$ $y = b \times \sin\theta$
Hyperboloïde à une nappe	$x = a \times \cos\theta \times \cosh\varphi$ $y = b \times \sinh\varphi$ $z = c \times \sin\theta\cosh\varphi$	Cylindre hyperbolique	$x = a\cos\varphi$ $y = b \times \sin\varphi$
Hyperboloïde à deux nappes	$x = a \times \cos\theta \times \sinh\varphi$ $y = b \times \cosh\varphi$ $z = c \times \sin\theta \times \sinh\varphi$	Cylindre parabolique	$x = a \times t$ $y = t^2/b$
Paraboloïde	$x = a \times t \times \cos\varphi$ $y = t^2$ $z = c \times t \times \sin\theta$	Cône	$x = a \times t \times \cos\theta$ $y = b \times t \times \sin\theta$ $z = c \times t$
Paraboloïde hyperbolique	$x = a \times t \times \cosh\theta$ $y = t^2$ $z = c \times t \times \sinh\theta$		

Table 2.1: Equations paramétriques des différentes quadriques

Représentation de Bernstein-Bézier

Sederberg [Sederberg85] a introduit une nouvelle représentation pour les surfaces implicites, que nous présentons uniquement pour le cas particuliers de la quadrique : soit V un tétraèdre

non dégénéré et $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ les coordonnées barycentriques d'un point par rapport à V. Une quadrique peut alors s'exprimer comme le lieu des points vérifiant :

$$\sum_{i+j+k+l=2} \frac{2}{i!j!k!l!} b_{i,j,k,l} \lambda_0^i \lambda_1^j \lambda_2^k \lambda_3^l = 0 \qquad \textbf{(Eq. 2.6)}$$

La quadrique est toujours définie par dix coefficients b_{ijkl}.

L'expression de gauche dans l'équation (Eq. 2.6) est une fonction de l'espace dont le graphe est une hyper-surface 4D représentée sous la forme d'une Bézier fonctionnelle dont l'équation est

$$\sum_{i+j+k+l} B^2_{i,j,k,l}(\lambda_1, \lambda_2, \lambda_3, \lambda_4) P_{i,j,k,l} = 0 \qquad \textbf{(Eq. 2.7)}$$

En identifiant les équations (Eq. 2.6) et (Eq. 2.7) on peut exprimer les coordonnées des points de contrôle de la Bézier (les quatre premières coordonnées sont barycentriques, la cinquième est cartésienne) :

$$P_{i,j,k,l} = (i/2, j/2, k/2, l/2, b_{i,j,k,l}) \qquad \textbf{(Eq. 2.8)}$$

En projetant les points de contrôle de la Bézier dans l'espace cartésien et en utilisant la dernière coordonnée comme coefficient de pondération, on obtient une espèce de polyèdre de contrôle de notre quadrique (en l'occurrence le tétraèdre). Sur la Figure 2.3 on a placé les dix points de contrôle (de coordonnées barycentriques $(i/2, j/2, k/2, l/2)$) avec la valeur des coefficients associés.

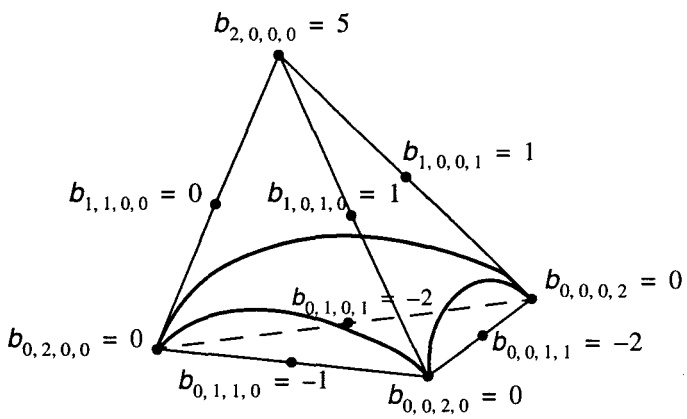


Figure 2.3 ***Un patch quadrique et ses coefficients sous sa représentation de Bernstein-Bézier***

Une quadrique peut donc être définie à partir de la Bézier ci-dessus. En effet la quadrique est l'intersection de cette Bézier avec l'espace cartésien 3D ($w = 0$). On peut donc profiter des bonnes propriétés des Bézier pour la modélisation notamment pour les raccordements et le contrôle de la continuité.

Dans la pratique on ne s'intéresse qu'au morceau de quadrique qui se trouve dans le tétraèdre. Se dégage alors la notion de *patch quadrique*.

Sous cette forme, les coefficients de la quadrique ont une signification géométrique : leur valeur influe sur la forme de la quadrique au voisinage du point considéré. L'utilisateur peut alors utiliser ces coefficients de façon intuitive. Par exemple pour interpoler un point de contrôle il suffit que le coefficient qui lui est associé soit nul.

2.2 Modélisation à base de quadriques

L'utilisation des quadriques naturelles en modélisation géométrique est largement répandue dans des domaines tels que la modélisation moléculaire ou CSG. Mais nous ne pouvons nous contenter de ces restrictions. Nous devons évaluer l'intérêt de la quadrique dans tous les types de modélisation : surfaces non contraintes, approximation etc.

Nous présentons brièvement les diverses façons d'aborder la modélisation à partir des quadriques. Deux grandes classes de méthodes apparaissent (Figure 2.4) : la première consiste à créer les objets d'une scène directement avec des quadriques (flèche 1) ; la deuxième consiste à transformer des objets complexes en un ensemble de morceaux de quadriques (flèche 2). Une étude plus complète se trouve dans [Froumentin96].

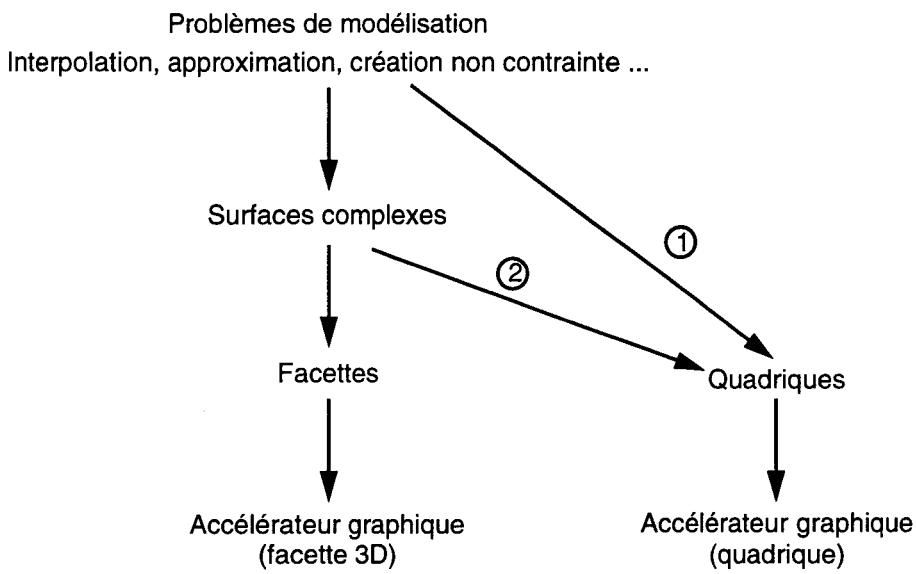


Figure 2.4 ***Deux approches pour modéliser avec des quadriques***

Un autre type de problème que l'on peut ranger en modélisation est le passage de la représentation CSG à une représentation B-Rep (cf paragraphe 1.1.2). Ce problème est apparu avec l'émergence au début des années soixante-dix, des premiers modeleurs CSG qui ont tout de suite inclus la quadrique comme primitive (voir l'historique de [Requicha82]).

Pour le résoudre, il faut être en mesure de calculer les courbes d'intersections entre deux primitives CSG, a priori des polyèdres (ou plus généralement des demi-espaces séparés par des plans) et des quadriques. Les courbes d'intersections sont rangées dans une structure de données adéquate pour avoir les notions de bord et de surface.

La difficulté principale de cette conversion réside dans le calcul de l'intersection de deux quadriques qui peut être une quartique (une courbe 3D de degré quatre). Les premiers travaux ont été réalisés sur des modèles non CSG utilisant la quadrique, pour lesquels on cherchait à produire une image filaire. Plus tard les travaux de Levin ont permis de résoudre complètement le problème. Cette méthode a été améliorée ensuite par Sarraga pour le modeleur GMSolid. Enfin Miller a proposé une méthode géométrique à ce problème. Tous ces travaux, intimement liés à la visualisation, sont présentés au paragraphe 2.3.1.

Récemment, Menon [Menon94] a proposé une solution pour des objets volumiques dont la surface est non contrainte. La surface de l'objet est approchée par un ensemble de patchs quadriques selon la représentation de Bernstein-Bézier. Pour obtenir un volume, il considère

pour chaque patch le volume défini par l'intersection du tétraèdre et de la quadrique, qu'il appelle un *trunctet*. La réunion de ces trunctets sert de base pour définir l'objet volumique. Il a ainsi une représentation hybride (appelée *CSR* ou *Constructive Shell Representation*) qui lui permet de passer aisément d'une représentation CSG à une representation B-Rep.

2.2.1 Modélisation directe

Nous décrivons ici les différentes approches permettant de créer des objets explicitement à partir de quadriques. La première approche consiste à utiliser des quadriques naturelles, qui sont manipulables de façon intuitive. Toutefois cette approche n'utilise pas toutes les quadriques et ne s'avère utile que pour certaines classes d'applications spécifiques.

On peut ensuite proposer à l'utilisateur de spécifier des points (avec éventuellement des normales) qui vont fixer des contraintes à partir desquelles on calcule les quadriques. Dans ce cas on ne spécifie plus le type de la quadrique. Dans ce cadre deux approches ont été développées. La première utilise la représentation algébrique. Bien que plus souple que l'approche géométrique, elle est encore limitée. La seconde utilise la représentation de Bernstein-Bézier pour définir les *macropatchs*.

Les quadriques naturelles

Les quadriques naturelles sont le cône, le cylindre et la sphère (ou plus généralement l'ellipsoïde). Ces primitives sont définies par leur type, leur taille, leur position et orientation. A partir de ces informations on peut facilement retrouver leur équation implicite : le type et la taille définissent l'équation canonique, la position et l'orientation définissent une transformation qui lui est appliquée.

Certaines applications n'utilisent qu'un seul type de quadrique naturelle ; par exemple la sphère pour les molécules ou pour les *metaballs* [Nishimura85]. Bien sûr le calcul de l'équation implicite en est facilité.

Le domaine où les quadriques naturelles sont particulièrement utiles est la modélisation CSG. Comme nous l'avons déjà dit, celle-ci apparaissent dès les premiers modeleurs au début des années soixante-dix. Depuis on les trouve dans quasiment tous les modeleurs CSG. Leur intérêt réside dans le fait qu'elles correspondent à des formes particulièrement naturelles et intuitives pour l'homme. De plus elles correspondent très bien aux opérations effectuées pour fabriquer des pièces mécaniques. Par exemple forer un trou dans un bloc de matière correspond à soustraire un cylindre d'une autre primitive.

Utilisation de la représentation algébrique

Cette représentation a été utilisée pour modéliser avec une classe particulière de quadriques : les quadriques de révolution. Elle sert également dans la méthode de Pratt et celle de Bajaj et Ihm mais cette fois-ci dans le cas général.

Beaucoup d'objets sont des objets de révolution : une bouteille, un vase, un abat-jour de lampe (et éventuellement le pied de cette même lampe)... Un objet de révolution peut être intuitivement caractérisé par son axe de révolution et son profil, c'est-à-dire la courbe qui forme l'objet si elle balaye un cercle centré sur l'axe de révolution (Figure 2.5).

Pour une quadrique de révolution le profil est une conique. L'utilisateur peut donc spécifier ce profil avec trois point. De plus on limite la quadrique par des plans perpendiculaires à l'axe de révolution. On peut ensuite mettre bout à bout plusieurs morceaux de quadriques de révolution (Figure 2.5) avec une continuité G^0 ou G^1.

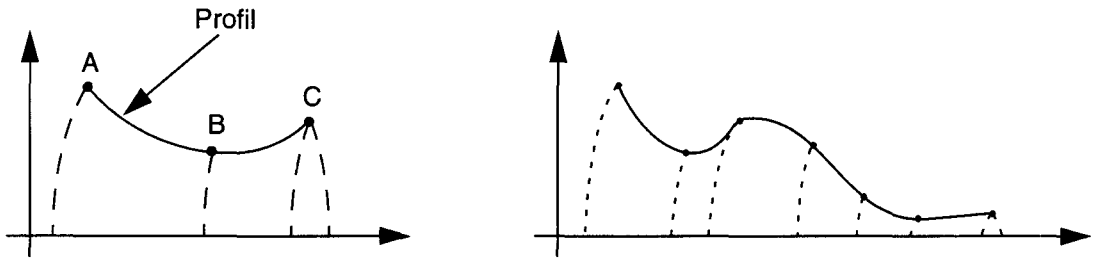


Figure 2.5 ***Quadrique de révolution : profil (à gauche) et interpolation en raccordant plusieurs morceaux de quadriques (à droite)***

Pratt [Pratt87] étudie les méthodes d'interpolation de points par une surface algébrique. Chaque point d'interpolation définit une équation. L'ensemble des équations forme un système dont la solution est la surface algébrique recherchée. Dans le cas particulier de la quadrique il faut neuf points.

Bajaj et Ihm [Bajaj92] étendent les résultats de Pratt à différentes contraintes d'interpolation : points avec ou sans normales et courbes algébriques ou paramétriques avec ou sans normales. Chaque contrainte fournit une ou plusieurs équations. L'ensemble des équations forment le système à résoudre.

D'un point de vue interactif il semble plus intéressant de travailler avec trois paires de points-normales. Toutefois il faut encore résoudre le système pour avoir la solution sous forme explicite. De plus ce système n'a pas toujours de solution. Enfin se pose le problème où la solution est une quadrique à deux nappes (dans ce cas la topologie de l'objet n'est plus respectée puisqu'il est alors constitué de deux parties).

Les macro-patchs

Différents auteurs [Dahmen89] [Guo93] se sont appuyés sur la représentation de Bernstein-Bézier pour résoudre le problème suivant : l'interpolation d'un nuage de points-normales dont on connaît la topologie (ie ces points font parties d'une triangulation). Pour cela ils cherchent à construire une surface complexe composée d'un ensemble de carreaux de surfaces algébriques de faible degré. Ces carreaux de surfaces sont regroupés en *macro-patchs*. Ces méthodes offrent l'avantage de ne pas à avoir à résoudre de système d'équations.

Dans les deux cas, l'idée est de partir de la triangulation. On associe à chaque triangle un certain nombre de carreaux quadriques qui forment un macro-patch. Ensuite il faut "boucher les trous" entre chaque macro-patch. Pour cela on utilise d'autres patchs appelés *filling patches*. Toutefois ces systèmes ne permettent pas toujours de trouver une solution. Les auteurs proposent une solution dans le cas général mais en utilisant des cubiques.

2.2.2 Conversion de surfaces complexes en quadriques

Aujourd'hui la plupart des modèles utilisés sont basés sur des surfaces complexes. Les surfaces paramétriques sont largement utilisées [Farin92] : Bézier, NURBS, etc et les surfaces implicites (et notamment algébriques) sont le sujet de plus en plus d'études. En général ces surfaces ne peuvent pas être représentées exactement avec des quadriques. En effet ces surfaces sont d'un degré élevé, comme par exemple un patch de Bézier bicubique, de degré dix-huit. Il nous faut donc trouver des algorithmes permettant d'approcher ces surfaces complexes au moyen d'un ensemble de quadriques. C'est l'équivalent de la facettisation. Deux grandes

classes de méthodes se distinguent : l'échantillonnage / construction[1] et l'approximation directe (Figure 2.6).

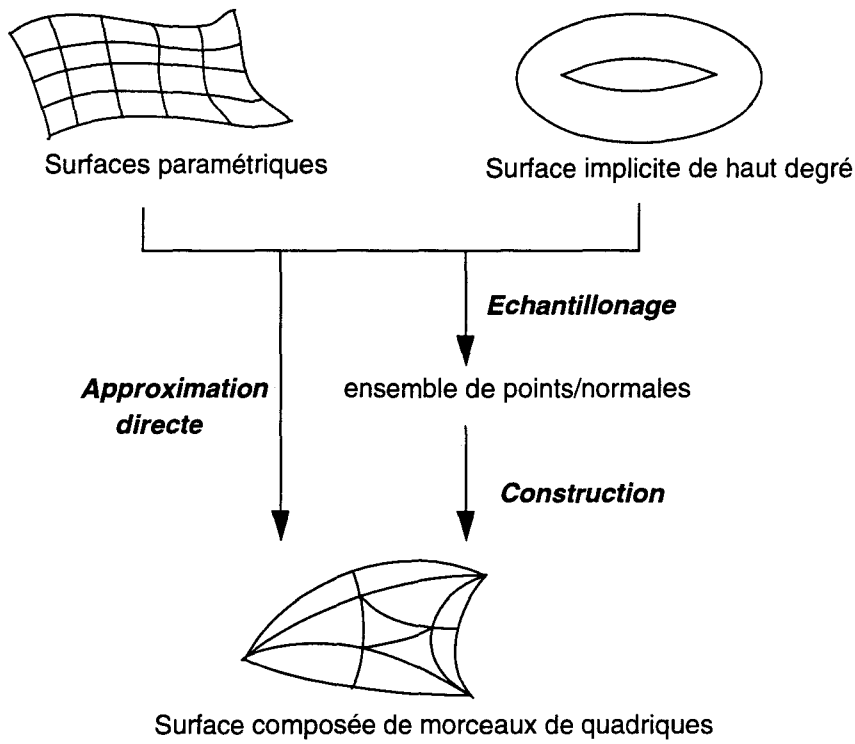


Figure 2.6 ***Décomposition de surfaces complexes en quadriques***

Echantillonnage et construction

Il est toujours envisageable d'échantillonner une surface complexe, éventuellement avec une information de normale. A partir de là on peut trouver une surface formée de morceaux de quadriques qui interpole ces points avec une continuité G^0 ou G^1. Cette méthode est bien connue lorsque l'on utilise des facettes et peut être adaptée à des quadriques grâce à des techniques proposées par [Pratt87], [Dahmen89], [Bajaj92] ou [Guo93]. On retombe dans les techniques d'interpolation présentées précédemment.

La méthode de construction dépend de la méthode d'échantillonnage qui elle-même dépend de la surface à convertir. Pour les surfaces paramétriques il est facile de produire un ensemble de points en échantillonnant dans l'espace des paramètres. Pour les surfaces implicites l'échantillonnage est réalisé par un partitionnement de l'espace [Ning93], mais cette technique souffre de la lenteur pour trouver les racines, de la mauvaise répartition des échantillons et de certaines ambiguïtés topologiques.

Approximation directe

Pour éviter des problèmes dus à la méthode d'échantillonnage, une approximation directe peut être envisagée. Bien que dépendante de la surface à convertir, cette méthode peut s'avérer plus rapide et plus précise. Aujourd'hui seules des méthodes utilisant des quadriques naturelles existent (comme les metaballs). Cette approche est développée par Maxime Froumentin (cf paragraphe 2.4.2).

1. Sampling / fitting

2.3 Visualisation de quadriques

Nous présentons tout d'abord les études relatives à l'affichage de quadriques en tant que surfaces. Les premières tentatives d'affichage de quadrique se sont heurtées au problème de la détermination de l'intersection entre deux quadriques. En effet, en visualisation on se base habituellement sur un objet décrit par une B-rep c'est à dire par ses bords et ses surfaces. Il faut donc être en mesure de déterminer ces informations.

La modélisation CSG offre un outil puissant en CAO mais pose la question de savoir comment produire une image d'un objet CSG. Pour y répondre, de nombreux efforts ont été déployés dans deux directions principales. La première consiste à convertir l'arbre CSG dans un modèle facilement affichable (typiquement sous forme de B-rep polygonale[1]). Le principe de la deuxième approche consiste à éviter de repasser par une représentation B-Rep : il s'agit du *rendu CSG direct*. Nous nous intéressons plus particulièrement à cette approche. En effet, contrairement à la première, elle traite directement des quadriques.

Enfin nous présentons les quelques études de machines utilisant les quadriques ou un sous-ensemble des quadriques. Les deux plus connues sont la machine Pixel Planes-5 et la Ray Casting Machine. Par ailleurs on a vu apparaître depuis peu une carte accélératrice pour compatibles-PC qui gère des quadriques.

Remarquons que les quadriques ont également été utilisées dans certaines applications spécifiques. Porter [Porter78] et Max [Max83] se servent des sphères et des cylindres en simulation moléculaire pour visualiser le modèle classique *stick-and-ball*. Herbison-Evans [Herbison82] représente le corps humains avec des ellipsoïdes. Gardner [Gardner84] [Gardner85] visualise des arbres et des paysages de collines ou bien des ciels nuageux au moyen de quadriques et de fonction de textures. Plus récemment Max [Max90] utilise des *cône-sphères* (deux sphères reliées par une portion de cône ou de cylindre tangent aux deux sphères) pour créer des objets de forme tubulaire tels que des racines d'arbres. Enfin dans le jeu Ecstetica de chez Psygnosis, les personnages sont réalisés avec des ellipsoïdes.

2.3.1 Etudes logicielles

Les premières méthodes pour afficher des quadriques n'ont pas résolu le problème de l'intersection de deux quadriques mais l'ont contourné. En effet ces méthodes ne sont pas capables de déterminer formellement l'expression de l'intersection mais sont capables de calculer un ensemble de points pour l'affichage. Le premier a avoir réellement calculé l'intersection de deux quadriques dans le cas général est Levin qui l'exprime sous forme paramétrique. Sa méthode a ensuite été reprise par Sarraga dans le modeleur GMSolid. Enfin Miller a proposé une méthode géométrique qui offre certains avantages notamment au niveau de la précision des calculs mais qui ne traite pas toutes les quadriques.

Méthode de Weiss

Weiss a développé un programme d'affichage de vue orthographique d'objets définis à partir de plans et de quadriques [Weiss66]. Il a tout d'abord défini ce que nous appellerons le *patch de Weiss*. Ensuite il a proposé un algorithme de visualisation qui comprend trois phases principales.

Le patch de Weiss est un ensemble de surfaces planes ou quadriques qui peuvent être de deux types : les *surfaces principales* qui définissent l'objet à proprement parler et les *surfaces englobantes* qui limitent les surfaces principales. Une surface principale peut être utilisée seule ; elle est visible et elle a des intersections visibles avec les autres surfaces principales et les surfaces englobantes qui lui sont associées. Une surface englobante doit être associée à une surface principale, est invisible et n'a d'intersections visibles qu'avec sa surface principale.

1. Pour les B-rep non polygonales voir l'introduction du paragraphe 2.2

Pour le rendu, Weiss distingue trois types de segments de courbes : l'intersection de deux surfaces principales, l'intersection d'une surface principale avec une de ses surfaces englobantes et la silhouette d'une surface principale. Ces segments sont appelés respectivement intersection, englobant et bord (cf Figure 2.7).

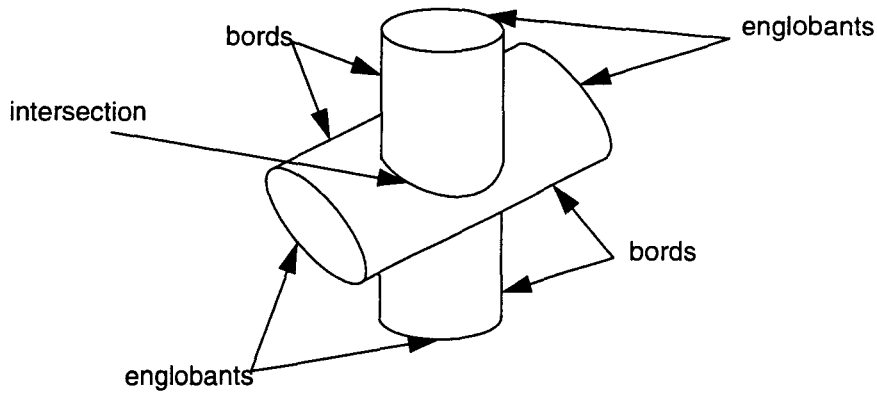


Figure 2.7 ***Patch de Weiss***

Pour la visualisation Weiss commence par déterminer des minima et maxima en x, y et z pour chaque surface. Pour cela il détermine cinq cas pour lesquels il peut donner une contrainte sur les coordonnées. Par exemple pour un cylindre de rayon R et d'axe z, on a $xmin=ymin=-R$ et $xmax=ymax=R$.

Ensuite il affiche les intersections et les englobants de la même façon (l'affichage des bords ne nécessite pas d'intersection quadrique-quadrique, il n'est donc pas détaillé). Pour cela il détermine un ensemble de points de la courbe.

A partir des équations des deux quadriques il utilise la méthode de Sylvester pour éliminer une variable (supposons x). Il obtient ainsi une équation du quatrième degré en y et z. Il balaye alors l'intervalle de validité d'une de ces deux variables (par exemple $[zmin, zmax]$ pour z). Pour chaque z, il résout alors l'équation quartique en y et obtient donc quatre valeurs dans le cas général. Chacune des quatre valeurs (avec son z correspondant) est tour à tour réinjectée dans les équations des deux quadriques que l'on résout en x. Si pour les deux quadriques on trouve des valeurs de x communes c'est que l'on a trouvé un point de la courbe d'intersection.

Pour l'élimination des parties cachées il teste chaque point par rapport à toutes les autres surfaces ce qui, comme le reconnaît Weiss, est fort coûteux.

Méthode de Woon et Freeman

Woon et Freeman [Woon71] ont développé une autre méthode pour le rendu d'objets quadriques identiques aux patchs de Weiss. Tout d'abord ils caractérisent l'objet en termes de sommets, arêtes et faces. Cette caractérisation est dépendante de la vue. Elle permet d'utiliser ensuite une méthode d'élimination des parties cachées dont le coût est réduit par rapport à celle utilisée dans l'algorithme de Weiss).

Le problème de l'intersection est résolu en calculant un ensemble de points de la courbe de façon itérative. En supposant que l'on a calculé k points de la courbe on cherche le $k+1$ ième[1]. Pour cela on va chercher à avancer d'un (petit) incrément en x ou y ou z selon la direction principale du vecteur donné par le k ième et le $(k-1)$ ième point. En réinjectant cette valeur dans

1. Notons que les auteurs n'indiquent pas comment amorcer l'algorithme.

les équations des deux quadriques, on obtient un système de deux équations quadratiques à deux variables. Ce système est alors résolu par la méthode de Newton.

L'élimination des parties cachées est basé sur l'algorithme de Loutrel [Loutrel70]. En premier lieu on détermine trivialement certaines arêtes assurément invisibles (par exemple une arête appartenant à deux faces arrières) qui ne sont donc pas prises en compte par la suite. Ensuite on utilise la notion d'*ordre de visibilité*. C'est un nombre positif ou nul indiquant le nombre de faces avant entre le point considéré et l'observateur. Lorsque l'on parcourt une arête de point en point on peut croiser une autre arête. Selon le type de cette arête l'ordre de visibilité est incrémenté ou décrémenté. Si l'ordre de visibilité est zéro le point est visible. Pour diminuer le nombre de tests on peut mémoriser l'ordre de visibilité des sommets communs à deux arêtes. On peut de plus utiliser des boîtes englobantes sur les projections des arêtes. Deux arêtes dont les boîtes englobantes sont disjointes n'ont pas d'intersection.

Méthode de Mahl

Mahl [Mahl72] effectue un rendu surfacique sur des patchs de Weiss. Pour cela il utilise une technique à balayage de lignes [Watkins70]. Chaque quadrique est coupée par le plan associé à la ligne de balayage considérée. Le problème de l'intersection de deux quadriques est donc ramené à l'intersection de deux coniques. Il propose alors deux algorithmes pour déterminer les parties visibles.

Le premier algorithme travaille sur des patchs qui n'ont pas d'intersection. De la même façon que Watkins il définit des intervalles pour lesquels un seul segment de conique est visible. Pour cela il détermine pour chaque patch des points spéciaux qui sont de deux types (Figure 2.8) : les points de la silhouette (points x_1 et x_4) et les points d'intersections de la conique et de ses englobants (points x_2 et x_3). Ces points une fois triés en x forment les extrémités des intervalles. Pour un intervalle donné il détermine le segment de conique visible en effectuant un tri en profondeur sur le point milieu de l'intervalle.

Lorsque les patchs ont une intersection, il calcule les points d'intersection et subdivise l'intervalle (Figure 2.8). Pour réduire le nombre d'intersections entre deux coniques à calculer il propose plusieurs optimisations. Finalement il ne lui reste qu'à trier en profondeur.

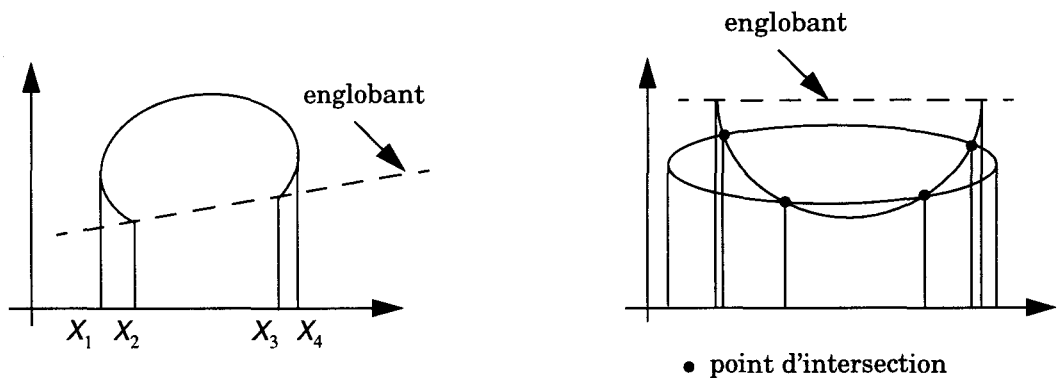


Figure 2.8 ***Patchs avec (à droite) et sans intersection (à gauche)***

Méthode de Levin, méthode de Sarraga

Levin a été le premier à proposer une méthode donnant l'expression de la courbe d'intersection de deux quadriques Q_1 et Q_2 sous forme paramétrique [Levin76]. Pour cela, il utilise leur *faisceau* défini comme étant l'ensemble de leurs combinaisons linéaires :

$$F(\lambda) = Q_1 + \lambda Q_2 \qquad \textbf{(Eq. 2.9)}$$

Il utilise ensuite deux propriétés intéressantes. Primo si Q_1 et Q_2 ont une intersection non nulle alors toute surface du faisceau partage cette intersection. Secundo il existe au moins une quadrique réglée dans le faisceau. Les surfaces réglées sont des surfaces qui peuvent être considérées comme une collection de lignes droites (par exemple un cylindre ou un cône). Elles sont intéressantes car elles ont une paramétrisation simple. Par exemple pour un cylindre de rayon R on a :

$$\begin{aligned} x &= R\cos t \\ y &= R\sin t \\ z &= s \end{aligned} \qquad \text{(Eq. 2.10)}$$

Levin est capable de déterminer la quadrique réglée la plus simple dans le faisceau. Il peut alors trouver un repère dans lequel cette surface a une paramétrisation simple du type de l'équation (Eq. 2.10). L'une des deux quadriques (Q_1 ou Q_2) est exprimée dans ce repère. On peut alors y substituer x, y et z par leur expressions paramétriques. On obtient ainsi la courbe d'intersection de Q_1 et Q_2 sous forme paramétrique :

$$a(t)\,s^2 + b(t)\,s + c(t) = 0 \qquad \text{(Eq. 2.11)}$$

Pour générer la courbe il suffit de balayer l'intervalle de définition de t, de résoudre l'équation (Eq. 2.11) en s, et de réinjecter dans (Eq. 2.10) pour trouver le point dans le repère de la surface réglée. Il ne reste qu'à transformer ce point pour le ramener dans l'espace de la scène.

Sarraga [Sarraga83] a développé et adapté les travaux de Levin dans le cadre du modeleur CSG de General Motors GMSolid [Boyse82]. Il a notamment étudié des techniques pour détecter les intersections planes. Par ailleurs il ne travaille que sur la sphère, le cône et le cylindre mais son travail peut parfaitement être étendu aux autres quadriques.

Méthode de Miller

Miller [Miller87] a le même objectif que Sarraga mais il choisit une approche géométrique. Comme il ne travaille que sur le cône, la sphère et le cylindre il n'a besoin que de deux types de surface de paramétrisation (ie la quadrique réglée) : le cône et le cylindre (l'intersection sphère-sphère, si elle existe, est un point ou un cercle, elle peut donc être traitée à part). Ensuite il résout chaque cas d'intersection en fonction du type des quadriques, de leurs positions relatives et de leurs tailles. Il détermine ainsi la topologie de la courbe d'intersection et ses points particuliers.

Cette méthode est moins sensible numériquement et un peu plus efficace car le code est spécifique pour chaque intersection (et donc plus long en tout). Toutefois comme le reconnaît Miller il ne peut pas être étendu à toutes les quadriques.

2.3.2 Le rendu CSG direct

Le rendu CSG direct consiste à rendre directement l'arbre CSG sans repasser par une représentation B-Rep. En général les méthodes développées l'ont été sur des primitives planes. Récemment une étude a été réalisée sur les quadriques. Avant de présenter cette étude et afin de bien la comprendre nous présentons les techniques classiques de rendu CSG.

Techniques classiques

En rendu CSG on peut distinguer trois grandes étapes, dont l'ordre peut varier mais qui est généralement :

1. Détermination des primitives présentes au pixel considéré : c'est la phase de conversion objet-pixel.
2. Détermination des primitives valides : c'est la phase de *classification* qui consiste à savoir si les primitive présentes au pixel considéré appartiennent ou non à la B-Rep de l'objet CSG.

3. Détermination de la primitive visible : c'est la phase d'élimination des parties cachées.

Les deux méthodes de rendu sont le lancer de rayon et le rendu projectif. Par ailleurs il existe deux types de classifications [Tilove80], la classification rayon-objet et point-objet.

De nombreux travaux ont cherché à réduire le coût élevé de la phase de classification, suivant deux grandes stratégies (pas forcément incompatibles). On peut réduire le nombre d'évaluations à réaliser (une évaluation consiste à déterminer si un point donné appartient à la B-rep de l'objet CSG ou non. Si l'on évalue tous les points d'intersections, on a réalisé la classification). On peut réduire le coût d'une évaluation en "simplifiant" l'arbre décrivant l'objet CSG. Nous présentons les grandes techniques pour l'une et l'autre des stratégies.

L'élimination des parties cachées, pour le lancer de rayon, consiste simplement à sélectionner le point valide le plus proche de l'observateur. Pour le rendu projectif, différentes méthodes d'élimination des parties cachées ont été adaptées. Nous présentons ici les principes des méthodes à balayage de ligne[1] [Akeley88] et celles basées sur l'algorithme du tampon en profondeur[2] [Rossignac86].

Les deux méthodes de classification

Roth [Roth82], un des pionniers du rendu CSG, a utilisé une classification rayon-objet. Pour cela il combine la classification de chaque primitive en fonction de l'arbre CSG. Le rayon est divisé en plusieurs intervalles par les points d'intersections avec la primitive. Chaque intervalle est dit intérieur si il est à l'intérieur de la primitive et extérieur dans le cas contraire. A chaque noeud les intervalles des sous-arbres gauche et droit sont composés en fonction de l'opérateur booléen du noeud (voir [Roth82] p.122 pour la table de composition complète). Lorsque l'on arrive à la racine la classification rayon-objet est terminée. La classification rayon-objet n'est valide que dans le cadre du lancer de rayon. En effet il est nécessaire d'avoir un rayon pour cette classification ce qui n'est pas le cas en rendu projectif.

L'autre méthode est la classification point-objet. Pour cela il faut définir la notion de *face avant* et *face arrière*. Une face est dite avant si sa normale est dirigée vers l'observateur. Elle est dite arrière dans le cas contraire. Un point est classé intérieur par rapport à une primitive s'il est derrière la face avant de la primitive et devant la face arrière (ce critère n'est valable que pour les objets convexes mais peut être étendu aux objets concaves). Ensuite on combine la classification par rapport à chaque primitive selon l'opérateur CSG à chaque noeud. Lorsque l'on arrive à la racine on obtient la classification du point par rapport à l'objet.

Réduire le coût d'une évaluation

L'idée générale est que l'évaluation d'un point ne dépend pas de l'ensemble de l'arbre décrivant l'objet CSG mais seulement d'un sous-ensemble. Si l'on peut trouver ce sous-ensemble on peut effectuer l'évaluation dessus, ce qui est moins coûteux que sur l'arbre complet. Il est à noter que ce sous-ensemble peut éventuellement dépendre du pixel considéré ou de la primitive à laquelle appartient le point à évaluer. Les trois grandes méthodes sont la façon de parcourir l'arbre, l'*élagage*[3] de l'arbre et la restructuration de l'arbre.

Parcours de l'arbre CSG

Il existe deux façons de parcourir l'arbre. La méthode standard consiste à parcourir l'arbre de la racine et à descendre (récursivement) jusqu'aux feuilles. Cette méthode simple peut être améliorée [Tilove80] [Roth82] en constatant que selon l'opérateur CSG au noeud considéré et le résultat de l'évaluation du sous arbre gauche, il n'est pas forcément nécessaire d'évaluer le sous-arbre droit. Par exemple pour un opérateur d'union si le point est classé intérieur pour le sous-arbre gauche alors la combinaison du sous-arbre droit et du sous-arbre gauche sera

1. scanline
2. Z-buffer
3. Tree pruning

forcément intérieur quel que soit le résultat du sous-arbre gauche. Il n'est donc pas nécessaire de l'évaluer.

L'autre méthode encore plus efficace est appelée *status tree technique* [Sato85] ou *bottom-up classification* [Bronsvoort86]. A chaque primitive et chaque noeud de l'arbre on associe un bit de statut initialisé à extérieur. L'idée de base est que pour un point à évaluer le statut d'une seule primitive change. On commence donc par la primitive à laquelle le point à évaluer appartient. Ensuite on remonte dans l'arbre jusqu'à ce que le bit de statut ne change plus ou que l'on arrive à la racine. Dans le premier cas, l'évaluation du point est la même que celle du point précédent. (si à un noeud donné le statut ne change pas il ne changera pas a fortiori dans les noeuds parents). Dans le second cas, si le statut de la racine est évalué intérieur le point classé est valide.

Elagage de l'arbre CSG

L'*élagage* d'un arbre peut se faire avant la phase de classification si par une méthode ou par une autre on peut connaître a priori le résultat de l'évaluation d'un sous-arbre. Cette connaissance a priori est possible par exemple en fonction de critères spatiaux. Si l'espace 3D est partitionné, on peut pour une partition donnée classer extérieures toutes les primitives qui n'ont pas d'intersection avec. On peut également pour une primitive donnée déterminer un sous arbre ne contenant que les primitives avec lesquelles elle a une intersection. La méthode des *zones actives*, proposée dans [Rossignac89] consiste à réécrire pour chaque primitive A_i l'arbre sous la forme :

$$A_i \cap I_{zone} \cup U_{zone} \qquad \textbf{(Eq. 2.12)}$$

Pour classer un point de la primitive A_i il suffit de considérer le sous arbre I_{zone} qui de plus peut être élagué en fonction de considérations spatiales.

Restructuration de l'arbre CSG

La *restructuration* de l'arbre est intéressante dans le cadre de classification à z-buffer. Dans ce type de système une union peut être gérée simplement par le z-buffer. Il est donc intéressant de réorganiser l'arbre sous la forme :

$$\bigcup_i \left(\bigcap_j P_{ij} \right) \qquad \textbf{(Eq. 2.13)}$$

L'affichage d'un objet, noté D(objet) se fait par l'affichage de chaque intersection de l'expression ci-dessus que l'on envoie dans le z-buffer. L'affichage d'une intersection se fait comme suit :

$$D\left(\bigcap_j P_{ij} \right) = \bigcup_k \left(D(P_{ik}) \bigcap_{j \neq k} P_{ij} \right) \qquad \textbf{(Eq. 2.14)}$$

On a de nouveau une union qui peut être gérée grâce au z-buffer. Cette méthode permet de réduire le coût d'une évaluation au prix de plusieurs rendus pour chaque primitive. Elle n'est donc intéressante que dans des systèmes ou le rendu est peu coûteux, c'est à dire dans des machines comme Pixel Planes-5.

L'adaptation à des systèmes matériels a été proposé parallèlement par Goldfeather et al. [Goldfeather86a] [Goldfeather89] et Jansen [Jansen86a] [Jansen86b]. On effectue tout d'abord une restructuration de l'arbre et un élagage. Ensuite, afin de minimiser la quantité de mémoire nécessaire on effectue le rendu de chaque primitive à chaque fois que l'on doit classer un point par rapport à cette primitive. On paye donc l'absence de mémoire par une répétition des calculs.

Réduire le nombre d'évaluations

La première idée, assez triviale part du fait que l'on ne cherche pas à réaliser une classification complète mais que l'on veut déterminer le premier point valide et visible de l'objet. On dispose alors de deux méthodes. Si l'on dispose d'un système à z-buffer on effectue le test du z-buffer avant l'évaluation. Ainsi tout point derrière le point visible courant est éliminé par le test du z-

buffer et n'a pas besoin d'être classé. L'autre méthode consiste à déterminer tous les points d'intersection, de les trier en profondeur puis de les classer en partant du plus proche de l'observateur. Ainsi le premier point classé comme appartenant à l'objet CSG est le point visible. Tous ceux qui sont derrières n'ont pas à être classés.

On peut chercher à réduire encore le nombre de points à classer. Tout d'abord par élimination des faces arrières[1]. Cette méthode doit être adaptée au rendu CSG. Pour chaque primitive on parcourt l'arbre de la racine vers la primitive et l'on compte le nombre de fois où l'on passe par la droite sur l'opérateur de différence. Si l'on passe un nombre de fois pair la primitive est dite positive et l'on conserve les faces avant. Dans le cas contraire la primitive est dire négative et l'on conserve les faces arrières.

Dans le *Trickle Algorithm*, Epstein, Rossignac et Jansen [Epstein89] se basent sur l'algorithme proposé par Goldfeather et al. pour Pixel Planes-5 et l'améliorent. Pour cela ils utilisent trois tampons de profondeurs, le premier (Z1) stocke la profondeur de l'objet courant, le second (Z2) stocke la profondeur du produit et le troisième (Z3) est le z-buffer "normal". Toute la subtilité de l'algorithme réside dans la sélection des points-tentatives pour un produit donné, dont la profondeur est rangée dans Z2. Ce point est testé par rapport aux autres faces des primitives du produit (dont la profondeur est rangé temporairement dans Z1). Lorsque l'on a trouvé le premier point valide du produit on effectue le test du z-buffer et on passe au produit suivant.

Enfin on peut profiter de la *cohérence CSG*. L'idée est qu'en un pixel donné, la classification est probablement la même qu'au pixel précédent. Pour le détecter il faut avoir une liste des intersections triées en profondeur. En un pixel, ces points ont été classés jusqu'au premier valide (et donc visible). Au pixel suivant si l'ordre des points d'intersections jusqu'au premier valide n'a pas changé alors l'évaluation de ces points n'a pas non plus changée. On n'a donc pas besoin de réévaluer les points. Si l'ordre a changé alors il faut réévaluer mais seulement à partir du premier changement dans la liste ordonnée. On peut vouloir mémoriser seulement la dernière liste triée ou bien toutes les listes triées. Il faut dans ce cas là avoir une structure de donnée adéquate [Jansen91]. La cohérence CSG est beaucoup utilisée dans les méthodes de rendu à balayage de lignes et en lancer de rayon.

Adaptation à la quadrique

Le rendu CSG peut être adapté à la quadrique aussi bien en lancer de rayon qu'en rendu projectif. Pour ce qui est du lancer de rayon, il s'adapte très bien à toute primitive dans la mesure où l'on est capable de calculer la ou les intersection(s) de cette primitive avec un rayon, ce qui est le cas de la quadrique. Comme il s'adapte aussi très bien au CSG il n'y pas d'étude particulière à faire dans ce cadre.

Les méthodes de rendu projectif posent problème avec la quadrique car c'est une primitive qui peut être concave. Nous présentons maintenant les travaux de R. van Kleij qui les a étudiées de façon approfondie.

Travaux de van Kleij

L'objet des travaux de van Kleij est d'effectuer un rendu CSG direct à ligne de balayage en incorporant les primitives quadriques [Kleij93]. Pour cela il détermine tout d'abord ce qu'il appelle la *Active Edge List* ou *AEL* qui permet de savoir pour chaque ligne de balayage quelles sont les primitives qui la coupent. Pour obtenir cette information pour une quadrique il détermine les bords de sa silhouette.

Ensuite il travaille par ligne de balayage. Le plan perpendiculaire à l'écran et passant par la ligne de balayage considérée coupe les objets se trouvant dans l'AEL. Dans ce plan on a donc un ensemble de segments de coniques. La liste ordonnée en x des extrémités de tous les segments permettent de définir un ensemble d'intervalles appelés *spans*. Toutefois même si l'ordre des primitives aux deux extrémités d'un span est le même on peut avoir des intersections. Pour

1. back face culling

résoudre ce problème il divise l'intersection du plan et d'une quadrique en segments monotones (Figure 2.9).

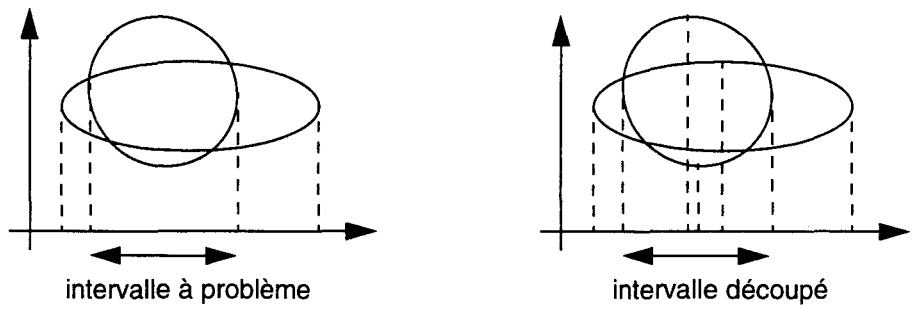


Figure 2.9 ***Span à problème (à gauche) et découpe en segments monotones (à droite).***

L'étape suivante consiste à déterminer s'il y a une intersection entre deux segments monotones. Un premier critère est le chevauchement des rectangles englobants des deux segments (cf. Figure 2.10).

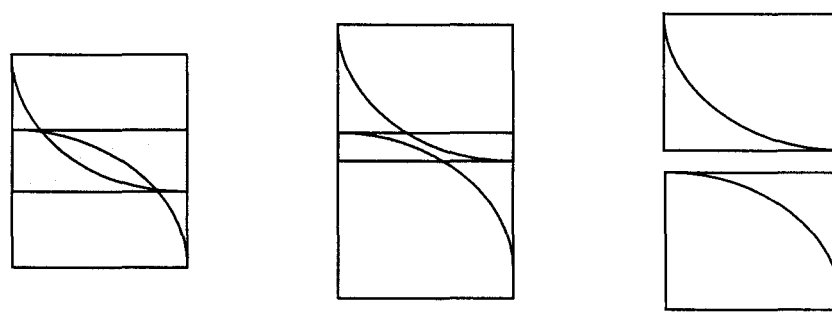


Figure 2.10 ***Un chevauchement en profondeur peut indiquer une intersection***

Dans le cas où il y a chevauchement il propose d'autres critères : le segment est-il concave ou convexe ? Est-il devant ou derrière l'autre segment ? De ces critères il déduit des configurations simples et d'autres plus compliquées. Dans ce dernier cas il subdivise récursivement le span jusqu'à avoir zéro ou une intersection.

Pour le calcul d'intersection il utilise la méthode de Newton-Raphson. Il choisit comme point de départ le centre de l'intersection des deux rectangles englobants des segments considérés. Pour les cas (rares) où la méthode diverge il détermine un autre point de départ par une méthode plus complexe. Le span est subdivisé en deux s'il y a un point d'intersection et en trois s'il y a deux points d'intersection.

2.3.3 Approches orientées matériel

Deux machines très célèbres autorisent le rendu quadrique, la Ray Casting Machine et la machine Pixel Planes-5. L'objectif recherché est d'afficher des arbres CSG par tracé de rayon (ie lancer de rayon pour lequel on ne lance que les rayons primaires) pour la première et par rendu projectif pour la seconde. Ces deux machines universitaires très sophistiquées contrastent

fortement avec une troisième approche, celle de la firme NVidia qui propose une carte accélératrice pour compatible PC.

La machine Pixel Planes-5

Ce système massivement parallèle est le successeur de Pixel planes-4 [Fuchs85] [Eyles88] également développé à Chapel Hill par l'équipe de Henry Fuchs. Une brève description de la machine permet de comprendre pourquoi l'on peut rendre des sphères et des cylindres moyennant une approximation sur le calcul de profondeur. Nous pouvons alors expliquer comment on effectue le rendu de ces deux objets.

La machine

L'unité de base de Pixel Planes-5 [Fuchs89], appelée Renderer, est un tableau de 128×128 processeurs pixel alimenté par un arbre d'additionneurs permettant d'évaluer des expressions du premier et second degré en x, y (x, y étant les coordonnées d'un pixel). Il est possible de mettre en parallèle plusieurs Renderer s'occupant chacun d'une zone de l'écran. Un anneau haut débit (8 voies de 80 Mo/s chacune) permet de connecter ces cartes, ainsi que les cartes de calculs géométriques, la mémoire de trame et l'ordinateur hôte.

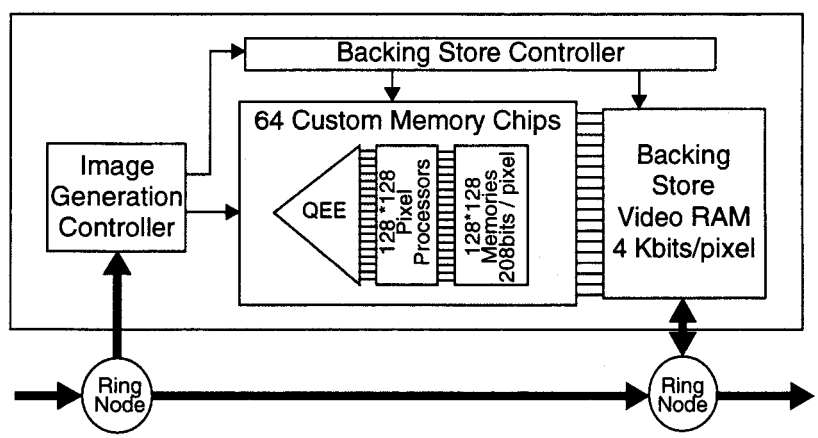


Figure 2.11 ***Architecture de Pixel Planes-5***

L'approximation de sphères et de cylindres

La sphère et le cylindre étant des primitives courantes, l'équipe de Chapel Hill a cherché à réaliser une machine qui puisse les rendre directement, en évitant la phase de facettisation [Goldfeather86a] [Eyles88]. Pour cela la capacité à évaluer des expressions du second degré en x, y se révèle indispensable.

La première difficulté consiste à déterminer les pixels à l'intérieur de la projection de la primitive. Les contours d'une sphère et d'un cylindre sont représentés Figure 2.12. On note que le contour d'une sphère est simplement un cercle et que le contour d'un cylindre peut être déterminé par une combinaison de cercles et de droites. Pour déterminer le contour on utilise une méthode par équations. Les droites et les cercles sont définis par des expressions linéaires et quadratiques. Un test d'inclusion d'un pixel par rapport à une droite (en fait le demi-plan associé) ou à un cercle se fait par la détermination du signe de l'expression linéaire ou quadratique correspondante. Le test d'inclusion pour le cylindre se fait par combinaison booléenne de ces signes.

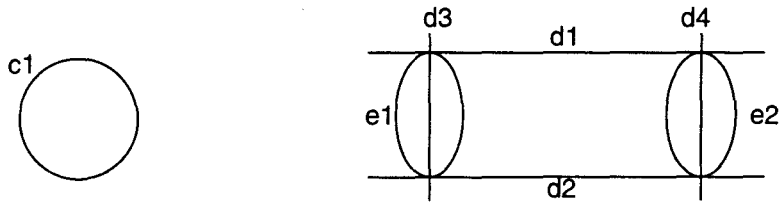


Figure 2.12 ***Contour d'une sphère (à gauche) et d'un cylindre (à droite)***

Ensuite il faut calculer la profondeur avant de la sphère ou du cylindre. Le calcul de la vraie profondeur nécessite une extraction de racine carrée, ce qui était trop cher à implanter dans la machine Pixel Planes-5. On calcule donc une profondeur approchée par une expression quadratique, ce qui revient à approcher la sphère par un paraboloïde et le cylindre par un cylindre parabolique (cf. Figure 2.13). Pour améliorer l'approximation on peut utiliser plusieurs cylindres paraboliques (de un à huit), c'est-à-dire plusieurs expressions quadratiques. Par ailleurs certaines applications comme le rendu CSG nécessitent de calculer la profondeur avant et arrière. On utilise donc d'autres cylindres paraboliques pour approcher la partie arrière de la sphère ou du cylindre.

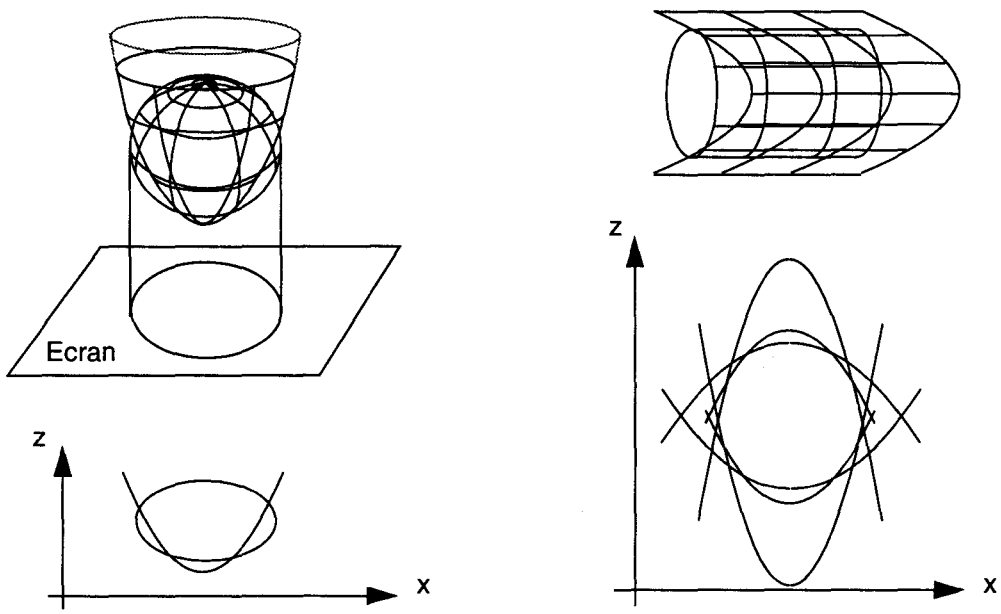


Figure 2.13 ***Approximation de la profondeur pour la sphère (à gauche) et le cylindre (à droite)***

Enfin pour l'éclairement il faut calculer la normale en chaque point. Pour la sphère, on utilise la profondeur approchée. Pour le cylindre, chacune des composantes qui normalement comprend une racine carrée est approchée par une expression quadratique par la même méthode utilisée pour la profondeur. Toutefois on n'utilise qu'une seule expression quadratique pour le cylindre tout entier, contrairement à la profondeur pour laquelle plusieurs expressions peuvent être utilisées.

L'absence d'extracteur de racine carré occasionne deux limitations. Tout d'abord la profondeur et la normale ne sont qu'approchées : pour la profondeur cela entraine des problèmes pour l'élimination des parties cachées qui est effectuée avec l'algorithme du tampon de profondeur ;

pour la normale cela entraine une erreur (peu importante toutefois) sur l'éclairement. De plus on ne peut pas manipuler toutes les quadriques.

La Ray casting machine

Cette machine [Kedem89] [Ellis91] est un système spécifiquement conçu pour afficher des images construites par combinaison CSG de primitives planes et quadriques. Le but de ce système est la production d'images relativement complexes dans un temps raisonnable (quelques dizaines de secondes).

La machine est organisée en un arbre dans lequel les noeuds sont des CC et les feuilles des PC. Un PC (Primitive Classifier) calcule la profondeur avant et arrière de la primitive quadrique qu'il a en charge (ou une profondeur s'il s'agit d'un plan) en chacun des pixels de l'écran. Un CC (Combine Processeur) réalise une opération CSG (union, intersection différence). Chaque CC reçoit de ses noeuds fils une liste de segments à partir de laquelle il en produit une nouvelle (en fonction de l'opérateur CSG à traiter) qu'il passe à son noeud père. L'éclairement est effectué par le hôte. L'élimination des parties cachées se fait en conservant le premier point du premier segment de la liste au niveau du noeud racine.

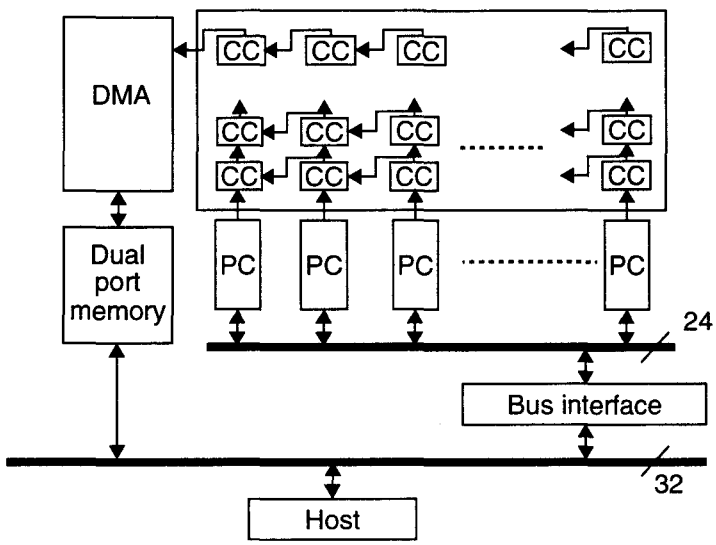


Figure 2.14 ***Architecture de la Ray Casting Machine***

Cette machine possède la remarquable particularité d'être une des rares machines universitaires à avoir été construite. Elle fonctionne en mode bit série. Le dernier prototype comprend 256 PC et 2048 CC. Les temps de calculs d'une image vont de quelques secondes à quelques minutes.

Carte NVidia

Cette société propose un processeur graphique pour compatibles PC offrant des possibilités d'accélération pour l'affichage de polygones 3D [NVidia95a]. Pour la texture, la carte ne réalise pas d'interpolation rationnelle linéaire à cause du coût élevé des diviseurs[1]. Toutefois une correction perspective est réalisée au moyen d'une interpolation quadratique (cette technique n'est pas exacte mais offre toutefois une correction perspective pour un coût faible).

1. Cette phrase, obscure pour le béotien, pourra être mieux comprise après la lecture du paragraphe 3.5.1

Ces interpolateurs quadratiques peuvent servir également pour rendre des surfaces courbes. De la même façon que l'on définit un triangle avec trois points, on peut ici définir un carreau de quadrique avec neuf points (Figure 2.15). On peut alors effectuer la conversion objet-pixel par suivi de contour des coniques qui le composent [NVidia95b]. On peut également interpoler les valeurs des paramètres aux "sommets" par une interpolation bi-quadratique (au lieu de bi-linéaire pour un polygone).

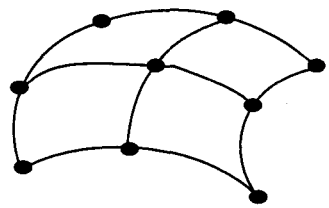


Figure 2.15 ***Un carreau de quadrique défini par neuf points***

Il est clair que la profondeur calculée n'est qu'une valeur approchée par une expression du second degré puisqu'il n'y a pas d'extracteur de racine carrée. On peut donc s'attendre à avoir les mêmes problèmes que sur Pixel Planes-5. D'autre part NVidia ne semble pas s'être posé la question de savoir comment utiliser cette capacité à part peut-être pour des cas particuliers (une sphère par exemple). On remarque néanmoins qu'avec cette carte on voit arriver des algorithmes de rendu accéléré de quadrique sur le marché grand public.

2.4 La quadrique au LIFL

Nous retraçons ici l'historique des études qui ont été faites sur la quadrique au LIFL. Les premiers travaux, conduits par Eric Nyiri, sont une tentative pour définir un système de synthèse d'images basé sur la quadrique. Les différentes propositions faites apportent une solution partielle au problème. En particulier, ses travaux permettent de mettre en évidence les problèmes liés à l'emploi d'une nouvelle primitive d'affichage.

Suite à ces travaux l'étude s'est scindée en deux axes, la modélisation et la visualisation [Laporte95a], qui font actuellement l'objet de travaux de thèses. L'axe visualisation fait l'objet de ce mémoire. L'axe modélisation incombe à Maxime Froumentin. Dans le contexte présenté brièvement au paragraphe 2.2, il propose un certain nombre d'approches originales pour utiliser la quadrique comme primitive de modélisation [Froumentin96].

2.4.1 Premières propositions

Eric Nyiri [Nyiri94] a étudié la quadrique comme primitive d'affichage en synthèse d'images dans le cadre le la machine IMOGENE [Chaillou91] [Karpf92]. Une première approche inspirée des travaux sur Pixel Planes-5 s'est vite révélée trop limitée. Il a donc proposé une méthode originale inspirée du rendu direct CSG, beaucoup plus intéressante. Toutefois ses tentatives n'ont pas abouti à la conception d'un système répondant à toutes les contraintes. Nous présentons donc les limitations de ses propositions.

Première approche

Les premiers travaux d'Eric Nyiri, basés sur ceux de Chapel Hill, l'on conduit à définir une première génération de processeur quadrique. Un objet quadrique est composé d'une quadrique et de *plans de coupe*.

Pour le rendu une méthode par contour est utilisée. La silhouette de la quadrique qui est une conique constitue le premier contour. L'intersection de la quadrique et d'un plan de coupe est aussi une conique. Sa projection (qui reste une conique) fournit le deuxième contour. Le troisième contour est la projection de l'intersection du plan de coupe et du plan polaire de la quadrique (Figure 2.16). Avec ces informations on peut réaliser le test d'inclusion.

Cette méthode ne fonctionne que pour un seul plan qui coupe la quadrique. Or plusieurs plans peuvent être utilisés et générer ainsi des contours plus complexes (Figure 2.16). Eric Nyiri propose un algorithme permettant de gérer un nombre quelconque de plans mais reconnaît qu'il est trop difficile à implémenter matériellement. Il limite donc la définition de l'objet de base à une quadrique et éventuellement un ou deux plans parallèles.

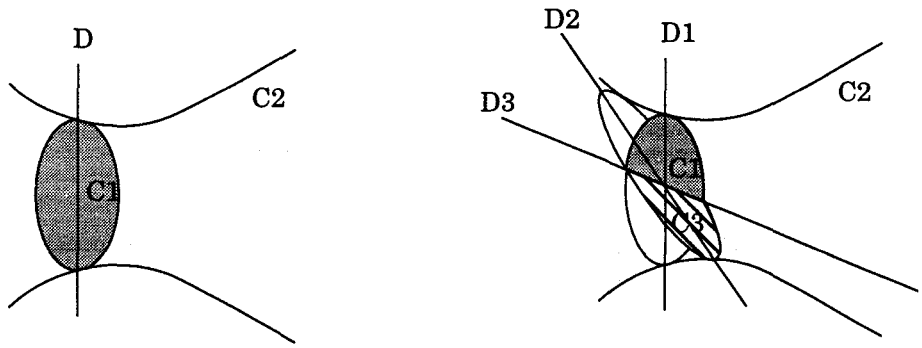


Figure 2.16 ***Rendu par méthode des contours***

Une nouveauté de cette première approche est l'emploi d'un extracteur de racine carrée ce qui permet de manipuler toutes les quadriques. De plus les profondeurs calculées sont exactes (aux erreurs d'arrondis près) ce qui évite certains défauts visuels qui apparaissent lorsque l'on calcule une profondeur approchée.

Deuxième approche

La première génération de processeur souffre du nombre réduit de plans qui doivent de plus être parallèles. Par ailleurs, la méthode par contour n'est pas très modulable et s'avère coûteuse si l'on veut augmenter le nombre de plans de coupe. Pour éviter ces problèmes, la seconde génération utilise de nouveaux algorithmes qui s'inspirent des méthodes de résolution d'arbre CSG au niveau pixel. E. Nyri définit ainsi le processeur quadrique volumique et le processeur polyèdre. Il propose également une méthode pour calculer les ombres portées. Enfin il définit avec l'auteur le processeur patch quadrique.

Processeur quadrique volumique

Les plans et la quadrique sont considérés comme définissant des demi-espaces. L'objet quadrique est le volume défini par l'intersection de tous ces demi-espaces. On a en fait simplement étendu la notion de quadrique naturelle à tous les types de quadriques. Pour le rendu on ne génère plus les contours. On construit l'objet à partir de la quadrique et en déterminant son intersection successivement avec tous les plans[1] selon l'algorithme de la Figure 2.17. Notons que zp est la profondeur du plan au pixel considéré, zf et zb les profondeurs avant et arrière de la quadrique et zmax la profondeur maximale de la scène (qui sert à initialiser le z-buffer). Au début de l'algorithme on suppose que les profondeurs de l'objet sont les profondeurs de la quadrique. Un plan de coupe est soit perpendiculaire à l'écran, soit avant

1. Ou plus exactement le demi-espace défini par le plan. Souvent on parle simplement des plans par abus de langage

(on garde ce qui se trouve derrière le plan pour l'observateur) soit arrière (on garde ce qui se trouve entre le plan et l'écran).

```
SI le plan frontière du demi-espace est perpendiculaire à l'écran
ALORS SI    on est du mauvais coté du plan
          ALORS     zf=zb=zmax
SI le plan frontière est avant
ALORS SI    (zp>zb)
          ALORS     zf=zb=zmax
          SINON     SI    (zp>zf)
                    ALORS     zf=zp
SI le plan frontière est arrière
ALORS SI    (zp<zf)
          ALORS     zf=zb=zmax
          SINON     SI    (zp<zb)
                    ALORS     zb=zp
```

Figure 2.17 ***Algorithme d'intersection avec un demi-espace***

Processeur polyèdre

A partir de la définition du processeur quadrique volumique, E. Nyiri définit le processeur polyèdre. Celui-ci fonctionne exactement de la même manière sauf que la quadrique disparaît. Au départ l'objet est en fait tout l'espace qui est successivement réduit par les plans de coupes jusqu'à obtenir un polyèdre convexe.

Ombre portée

Puisqu'il manipule facilement les volumes, E. Nyiri s'est intéressé aux volumes d'ombre. Il a adapté la méthode de Crow pour les quadriques. La méthode sera décrite au paragraphe 3.4.3.

Processeur patch quadrique

Enfin une étude menée conjointement avec l'auteur [Laporte93] a donné lieu à la définition d'un quatrième type de processeur gérant des surfaces, le processeur patch quadrique. L'objet quadrique est considéré comme la partie de la surface quadrique comprise dans le tétraèdre formé par l'intersection de quatre plans. Une adaptation de l'algorithme de la Figure 2.17 permet de "rendre invisibles" les plans de coupes. Cette étude a été réalisé afin de pouvoir afficher le patch quadrique qui est apparu avec les premiers travaux de Maxime Froumentin en modélisation. On étendait ainsi les possibilités d'affichage sans pour autant avoir de solution globale et générique.

Limitations

Il est difficile de concevoir un système de visualisation quel qu'il soit lorsque l'on ne sait pas exactement ce que l'on veut afficher. E Nyiri s'est heurté au problème de la primitive de visualisation. Il a fait trois propositions, la quadrique volumique, le patch quadrique et le polyèdre. Elles sont intéressantes mais n'offrent pas de solution générale. A partir de là d'autres problèmes persistent.

Premièrement il n'est pas possible de concevoir des algorithmes généraux. On a un type d'algorithme par type d'objet quadrique. Ces algorithmes marchent bien mais seulement dans des cas limités. Or il n'est pas envisageable d'implémenter matériellement chacun des algorithmes. Une uniformisation est donc indispensable.

L'absence d'uniformisation se fait également ressentir pour les algorithmes d'amélioration de la qualité, qui sont pourtant indispensables aujourd'hui. Les ombres portées sont un élément de réponse mais il faut être en mesure d'apporter des solutions pratiques notamment aux problèmes d'aliassage et de placage de texture.

2.4.2 Aspect modélisation

Le travail de Maxime Froumentin s'inscrit dans la continuité des travaux présentés au paragraphe 2.2. L'objectif est de pouvoir fournir des outils puissants pour modéliser des scènes en quadriques afin de nourrir l'accélérateur décrit dans ce mémoire. Pour cela les deux voies proposées, la modélisation directe et la conversion de surfaces complexes en ensembles de patchs quadriques, sont explorées. Tous ces travaux sont décrits en détails dans [Froumentin96].

Modélisation directe

Dans un premier temps Froumentin s'inspire des techniques à macro-patchs pour proposer une méthode plus souple et plus interactive de modélisation avec les quadriques [Froumentin94]. Pour cela il propose une méthode à polygone de contrôle, couramment utilisée en modélisation. Les surfaces les plus simples qu'il est possible d'imiter ainsi sont les triangles de Bézier quadratiques.

Il propose tout d'abord un macro-patch à quatre quadriques. Ensuite il propose des macro-patchs ayant de quatre à dix-huit patchs selon les exigences sur la continuité (continuité G^0 ou G^1 sur un macro-patch, raccordement G^0 ou G^1 entre les macro-patchs).

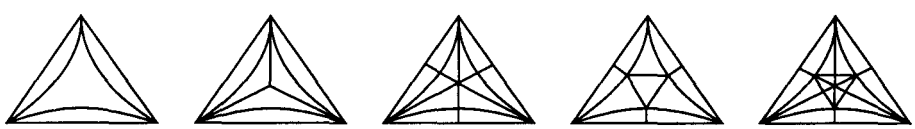


Figure 2.18 ***Différents macro-patchs. Plus le nombre de patchs (indiqué sous le dessin) est élevé plus les propriétés de continuité sont bonnes.***

Avec cette approche on a une vraie primitive de modélisation quadrique. Toutefois elle s'avère finalement trop contraignante, la souplesse recherchée reste faible, notamment pour le raccordement entre macro-patchs avec une continuité G^1. Afin d'avoir plus de souplesse, la méthode a été étendue au B-splines.

L'idée est de remplacer, dans la représentation de Bernstein-Bézier, la surface de Bézier triangulaire fonctionnelle de dimension trois par une surface B-spline. Pour cela on utilise la généralisation des courbes B-splines aux B-splines de dimension trois. Comme pour les surfaces de Bézier on utilise des surfaces B-splines triangulaires.

L'avantage majeur des surfaces B-splines est que le degré de continuité ne dépend plus des points de contrôle mais de la position des noeuds. On va donc pouvoir enfin avoir toute la souplesse recherchée. D'autre part on assure une compatibilité ascendante avec les macro-patchs selon la représentation de Bernstein-Bézier.

Avec cette méthode on a une surface quadrique par morceaux. Pour les besoins du rendu il faut connaître l'équation algébrique de chaque morceau ainsi que le tétraèdre qui lui est associé. Maxime Froumentin a développé un algorithme original à cette fin.

Conversion de surfaces complexes en ensembles de quadriques

Maxime Froumentin a développé un algorithme [Froumentin95] qui convertit les surfaces algébriques exprimées sous la forme de Bernstein-Bézier. On peut alors appliquer certaines propriétés bien connues des carreaux de Bézier aux surfaces algébriques, en particulier les algorithmes de réduction de degré.

L'algorithme de Froumentin nécessite en entrée une surface algébrique et une boîte englobante. Il utilise le partitionnement spatial comme le font les algorithmes de facettisation : la boîte englobante est subdivisée en tétraèdres. Chaque carreau de surface est alors exprimé selon la représentation de Bernstein-Bézier. On peut alors réaliser une réduction de degré. On obtient ainsi une surface composée d'un ensemble de quadriques qui forment une bonne approximation de la surface originale. Toutefois la surface n'est que G^0.

Cette méthode peut également être employée pour les surfaces paramétriques. Maxime Froumentin a écrit un algorithme qui convertit les surfaces de Steiner qui sont des surfaces de Bézier triangulaires quadratiques rationnelles [Sederberg85]. Pour cela on calcule l'équation implicite (c'est à dire on passe de la forme paramétrique à la forme implicite) qui est de degré quatre à partir de l'expression paramétrique. Ensuite on partitionne l'enveloppe convexe de la surface, un octaèdre, en quatre tétraèdres. Enfin on applique l'algorithme de réduction de degré.

2.5 Bilan

Cet état de l'art nous a permis d'aborder la quadrique sous différents aspects dans différents domaines et pas seulement en visualisation. Nous avons commencé notre état de l'art sur la quadrique par un rapide exposé mathématique. Ensuite nous avons exhibé toutes les implications au niveau de la modélisation dues au changement de la primitive de visualisation. Nous avons montré que les deux façons de procéder étaient soit de modéliser directement avec des quadriques, soit de modéliser avec des surfaces complexes et de les convertir en un ensemble de quadriques. Les études menées au LIFL sur ces problèmes, rapidement décrites dans ce mémoire sont abordées en détail dans [Froumentin96].

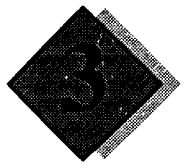
Néanmoins la visualisation reste notre première préoccupation. Nous avons recensé les travaux existant sur le rendu de la quadrique. Ces travaux concernent principalement le rendu d'arbre CSG, ou le rendu par lancer de rayon. Quelques travaux se sont intéressés à la quadrique en rendu projectif mais soit ils ne traitent pas toutes les quadriques, soit ils ne répondent pas à nos contraintes de performance temps réel.

Les questions restées en suspend dans les travaux de E. Nyiri sont la motivation première de notre étude. Elles forment un premier cahier des charges. Les principaux problèmes sont les suivants :

- Définition d'une primitive de visualisation unique et générique
- Etude des algorithmes pour le rendu de la primitive de visualisation
- Amélioration de la qualité visuelle des images produites
- Etude d'un processeur de rendu qui manipule la primitive de visualisation au moyen des algorithmes proposés
- Etude d'un système complet dans lequel s'insère le processeur de rendu

Les solutions que nous proposons à ces problèmes font l'objet des chapitres trois et quatre. Dans le chapitre trois nous abordons le problème de la primitive de visualisation et les problèmes

algorithmiques. Dans le chapitre quatre nous présentons les développements matériels et logiciels.



Notre proposition

Nous allons maintenant aborder les problèmes liés à la conception de notre système. Tous les problème qui apparaissent sont dûs au changement de primitive de visualisation. Tout d'abord nous reconsidérons l'organisation générale du système de visualisation. Ensuite nous définissons la primitive de visualisation. Enfin nous mettons en évidence les problèmes algorithmiques liés au changement de la primitive de visualisation et nous présentons les solutions que nous y apportons.

Pour ce qui est de l'organisation générale de notre système, nous montrons que le pipeline de rendu habituellement utilisé (ie le pipeline de Gouraud, cf. paragraphe 1.2.4) ne convient pas à notre primitive de visualisation. Nous choisissons de le remplacer par le pipeline de Phong que nous présentons.

Comme nous l'avons remarqué au paragraphe 2.4.1, la définition de la primitive de visualisation est un préambule indispensable à toute étude algorithmique. En effet pour savoir *comment* afficher il faut savoir *quoi* afficher. Plusieurs contraintes, qui de plus sont de nature variée, doivent être prises en compte. Nous donnons une définition de l'objet de base et présentons ses caractéristiques, notamment sa grande souplesse.

Nous réalisons les études algorithmiques spécifiques à notre système graphique. Nous présentons en détail les algorithmes de rendu. Ensuite nous proposons des adaptations pour la quadrique des algorithmes classiques d'amélioration de la qualité. Le placage de texture est une technique particulièrement efficace qui prend de plus en plus d'importance dans les algorithmes de rendu. Nous étudions donc celle-ci de façon plus approfondie.

3.1 Organisation générale du système

L'organisation générale des accélérateurs graphiques habituels repose sur la méthode d'ombrage, en l'occurrence celle de Gouraud. Le pipeline de rendu utilisé est donc celui de Gouraud (cf. paragraphe 1.2.4). Or l'interpolation de Gouraud a été développée spécifiquement pour les facettes. L'usage de la quadrique remet donc en cause le choix du pipeline de Gouraud.

Une première possibilité est l'adaptation de l'ombrage de Gouraud à la quadrique. On peut envisager de faire une interpolation biquadratique de la couleur sur un morceau de quadrique ayant les caractéristiques de la Figure 2.15 (ie un carreau carré défini avec neuf points). On profite ainsi de la simplicité de l'interpolation de Gouraud. Malheureusement on hérite également de ses inconvénients. En fait, utiliser une telle méthode revient à travailler avec des facettes pour ce qui est de la qualité de l'éclairement. De plus on a une contrainte forte sur la forme de la primitive de visualisation. En particulier les quadriques naturelles ne sont pas définies comme des carreaux de surfaces et donc ne conviennent pas à l'interpolation.

Une seconde possibilité est le calcul direct de la couleur en chaque pixel en fonction de la formule d'éclairement de Lambert (une composante ambiante et une composante diffuse, voir annexe 6.1). Celle-ci nécessite la détermination de la normale au point considéré. Or, comme nous l'avons vu au paragraphe 2.1.1, il est relativement aisé de calculer en tout point la

normale à une quadrique. Toutefois, avec cette méthode, on ne peut plus avoir d'effet spéculaire. De plus la source lumineuse doit se trouver à l'infini.

La qualité d'éclairement offerte par les deux méthodes précédentes n'est pas satisfaisante. Nous nous sommes donc tournés vers la méthode d'éclairement de Phong. L'architecture d'un système basé sur cette technique est le pipeline de Phong. Celui-ci est basé sur l'interpolation des normales. Dans le cas de la quadrique nous pouvons garder l'architecture du pipeline de rendu de Phong en sachant que la normale n'est pas interpolée mais en fait calculée exactement.

L'utilisation du pipeline de Phong à la place de celui de Gouraud est à la fois un inconvénient et un avantage. En effet le pipeline de Gouraud est plus simple et plus efficace pour le rendu temps réel. L'emploi du pipeline de Phong induit un coup de calcul de l'éclairement très important. Néanmoins il offre deux avantages majeurs : tout d'abord il s'adapte à toute primitive dont on peut déterminer la normale en chaque point ; par ailleurs la qualité de l'éclairement et donc la qualité visuelle est bien supérieure.

3.1.1 Le pipeline de Phong

Nous présentons le pipeline de Phong (Figure 3.1) en donnant les différences par rapport à celui de Gouraud (cf. paragraphe 1.2.4). On y retrouve les deux grandes parties évoquées dans celui de Gouraud, la préparation et le rendu. De plus nous voyons apparaître une troisième partie, le post-éclairement.

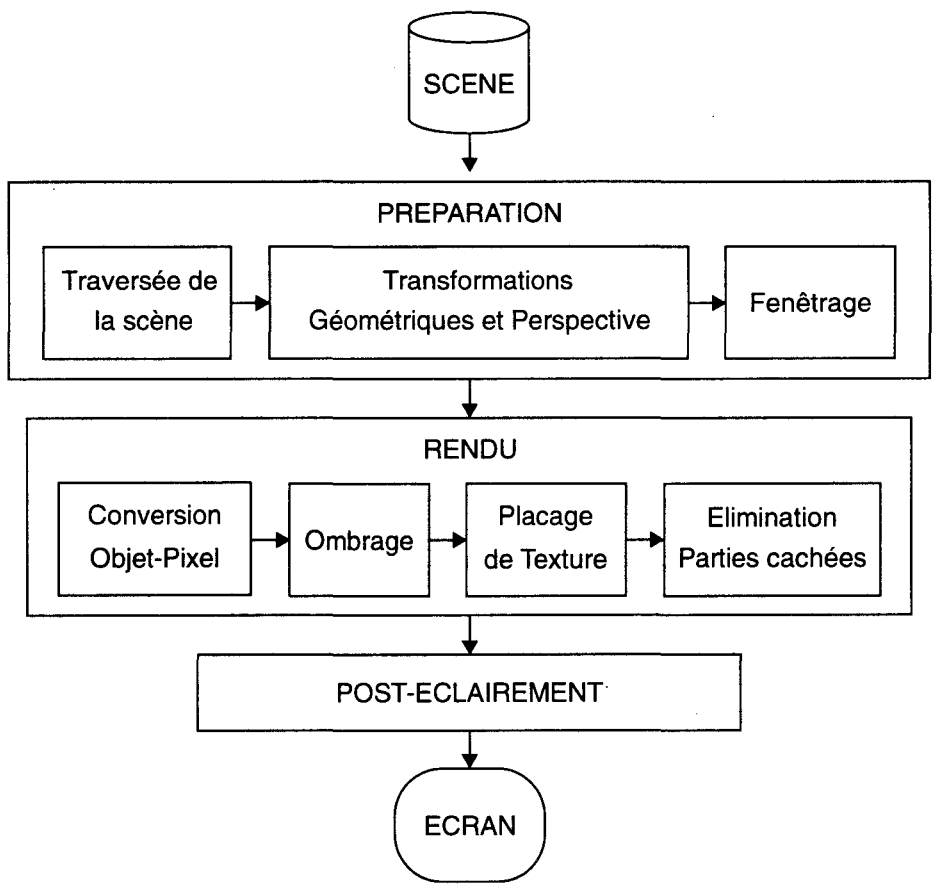


Figure 3.1 ***Le pipeline de rendu de Phong***

L'éclairement n'est plus réalisé lors de la préparation puisqu'il constitue la troisième partie du pipeline de rendu. Les phases de transformations géométriques et de transformation perspective sont donc regroupées. Globalement la préparation est donc moins coûteuse que dans le pipeline de Gouraud.

Au niveau du rendu la modification principale consiste, lors de la phase d'ombrage, à interpoler les normales au lieu d'interpoler les couleurs. Toutefois pour la quadrique la normale n'est pas interpolée mais calculé exactement. Notons que ce calcul est plus coûteux qu'une simple interpolation linéaire. De plus il faut calculer les coordonnées du point considéré dans le repère monde car l'éclairement doit se calculer dans ce repère.

Lors du post-éclairement, l'éclairement est calculé en chaque pixel en utilisant par exemple la formule proposée par Phong [Phong75]. Il est à noter qu'il n'intervient qu'après la phase d'élimination des parties cachées, ce qui limite le nombre de calculs. L'implémentation d'un post-éclairement de Phong est aujourd'hui encore un problème compliqué à cause de la charge de calcul. Celle-ci, ainsi que les différentes propositions d'implémentations existantes, sont brièvement présentées en annexe 6.2.

3.2 L'objet de base

L'une des plus grosses difficultés rencontrées dans les travaux précédents sur la quadrique au LIFL est la définition exacte d'une primitive de visualisation. Celle-ci, que nous appelons *objet de base*, conditionne l'ensemble de la conception des algorithmes de visualisation de notre système graphique.

Dans un premier temps nous montrons que la définition de l'objet de base n'est pas facile. En effet de nombreuses contraintes doivent être prises en compte. Elles sont dues à la nature même de la quadrique, à la modélisation, à la visualisation et plus particulièrement à la visualisation temps réel. Notre premier travail est donc d'identifier ces contraintes. A partir de là, nous pouvons donner une définition intuitive des caractéristiques de l'objet de base.

Nous formalisons alors les idées présentées précédemment puis nous illustrons les possibilités offertes par notre objet de base [Laporte95b]. Nous mettons ainsi en évidence sa souplesse, qui permet de gérer de la même façon les quadriques naturelles, les patchs quadriques, les polyèdres et les facettes, offrant par cette dernière possibilité une compatibilité ascendante avec les systèmes classiques. Enfin nous revenons sur la dualité surface/volume et nous choisissons le nombre de plans de la primitive de visualisation.

3.2.1 Position du problème

A partir de la liste exhaustive des contraintes imposées sur l'objet de base nous allons tenter d'en donner une définition intuitive.

Les contraintes à respecter sont les suivantes :

- L'utilisation de la quadrique seule comme primitive de visualisation n'est pas réalisable. En effet la plupart du temps les quadriques sont infinies dans au moins une direction. Il faut donc pouvoir les limiter. De plus les modèles n'utilisent que rarement les quadriques "entières", mais plutôt des morceaux de quadriques. Il faut donc être en mesure de "découper" un morceau de quadrique.
- Un des points les plus épineux est la "dualité" surface/volume de la quadrique. En effet nous avons vu que la quadrique peut être considérée soit comme surface, soit comme volume. Mathématiquement c'est une surface. De plus le patch quadrique est une primitive de modélisation surfacique. Néanmoins une quadrique divise l'espace en deux demi-espaces. Intuitivement on comprend que l'on puisse considérer l'un de ces demi-espaces comme

l'intérieur. C'est le cas des quadriques naturelles. Notre définition devra donc unifier ces deux approches.

- La définition de l'objet de base se doit d'être simple et précise afin qu'il puisse être manipulé facilement et efficacement par les algorithmes de visualisation.
- Les contraintes de temps-réel ont des conséquences importantes sur la complexité de l'objet de base. Même si algorithmiquement on peut manipuler un objet compliqué il faut de plus que l'implémentation matérielle soit facile. D'une part les calculs ne doivent pas être trop complexes. D'autre part ils ne doivent pas être trop nombreux.

La première idée est d'utiliser plusieurs quadriques pour définir l'objet de base, c'est-à-dire d'avoir une quadrique qui soit découpée par une ou plusieurs autres quadriques. Cette approche a été proposée par Weiss (voir 2.3.1). Bien que cette solution soit attrayante, elle ne peut pas être retenue pour des raisons de complexité. Comme nous le verrons dans le prochain chapitre le coût matériel pour effectuer le rendu d'une quadrique est élevé. Il devient donc prohibitif si l'on veut pouvoir gérer plusieurs quadriques.

L'idée que nous avons retenue est de découper la quadrique avec des plans. Les plans sont des primitives simples et faciles à manipuler. Ainsi nous obtenons la souplesse nécessaire pour notre objet de base tout en limitant la complexité du processeur.

3.2.2 Définition formelle

Rappelons qu'une quadrique est définie par une équation implicite du second degré en *(x, y, z)*. Un plan est défini par une équation implicite du premier degré en *(x, y, z)*. Notre objet de base est défini à partir d'un ensemble de primitives comprenant une quadrique (éventuellement zéro) et n plans.

On note la quadrique $R_0(x, y, z) = 0$ et les plans $R_1(x, y, z) = 0 \ldots R_n(x, y, z) = 0$. Une primitive, que ce soit la quadrique ou un plan définit deux demi-espaces : $R_i(x, y, z) \geq 0$ et $R_i(x, y, z) \leq 0$.

L'objet de base est une union de surfaces limitée par un volume "englobant". Ce dernier est défini par l'intersection de demi-espaces. Parmi les $n+1$ primitives servant à la définition de notre objet de base, supposons que p primitives contribuent à l'union de surfaces. Notre objet de base est mathématiquement défini ainsi :

$$\bigcup \begin{cases} R_{i_1}(x, y, z) = 0 \\ R_{i_2}(x, y, z) = 0 \\ \vdots \quad \vdots \\ \vdots \quad \vdots \\ R_{i_p}(x, y, z) = 0 \end{cases} \bigcap \begin{cases} R_0(x, y, z) \geq 0 \\ R_1(x, y, z) \geq 0 \\ \vdots \quad \vdots \\ \vdots \quad \vdots \\ R_n(x, y, z) \geq 0 \end{cases} \qquad \textbf{(Eq. 3.1)}$$

ou, de façon plus concise :

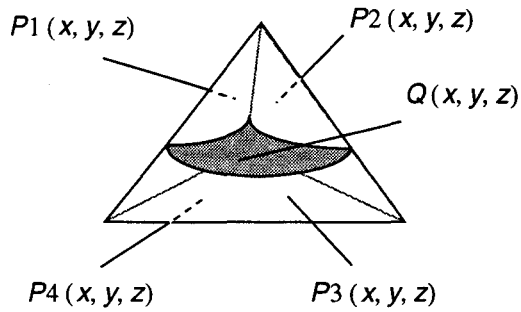
$$\bigcup_{j=1}^{p} (R_{i_j}(x, y, z) = 0) \bigcap_{i=0}^{n} (R_i(x, y, z) \geq 0) \qquad \textbf{(Eq. 3.2)}$$

avec $0 \leq i_j \leq n \ \forall j \in 1..p$

Chaque R_{i_j} est appelée *primitive visible*. Les autres sont appelées *primitives limitantes* ou *primitives de clipping*. Pour un ensemble donné de primitives, différents objets de base peuvent être définis selon les primitives qui sont visibles ou non. Toutefois, le volume englobant est systématiquement défini comme l'intersection de *tous* les demi-espaces définis par *toutes* les primitives.

3.2.3 Patch quadrique

La plupart des travaux proposant des modèles à base de quadriques utilisent des patchs quadriques. Notamment, Maxime Froumentin utilise des morceaux de quadriques inscrits dans un tétraèdre. En considérant le tétraèdre comme l'intersection de quatre demi-espaces définis par les plans supports de chaque face du tétraèdre, on fait correspondre notre objet de base avec un patch quadrique (Figure 3.2).



$$(Q(x,y,z) = 0) \cap \begin{cases} Q(x,y,z) \geq 0 \\ P1(x,y,z) \geq 0 \\ P2(x,y,z) \geq 0 \\ P3(x,y,z) \geq 0 \\ P4(x,y,z) \geq 0 \end{cases}$$

Figure 3.2 ***Un patch quadrique dans son tétraèdre***

Dans ce cas, seule la quadrique est visible, les plans sont des plans limitants. Toutefois, en considérant les plans visibles, on peut gérer les *trunctets* de Menon [Menon94].

3.2.4 Quadrique naturelle

Les quadriques naturelles sont la sphère (ou plus généralement l'ellipsoïde) le cône et le cylindre. Ces quadriques sont utilisées dans un certains nombres d'applications, notamment en modélisation CSG. Pour la sphère il n'y a pas de problème. En revanche le cylindre et le cône sont des primitives infinies dans deux directions. Il faut donc utiliser deux plans pour les "fermer".

Dans les Figure 3.3 et Figure 3.4 nous prenons le cylindre comme exemple pour illustrer les différentes possibilités offertes en fonction de la nature des plans (ie les plans sont-ils visibles ou limitants). Dans le premier cas nous supposons les deux plans comme étant limitants. On obtient donc un tronçon de cylindre sans épaisseur. Dans le deuxième cas nous avons un seul plan visible. On a alors un gobelet toujours sans épaisseur. Dans le troisième cas, les deux plans sont visibles. On obtient alors un cylindre au sens commun du terme. Remarquons que cet objet est visuellement un volume bien qu'il soit mathématiquement défini comme une union de surface. Enfin on peut éventuellement utiliser un troisième plan pour ne conserver qu'un morceau du cylindre ; sur la figure, un cylindre coupé longitudinalement.

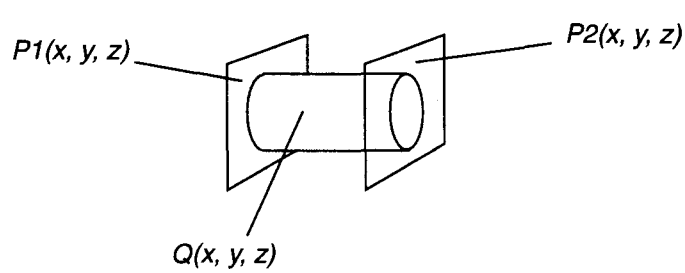


Figure 3.3 ***Un cylindre défini à partir de trois primitives***

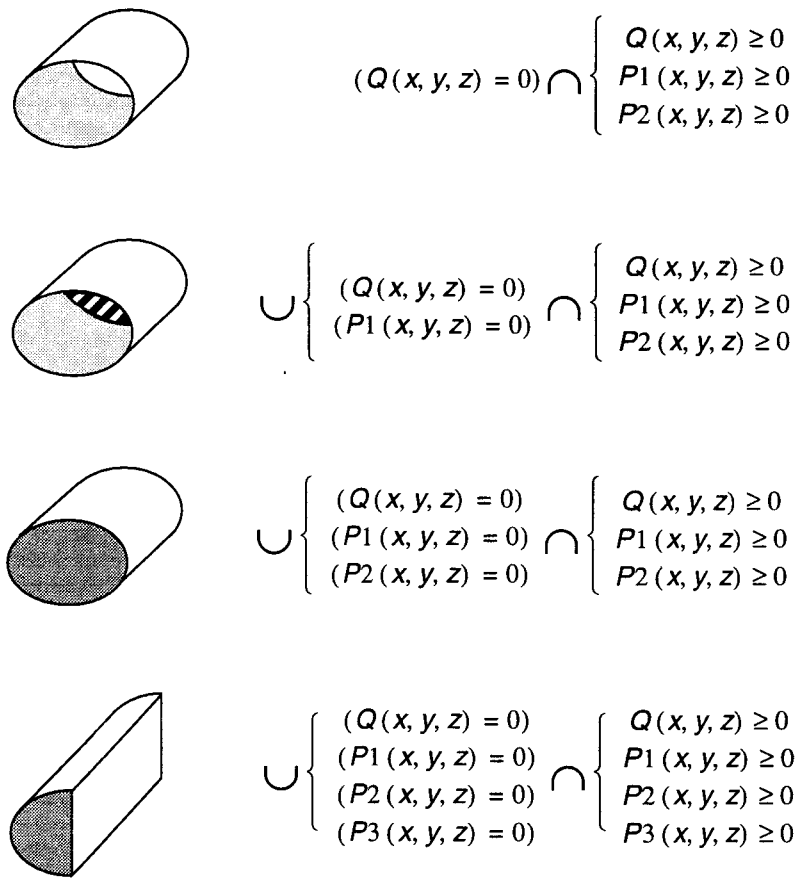


Figure 3.4 ***Différents objets cylindriques***

3.2.5 Polyèdre

En théorie, lorsque l'objet de base définit un polyèdre, la quadrique ne sert plus à rien. Nous utilisons dans ce cas n plans pour définir un polyèdre convexe à n cotés (Figure 3.5). Par exemple, un cube est défini par six plans, s'ils sont tous visibles (Figure 3.6). De nouveau on peut jouer sur la nature des plans pour obtenir par exemple une boite cubique sans couvercle ni épaisseur.

Remarquons que la quadrique peut tout de même être utilisée si elle est dégénérée du second ordre. C'est alors une paire de plans (parallèles, sécants ou confondus).

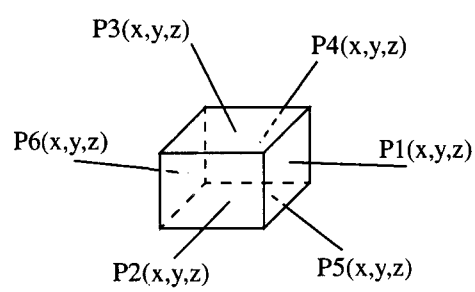
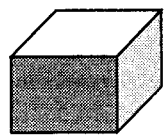
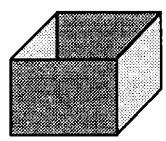


Figure 3.5 ***Six primitives planes pour former un cube***



$$\bigcup \begin{cases} (P1(x,y,z) = 0) \\ (P2(x,y,z) = 0) \\ (P3(x,y,z) = 0) \\ (P4(x,y,z) = 0) \\ (P5(x,y,z) = 0) \\ (P6(x,y,z) = 0) \end{cases} \bigcap \begin{cases} P1(x,y,z) \geq 0 \\ P2(x,y,z) \geq 0 \\ P3(x,y,z) \geq 0 \\ P4(x,y,z) \geq 0 \\ P5(x,y,z) \geq 0 \\ P6(x,y,z) \geq 0 \end{cases}$$



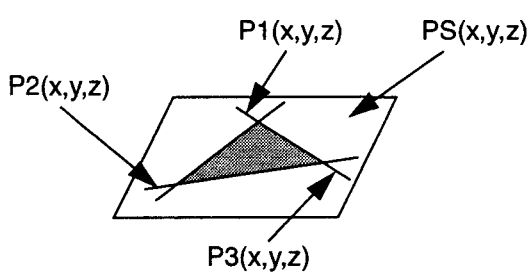
$$\bigcup \begin{cases} (P1(x,y,z) = 0) \\ (P2(x,y,z) = 0) \\ (P4(x,y,z) = 0) \\ (P6(x,y,z) = 0) \end{cases} \bigcap \begin{cases} P1(x,y,z) \geq 0 \\ P2(x,y,z) \geq 0 \\ P3(x,y,z) \geq 0 \\ P4(x,y,z) \geq 0 \\ P5(x,y,z) \geq 0 \\ P6(x,y,z) \geq 0 \end{cases}$$

Figure 3.6 ***Polyèdre : exemple avec six plans***

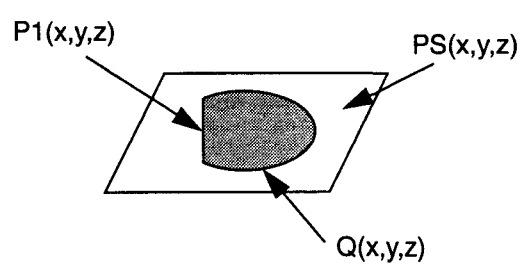
3.2.6 Facette

Nous étendons la définition usuelle de la facette comme suit : une facette est une surface plane avec zéro ou un bord conique et zéro, un ou plusieurs bords droits. La seule primitive visible, qui est un plan, est appelée *plan support*. Un bord est l'intersection de ce plan support avec une primitive limitante[1].

Cette définition englobe les facettes convexes au sens habituel du terme et notamment les triangles et quadrilatères qui sont les primitives classiques des accélérateurs graphiques.



$$(PS(x,y,z) = 0) \bigcap \begin{cases} PS(x,y,z) \geq 0 \\ P1(x,y,z) \geq 0 \\ P2(x,y,z) \geq 0 \\ P3(x,y,z) \geq 0 \end{cases}$$



$$(PS(x,y,z) = 0) \bigcap \begin{cases} PS(x,y,z) \geq 0 \\ P1(x,y,z) \geq 0 \\ Q(x,y,z) \geq 0 \end{cases}$$

Figure 3.7 ***Facettes***

1. Pour être tout à fait exact, notons que cette intersection est elle même limitée par le volume limitant.

3.2.7 Surface, volume et CSG

On s'est longtemps demandé si notre objet de base était une surface ou un volume. Mathématiquement les primitives utilisées (ie le plan et la quadrique) sont des surfaces. De plus cela correspond bien à l'approche habituelle des objets facettisés. Néanmoins les quadriques naturelles sont des volumes et on voudrait bien pouvoir gérer ces objets couramment utilisés d'autant plus que cela nous ouvrirait les portes du rendu CSG.

La réponse est que notre objet est une surface d'un point de vue mathématique mais qu'il peut être considéré soit comme une surface soit comme un volume. En fait l'algorithmique développée aux paragraphes 3.3 et 3.4 ne fait pas la différence, c'est l'utilisateur qui la fait. Ceci nous permet de gérer aussi bien l'approche surfacique que le rendu direct CSG.

En rendu projectif temps-réel, l'objet est habituellement une surface. Cependant, en CSG, on considère les objets comme des volumes. Comme nous l'avons vu (cf 2.3.2) la structure des arbres CSG rend très difficile l'affichage temps réel. Toutefois notre système peut facilement faire du rendu CSG. Dans notre architecture les primitives sont converties par le processeur quadrique. Ensuite la gestion de l'arbre (la classification) peut être réalisé lors du post-traitement.

3.2.8 Le choix du nombre de plans

On s'est longtemps demandé combien de plans de coupes on devait prévoir dans notre processeur quadrique. Bien sûr, plus il y en a et plus on a de possibilités au niveau de notre objet de base. Le nombre minimum est de quatre pour pouvoir gérer au moins les patchs quadriques. Aller au delà de huit pour les polyèdres ne semblent pas intéressant. Mais il nous faut également tenir compte des contraintes d'implémentations matérielles.

Nous avons choisi de nous limiter à quatre plans. En effet on peut facilement gérer les patchs quadriques, les quadriques naturelles et les facettes. Pour les polyèdres, on se restreint mais on garde suffisamment de souplesse. En effet si on utilise la quadrique comme quadrique dégénérée on dispose de un ou deux plans supplémentaires. On peut donc faire tous les polyèdres à six faces, notamment les parallélépipèdes, ce qui est déjà très satisfaisant. Pour les polyèdres plus complexes (et donc plus rares) on peut décomposer en plusieurs polyèdres plus simples. En choisissant de ne gérer que quatre plans, l'économie sur le processeur est substantielle (cf. paragraphe 4.2).

3.3 Rendu de l'objet quadrique

Nous allons traiter dans cette partie des algorithmes de rendu élémentaire. Les algorithmes habituels nécessitent une adaptation importante afin de pouvoir être utilisables. Pour la conversion objet-pixel nous utilisons une méthode par équation. Pour l'élimination des parties cachées, nous calculons les profondeurs de chaque primitive pour être en mesure de réaliser l'algorithme du z-buffer. Enfin nous calculons en tout point les éléments nécessaires pour calculer l'éclairement dans le post-processeur: la normale à l'objet et le point de l'objet dans le repère monde.

3.3.1 Calcul des profondeurs

La quadrique

Le calcul des profondeurs d'une quadrique se déduit des équations (Eq. 2.1) et (Eq. 2.2). On obtient :

$$Az^2 + Bz + C = 0 \qquad \text{(Eq. 3.3)}$$

avec $A = c$, $B = ex + fy + i$ et $C = ax^2 + by^2 + dxy + gx + hy + j$.

On a une équation du second degré en z. Ils existent donc deux solutions réelles à cette équations lorsque la valeur du discriminant $\Delta = B^2 - 4AC$ est strictement positive. Les deux racines sont de la forme :

$$Z_f = \frac{-B - \sqrt{\Delta}}{2C} \text{ et } Z_b = \frac{-B + \sqrt{\Delta}}{2C} \qquad \text{(Eq. 3.4)}$$

Z_f et Z_b sont respectivement les profondeurs avant (front) et arrière (back) de la quadrique. On peut récrire les expressions de Z_f et Z_b de la façon suivante :

$$Z_f = f_1(x, y) - \sqrt{f_2 x, y} \text{ et } Z_b = f_1(x, y) + \sqrt{f_2 x, y}, \qquad \text{(Eq. 3.5)}$$

avec $f_1(x, y)$ une expression du premier degré en x, y et $f_2(x, y)$ une expression du second degré en x, y :

$$f_1(x, y) = \frac{-e}{2c}x + \frac{-f}{2c}y + \frac{-i}{2c} \qquad \text{(Eq. 3.6)}$$

$$f_2(x, y) = \frac{e^2 - 4ac}{4c^2}x^2 + \frac{f^2 - 4bc}{4c^2}y^2 + \frac{ef - 2cd}{2c^2}xy + \frac{ei - 2cg}{2c^2}x + \frac{fi - 2ch}{2c^2}y + \frac{i^2 - 4cj}{4c^2} \qquad \text{(Eq. 3.7)}$$

Les plans

Soit l' équation implicite d'un plan :

$$ax + by + cz + d = 0 \qquad \text{(Eq. 3.8)}$$

En tout point la profondeur s'exprime sous forme d'une expression du premier degré en x, y :

$$z = -\left(\frac{ax + by + d}{c}\right) \qquad \text{(Eq. 3.9)}$$

3.3.2 La conversion objet-pixel

Pour savoir si un pixel est recouvert ou non par l'objet de base, nous déterminons s'il se trouve ou non à l'intérieur de la silhouette de la quadrique. Or l'expression de la silhouette de la quadrique dans le plan écran est $f_2(x, y)$. On fait donc d'une pierre deux coups : en calculant les profondeurs de la quadrique on détermine si elle est présente au pixel considéré.

Cette méthode s'apparente aux techniques de conversion objet-pixels par équations et test d'inclusion. Ces techniques, du moins sous une forme brute, testent tous les pixels de l'écran. Etant donné que la primitive ne couvre en général qu'une petite zone de l'écran (de quelques dizaines à quelques centaines de pixels) un grand nombre de tests sont faits inutilement. Pour éviter ce problème on cherche à réduire cette zone.

Pour cela nous définissons une zone de l'écran (appelée *zone englobante*) qui inclue la projection de la primitive (Figure 3.8). Dans le cas général on la définie comme étant le plus petit rectangle contenant la projection de la boite englobante de la primitive (on parle alors de *rectangle englobant*).

Mais dans certains cas on peut trouver une zone englobante plus petite. Pour le patch quadrique sa boite englobante est le tétraèdre qui le contient. Dans ce cas il serait intéressant d'utiliser directement la projection du tétraèdre comme zone englobante. En effet le nombre de

points testés inutilement est largement réduit. Toutefois le parcours de cette zone est quelque peu plus complexe que le parcours du rectangle englobant.

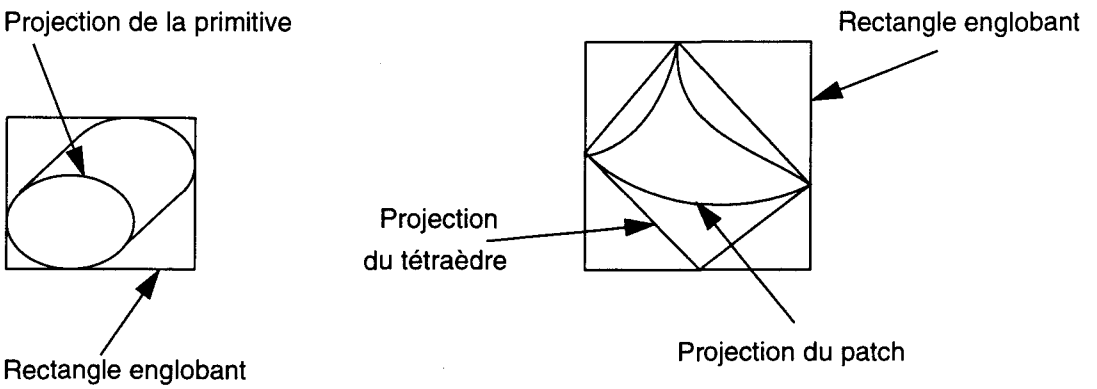


Figure 3.8 ***Zone englobante : cas général (à gauche) et cas du patch (à droite)***

3.3.3 Algorithme de sélection de profondeur

Contrairement à la facette qui n'a qu'une profondeur, notre objet de base a $n+2$ profondeurs si n est le nombre de primitives planes (soit deux pour la quadrique et une par plan). La question qui se pose alors est : quelle est la bonne. En d'autres termes, laquelle va être envoyée au Z-buffer. Nous présentons ici le moyen de sélectionner la bonne profondeur.

En fait cela revient à faire une détermination de la surface visible au sein de l'objet de base. Pour cela nous allons procéder en trois étapes. Tout d'abord l'objet est coupé en deux demi-objets. Ensuite l'objet est construit "plan par plan", de façon itérative. Enfin, l'objet étant construit, il ne reste qu'à sélectionner la bonne profondeur, qui correspond à la première primitive visible.

Chaque primitive peut être soit visible soit invisible (ie primitive limitante). La bonne profondeur doit forcément correspondre à une primitive visible. Il faut donc conserver cette information. Pour cela on associe un bit à chaque primitive R_i, appelé *bit de visibilité* et noté V_i. Par convention, le bit de visibilité V_i vaut un pour une primitive visible et zéro dans le cas contraire. Pour l'objet de base en cours de construction, on associe également un bit de visibilité à chacune de ses profondeurs.

Mais il ne suffit pas de savoir quelle est la bonne profondeur, il faut également savoir à quelle primitive elle correspond. En effet cette information est essentielle pour savoir quelle est la normale qui correspond. Pour chaque profondeur de l'objet que l'on construit on associe donc un numéro de primitive. Ce numéro de primitive est susceptible de changer dans les mêmes circonstances que le bit de visibilité. Pour des raisons de simplicité, on "oublie" le numéro de primitive dans la présentation des algorithmes suivants.

Etape 1 : l'objet est coupé en deux

Intuitivement, l'objet de base peut être vu comme un volume (le volume englobant) dont les surfaces peuvent être visibles (primitive visible) ou transparentes (primitive limitante). Ce volume est l'intersection de demi-espaces définis soit par un plan soit par une quadrique. Or du fait de la primitive quadrique, ce volume peut être concave. En terme de profondeur, l'objet en un pixel donné peut avoir quatre profondeurs (au maximum). Cette caractéristique déjà

remarquée dans [Nyiri92a] nous oblige à couper l'objet en deux demi-objets qui sont de façon certaine convexes. Chaque demi-objet possède deux profondeurs.

Ce découpage se fait à partir de la primitive quadrique (puisque c'est elle qui fait apparaître le problème). Il y a quatre cas à envisager, qui dépendent de deux paramètres binaires : le signe du coefficient c dans l'équation (Eq. 2.1) qui nous indique quel demi-espace nous intéresse ; le signe de l'expression de $f_2(x, y)$ qui indique si l'on se trouve à l'intérieur de la silhouette de la quadrique ou non.

L'algorithme de la Figure 3.9 nous donne l'initialisation de zf1, zb1, zf2, zb2 qui sont respectivement les profondeurs avant et arrière du premier et du second demi-objet, ainsi que leur bit de visibilité associé vf1, vb1, vf2, vb2. ZMIN et ZMAX sont respectivement les profondeurs minimales et maximales de la scène.

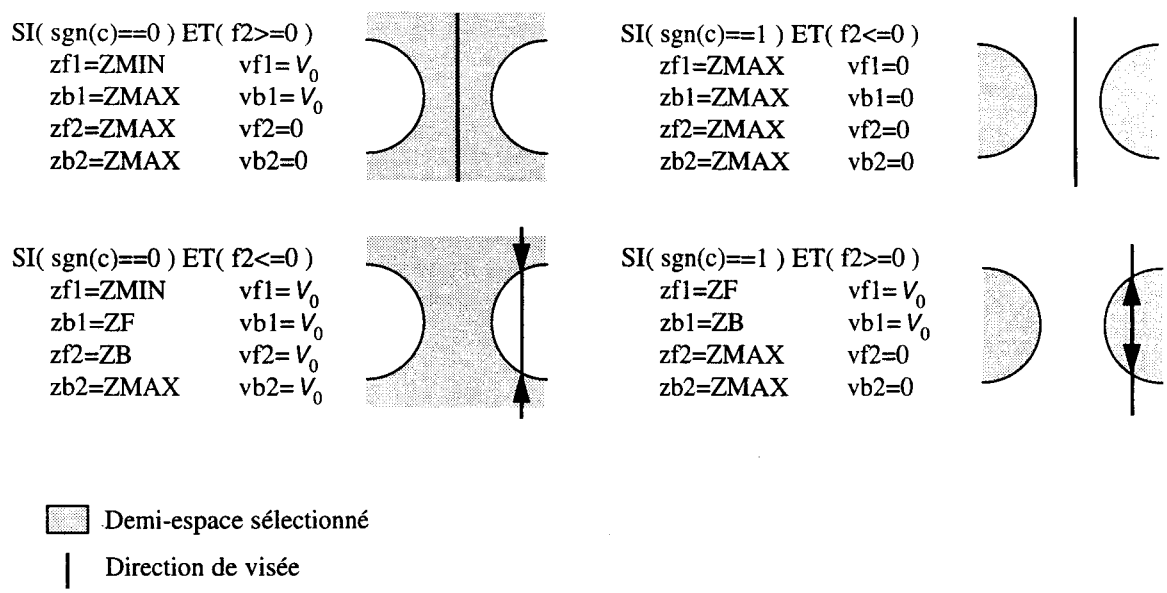


Figure 3.9 ***Découpage de l'objet de base en deux demi-objets***

Etape 2 : construction itérative de l'objet

Cette étape consiste à construire l'objet en prenant en compte successivement chaque plan. Elle est basée sur des comparaisons entre la profondeur du plan considéré et les profondeurs de l'objet en cours de construction.

Un paramètre indispensable à ces comparaisons est l'orientation des plans. Un plan est soit perpendiculaire au plan de l'écran (plan perpendiculaire) soit orienté vers l'écran (plan avant) soit orienté dans le sens contraire (plan arrière).

Les comparaisons donne la position relative du plan par rapport à l'objet. En fonction de cette position et de l'orientation du plan, les profondeurs de l'objet sont modifiées ou non. Si elles sont effectivement modifiées, les bits de visibilité associés sont également modifiés.

Soit zf et zb les profondeurs avant et arrière d'un demi-objet, ZPi la profondeurs du plan i et vf, vb et VPi les bits de visibilité associés, la Figure 3.10 présente l'algorithme à effectuer pour chaque plan sur chaque demi-objet.

```
SI plan perpendiculaire
    SI (ZPi<0)
        zf=zb=ZMAX; vf=vb=0;
SI plan avant
    SI (ZPi>zb)
        zf=zb=ZMAX; vf=vb=0;
    SINON SI (ZPi>zf)
        zf=ZPi; vf=VPi;
SI plan arrière
    SI (ZPi<zf)
        zf=zb=ZMAX; vf=vb=0;
    SINON SI (ZPi<zb)
        zb=ZPi; vb=VPi;
```

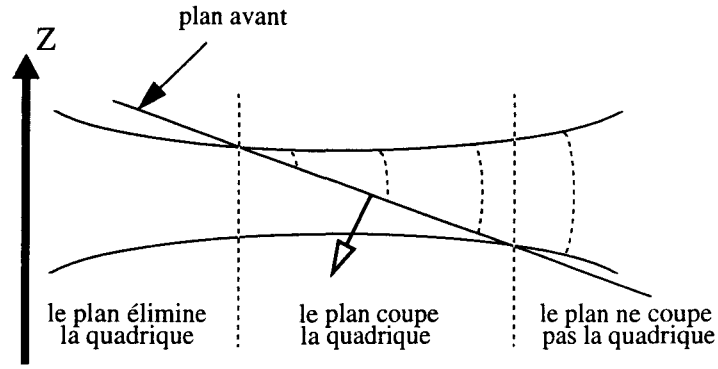


Figure 3.10 ***Construction itérative de l'objet***

Etape 3 : on récupère la bonne profondeur

A la fin de la deuxième étape, on a les quatre profondeurs de l'objet avec leur visibilité. La bonne profondeur est tout simplement la première profondeur (ie la plus petite) visible. Comme l'on sait que *zf1<zb1<zf2<zb2*, l'algorithme de sélection est le suivant :

```
SI (vf1==1)          ALORS  Z1=zf1;    Z2=zb1;
SINON SI (vb1==1)    ALORS  Z1=zb1;    Z2=zb1;
SINON SI (vf2==1)    ALORS  Z1=zf2;    Z2=zb2;
SINON SI (vb2==1)    ALORS  Z1=zb2;    Z2=zb2;
SINON                       Z1=ZMAX;   Z2=ZMAX;
```

Figure 3.11 ***Algorithme de sélection de la bonne profondeur***

On peut noter que cet algorithme sélectionne deux profondeurs *Z1* et *Z2*, *Z1* correspondant à la bonne profondeur. Toutefois, si l'on veut considérer l'objet comme un volume (par exemple pour réaliser un post-traitement CSG), on a besoin de la profondeur avant (*Z1*) et arrière (*Z2*) de l'objet.

Néanmoins, le processeur ne fait aucun contrôle quant à la validité de l'objet, c'est à dire que si l'objet n'est pas un solide, il fournira deux profondeurs fausses. Par ailleurs l'objet doit être convexe car on ne fournit que les deux premières profondeurs (le premier demi-objet) et non les quatre (le premier demi-objet et le deuxième). Si cette dernière condition n'est pas assurée par l'utilisateur, celui-ci ne récupérera qu'un morceau (le premier demi-objet) de l'objet, il lui manquera le reste (le deuxième demi-objet).

3.3.4 Les normales

Tout comme pour les profondeurs il faut être en mesure de calculer la normale de chaque primitive d'une part et de sélectionner la normale à conserver d'autre part. Nous montrons que ces tâches ont des solutions simples.

La quadrique

Rappelons que les composantes de la normale en un point (x, y, z) d'une quadrique $R_0(x, y, z) = 0$ sont les dérivées partielles de l'équation qui définit la quadrique. On obtient :

$$\begin{cases} N_x = \frac{\partial}{\partial x} R_0(x, y, z) = 2ax + dy + ez + g \\ N_y = \frac{\partial}{\partial y} R_0(x, y, z) = 2by + dx + fz + h \\ N_z = \frac{\partial}{\partial z} R_0(x, y, z) = 2cz + ex + fy + i \end{cases} \quad \text{(Eq. 3.10)}$$

En un pixel donné, la quadrique a deux profondeurs et donc deux normales. La normale avant est associée à la profondeur avant et la normale arrière est associée à la profondeur arrière.

Les plans

La normale à un plan est constante. Si le plan P a pour équation $ax + by + cz + d = 0$ alors la normale est $\vec{N} = (a, b, c)$. Toutefois dans le cas d'un objet facettisé, la normale sur la facette n'est pas constante. On doit pouvoir interpoler la normale sur la facette ce qui revient à faire de l'interpolation de Phong. Dans ce cas chaque composante de la normale correspond à une expression linéaire en x, y.

Récupération de la bonne normale

Tout comme pour les profondeurs, on calcule pour un objet donné un ensemble de normales. Encore faut il retrouver la bonne, c'est à dire celle associée à la bonne profondeur. On utilise pour cela le numéro de primitive associée à chaque profondeur. En fonction de ce numéro on sélectionne la normale correspondante.

3.3.5 Calcul du point dans le repère monde

Comme nous l'avons expliqué précédemment (cf 3.1.2) la formule d'éclairement de Phong s'utilise dans un repère avant projection perspective. Or lors de la phase de rendu on travaille dans le repère écran donc après la projection perspective. Il faut donc "revenir" avant la projection, typiquement dans le repère global de la scène que nous appelons repère monde.

Dans la formule de Phong il faut connaître entre autre chose les coordonnées du point de l'objet considéré. Les coordonnées de ce point dans le repère écran sont $x_e, y_e, z_e, 1$ avec x_e, y_e les coordonnées du pixel et z_e la profondeur obtenue en sortie du module de sélection de profondeur. On peut retrouver les coordonnées du points P_m exprimés dans le repère monde au moyen du calcul matriciel. En effet on peut facilement connaître la matrice de passage du repère monde au repère écran, noté M_{me}. On a alors :

$$P_m = M_{me} P_e \quad \text{(Eq. 3.11)}$$

Chaque coordonnées x_m, y_m, z_m, w_m de P_m est une expression linéaire en x_e, y_e, z_e. Pour obtenir les coordonnées cartésiennes, il reste à diviser les trois premières coordonnées par la dernière.

3.4 Amélioration de la qualité

Aux débuts de la synthèse d'images, la qualité des images produites était assez frustre. Aujourd'hui les images produites doivent offrir une qualité bien supérieure. Pour cela un ensemble de techniques d'amélioration de la qualité ont été développées. Nous avons donc étudié leur adaptation à la primitive quadrique. Pour certaines de ces techniques, l'adaptation est immédiate car elles ne reposent pas sur la nature de la primitive de visualisation. Pour d'autres il a fallu les modifier de façon plus importante.

Nous présentons d'abord la technique que nous avons choisi pour l'anti-aliassage. Après un rapide rappel sur les techniques existantes, nous discutons des méthodes intéressantes dans notre cadre. Ensuite nous étudions les techniques de transparence. Pour les ombres portées nous repartons des travaux d'Eric Nyiri.

3.4.1 Anti-aliassage

Dès que l'on discrétise un signal continu, on s'expose au phénomène d'aliassage. En synthèse d'image ce phénomène est dû principalement à deux causes. Tout d'abord la fréquence d'échantillonnage est trop faible par rapport à la fréquence maximale de notre signal, ce qui entraîne l'apparition de basse fréquences parasites. Notre signal, c'est l'intensité (ou la couleur) qui varie selon la position à l'écran. On peut dire que ce signal a une fréquence maximale infinie lorsque l'on passe brusquement d'un objet noir à un objet blanc. Dans ce premier cas nous parlerons d'aliassage. Ensuite lorsque l'on reconstruit le signal, c'est à dire qu'on l'affiche à l'écran, le filtre de reconstruction, imparfait, introduit des hautes fréquences. On parlera de crénelage.

Ces problèmes se traduisent visuellement de différentes façons. Les deux principales sont les moirés sur les textures et les marches d'escaliers sur le contour des objet. Les deux façon d'attaquer ces problèmes sont le pré-filtrage (ou échantillonnage surfacique) et le sur-échantillonnage. Nous présentons maintenant ces techniques pour résoudre les problèmes de marches d'escaliers. Nous y reviendrons au paragraphe 3.5 pour les textures.

Ces problèmes et les techniques développées pour les résoudre sont décrits en détail dans [Preux95].

Techniques usuelles

Dans les techniques de pré-filtrage un pixel n'est plus considéré comme un point mais comme une surface. Lors du rendu d'un objet on calcule la surface du pixel qu'il couvre. Cette valeur, souvent notée α, est appelée *taux de couverture*. Pour déterminer cette valeur on utilise la distance du bord de l'objet par rapport au centre du pixel.

Toutefois pour traiter correctement les sommets de polygones il faut aussi tenir compte de la position de chaque bord dans le pixel. On peut alors combiner les bords pour obtenir le bon taux de couverture. Les techniques à *masques de sous-pixels*, souvent regroupées sous le terme générique de *A-Buffer*, offrent une bonne solution.

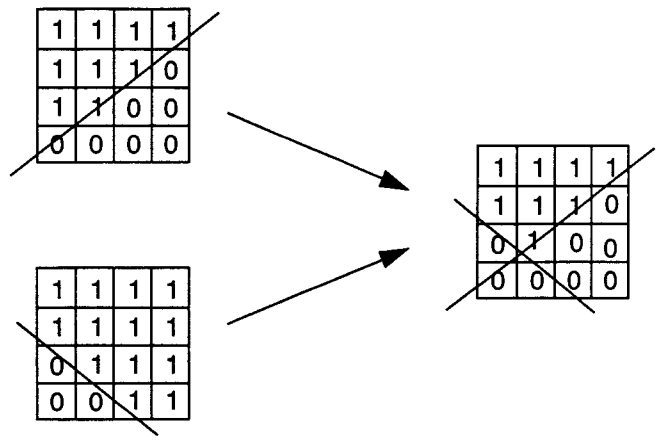


Figure 3.12 ***Méthode à masque de sous-pixels***

Toutefois il reste des problèmes pour l'élimination des parties cachées (on ne conserve normalement qu'une seule profondeur par pixel, or plusieurs objets peuvent contribuer à la couleur du pixel) et lorsqu'il y a des facettes qui se coupent dans le pixels (problèmes des *arêtes implicites*).

Le sur-échantillonnage consiste à prendre N échantillons là où l'on en prend normalement qu'un seul. A l'écran un pixel est donc décomposé en $N \times N$ sous-pixels. On peut alors faire varier le nombre de sous pixels (échantillonnage 2×2 jusqu'à 16×16), la façon dont on les choisit (échantillonnage régulier ou stochastique) et si même on les choisit tous (par exemple calcul de 8 sous-pixels dans une grille de 64). Ensuite on détermine la couleur du pixel en effectuent une moyenne (pondérée ou non) des sous-pixels.

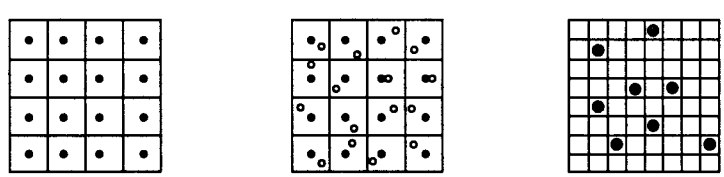


Figure 3.13 ***Echantillonnage régulier, stochastique et sélection de 8 échantillons parmi 64***

Ce type de méthode ne résout pas le problème d'aliassage, elle ne fait que le repousser. Toutefois si on le repousse assez loin, on ne voit plus de défaut. Cette technique est souvent plus coûteuse que celle à pré-filtrage mais s'adapte mieux à une implémentation matérielle. Pour cela ils existent deux méthodes. Soit on duplique tous les éléments de rendu (interpolateurs et mémoire) ce qui revient très cher mais évite de dégrader les performances. La deuxième méthode, appelée *Accumulation-buffer*, est une méthode multi-passes. Elle évite de répliquer les éléments matériels mais nécessite n fois plus de temps si n est le nombre de sous-pixels par pixel.

Quelle méthode dans notre contexte

La question est de savoir quelle méthode nous devons choisir pour notre système basé sur la quadrique. Une discussion des mérites et inconvénients de chacune des méthodes nous permet d'éliminer le pré-filtrage. Pour le sur-échantillonnage, l'adaptation est immédiate.

Nous éliminons la technique de pré-filtrage pour plusieurs raisons. Tout d'abord il n'est absolument pas simple d'adapter cette méthode à la quadrique. En effet elle repose sur des considérations géométriques, notamment la distance d'un point à une droite. Or pour la quadrique la droite se transforme en conique. Il faut d'abord déterminer la conique (faisable mais cela suppose des calculs en plus), puis calculer la distance du pixel à la conique ce qui n'est pas simple. Le problème suivant est que ces méthodes ne gère pas les arêtes implicites. Or notre objet de base de par sa définition en contient. On risque donc de souffrir fortement de ce problème. Enfin ces méthodes ne sont pas très facile à implémenter matériellement. Aujourd'hui aucune machine utilisant les techniques de A-buffer n'a été réalisée.

A contrario le sur-échantillonnage est facile à implémenter matériellement. Les deux méthodes citées précédemment ont été utilisées dans des accélérateurs graphiques du commerce. D'autre part le sur-échantillonnage gère les arêtes implicites, ce qui est important pour nous. Enfin cette technique est indépendante de la primitive de visualisation. L'adaptation à la quadrique est donc immédiate.

3.4.2 Transparence

Les effets de transparences, que l'on rencontre dans beaucoup de scènes, sont lourds à gérer si l'on veut le faire correctement. Usuellement on distingue la transparence sans réfraction (ie les rayons lumineux ne sont pas déviés lorsqu'ils traversent le matériaux) et la transparence avec réfraction, beaucoup plus difficile à gérer. Nous allons nous intéresser à la première catégorie, présenter les deux façons de la gérer, puis nous verrons comment implémenter ces méthodes. Pour finir nous remarquons que ces méthodes sont applicables aussi bien aux quadriques qu'aux polygones.

Généralement on utilise deux formules pour calculer l'intensité (la couleur) d'un pixel lorsque celle-ci dépend d'objets transparents. Soit un polygone A, transparent, placé devant un polygone B opaque. La première méthode consiste à interpoler l'intensité :

$$I = (1 - k_A)\, I_A + k_A I_B \qquad \textbf{(Eq. 3.12)}$$

où I_A et I_B sont les intensités de chaque polygone pour le pixel considéré et k_A est le coefficient de transparence du polygone A.

La seconde méthode considère la surface transparente comme un filtre à travers lequel la surface opaque est vue :

$$I = I_A + k_A O_A I_B \qquad \textbf{(Eq. 3.13)}$$

où O_A est la couleur de l'objet transparent.

L'implémentation de ces méthodes supposent que l'on ait la liste triée des objets au pixel considéré, ce qui n'est pas le cas dans le cadre d'un système à Z-buffer. Une méthode qui se passe de la liste triée mais qui est inexacte, consiste à rendre tout d'abord tous les polygones opaques et à déterminer ainsi une couleur pour le pixel qui est ensuite combinée avec celles des objets transparents. Cette méthode est exacte lorsqu'il n'y a qu'un seul objet transparent sur le pixel.

Une autre méthode, qui réalise une interpolation permet de gérer plusieurs objets transparents au même pixel. Il s'agit de la *screen-door transparency*. On considère que l'objet transparent est en fait une espèce de treillis. Là où il y a le treillis l'objet est considéré opaque, là où l'on est entre les mailles du treillis l'objet est considéré absent. L'oeil humain va intégrer spatialement ces points opaques et ces trous et aura l'impression de transparence. Cette méthode a le gros avantage d'être compatible avec le Z-buffer. Néanmoins elle pose des problèmes dus au treillis. Par exemple un objet ayant le même treillis qu'un autre risque de le cacher complètement. Par ailleurs cette méthode peut être appliquée au niveau des sous-pixels ce qui rend l'algorithme indépendant de la surface couverte par l'objet transparent.

Mammen [Mammen89] utilise plusieurs mémoires de profondeur pour résoudre correctement le problème de transparence. Il effectue tout d'abord une première passe pour le rendu des objets opaques. Ensuite il effectue une passe par objet transparent devant l'objet opaque présent au pixel considéré. Pour chacune de ces passes il détermine (grâce au second tampon de profondeur) l'objet transparent le plus proche de l'objet opaque (et devant celui-ci). Il effectue alors l'interpolation de couleur. Cette méthode bien que correcte coûte cher en mémoire et en temps de calcul (besoin de plusieurs passes).

On peut remarquer que toute ces méthodes ne dépendent pas de la primitive de visualisation. On peut donc les utiliser dans notre système de visualisation à base de quadrique. La *screen-door transparency* ainsi que la méthode de Mammen peuvent être intégrées dans notre système, la première offrant la simplicité la seconde offrant la qualité.

3.4.3 Ombre portée

Le phénomène lumineux qu'est l'ombre est complètement intégré par l'homme dans sa vision. Il s'en sert notamment pour connaître la position relative des objets. Ainsi un véhicule se déplaçant sur une route donne l'impression de voler légèrement au dessus si son ombre sur cette route n'est pas représentée.

Pour améliorer la qualité des images il est donc important de restituer ce phénomène d'ombre[1]. Les différentes méthodes développées à cette fin peuvent être classées en cinq familles : génération à balayage de ligne, méthode à deux-passes (polygones d'ombre), traitement par volumes d'ombre, méthode à Z-buffer et lancer de rayons [Foley90] [Bergeron86].

On élimine trivialement les deux premières et la cinquième qui sortent complètement de notre cadre de travail. Nous présentons tout d'abord les deux méthodes restantes puis l'adaptation du traitement par volume d'ombre à la quadrique proposée par E. Nyiri. Nous reprenons cette méthode et nous l'améliorons. Enfin nous concluons.

Traitement par volumes d'ombre

L'idée de base est que chaque objet crée un volume d'ombre [Crow77]. Tout objet se trouvant dans le volume d'ombre d'un autre ne reçoit donc pas de lumière directement de la source. Son éclairement se limite donc à la composante ambiante.

Il faut donc, dans un premier temps, calculer l'ensemble des volumes d'ombre. Lors du rendu de la scène on teste si le point à afficher se trouve dans l'un des volumes d'ombres (ce test d'inclusion est géré par Crow grâce à un compteur qui indique combien de fois on est entré et sorti d'un volume d'ombre entre l'oeil et le point testé).

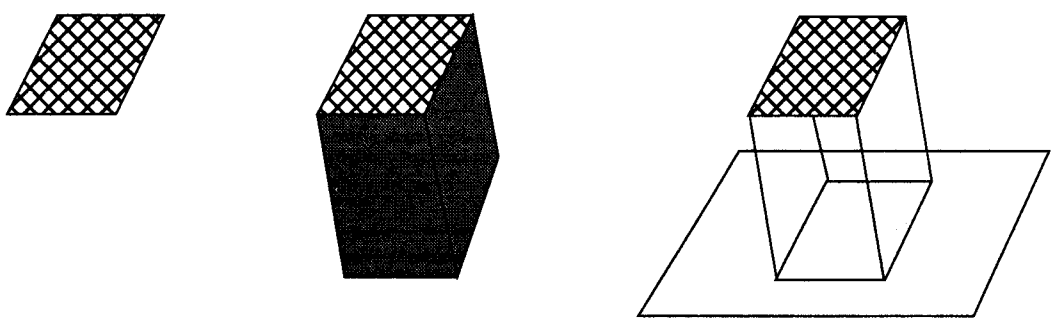


Figure 3.14 ***Un objet, son volume d'ombre et som ombre portée sur un plan***

1. Nous ne parlerons pas du phénomène de pénombre, encore plus difficile à prendre en compte. Pour plus de détails, voir [Woo90].

Méthode à Z-buffer

Williams a proposé une méthode basée sur le tampon de profondeur [Williams78]. L'idée de base est que seuls les points qui sont "vus" par la source lumineuse sont éclairés. Les autres sont dans l'ombre. Il utilise donc, dans un premier temps, le Z-buffer pour rendre la scène du point de vue de la lumière et ainsi déterminer tous les points éclairés. Ensuite lors de la phase de rendu "normale", à chaque fois qu'un point est visible par l'observateur il teste si ce point est également visible par la lumière. Si c'est le cas il l'éclaire.

Cette méthode très simple en soit possède quatre inconvénients : tout d'abord il faut un tampon de profondeur par source lumineuse. Ensuite il faut rendre la scène autant de fois qu'il y a de source (en plus du rendu pour l'observateur). De plus chaque point visible par l'observateur est exprimé dans le repère observateur. Pour faire le test d'ombre il faut repasser ce point dans le repère de la source lumineuse. Enfin les problèmes d'aliassage sont très importants.

Proposition de Nyiri

Les premières études sur la quadrique au LIFL ont mené Eric Nyiri à choisir la méthode de Crow pour les quadriques. En effet, comme nous l'avons vu notre système gère les volumes. Le test d'inclusion est donc très simple : il suffit qu'un point soit compris entre la profondeur minimale et la profondeur maximale d'un volume d'ombre pour être à l'ombre. Dans un premier temps il faut rendre les objets de la scène. Ensuite on rend les volumes d'ombres. Si le point visible en un pixel donné est dans un volume d'ombre il est marqué. Cette marque servira lors de la phase de post-éclairement.

Toutefois cette apparente simplicité n'est pas toujours vraie. En effet le volume d'ombre n'est pas toujours simple. Pour les quadriques naturelles le volume d'ombre V_N s'exprime ainsi :

$$V_N = V_0 \bigcup_{i=1\ldots n} V_i \qquad \textbf{(Eq. 3.14)}$$

Les choses se compliquent pour le patch quadrique dont le volume d'ombre V_P s'exprime ainsi :

$$V_P = V_0 \bigcup_{i=1}^{F+B} V_i - \bigcup_{i=1}^{F} (V_{fi} \cap V_q) - \bigcup_{\substack{i=1 \\ j=1}}^{\substack{F \\ B}} (V_{fi} \cap V_{bj}) \qquad \textbf{(Eq. 3.15)}$$

avec F le nombre de plans avants, B le nombre de plans arrières. Les volumes élémentaires (V_0, V_q, les V_i, les V_{fi} et les V_{bj}) sont construits à partir de la projection de chaque primitive coupée par les autres primitives. On obtient ainsi l'équivalent d'une facette (au sens donné au paragraphe 3.2.6). Si la source lumineuse est à l'infini le volume d'ombre s'obtient par l'extrusion de la facette dans la direction de la lumière. Si la source n'est pas à l'infini on effectue une transformation perspective sur les primitives on détermine le volume d'ombre sur lequel on applique la transformation perspective inverse.

Dans le cas d'une quadrique naturelle le volume d'ombre est donc l'union de cinq volumes simples. Si le point testé appartient à l'un de ces volumes simples alors il est à l'ombre. Dans le cas d'un patch, les intersections qui apparaissent supposent le stockage de deux primitives. De plus le test d'inclusion se complique. Dans le cas général, si l'on veut gérer un objet quelconque la complexité augmente encore.

Notre apport

Dans le cas général il faut appliquer l'équation à chacune des primitives. Le volume d'ombre devient alors relativement complexe. Pour pouvoir le gérer, on a deux possibilités. Si l'on dispose d'un système qui gère le CSG alors on peut profiter de ces mécanismes pour les ombres

portées. On pourra d'ailleurs utiliser la méthode proposée dans [Jansen90] pour générer les ombres portées des objets CSG.

Si on ne dispose pas de tels mécanismes, on peut en faire une implémentation partielle en tenant compte de la spécificité de notre volume d'ombre. En effet ce volume est décrit par un arbre CSG qui est toujours le même et de taille connue à l'avance, ce qui simplifie les mécanismes. De plus si l'on ne gère pas les primitives séparément on s'aperçoit que certains sous arbres peuvent être simplifiés.

Conclusion

Il est particulièrement remarquable qu'aujourd'hui aucun accélérateur n'intègre de réelles possibilités de gestion des ombres. Les seules exceptions se contentent de texture d'ombre [Segal92] ou d'ombre uniquement sur le sol et d'un seul objet (typiquement un char dans les simulateurs militaires). Cette lacune s'explique. En effet les diverses méthodes envisageables sont très coûteuses.

La solutions que nous proposons est également coûteuse. Toutefois elle est parfaitement réalisable. Elle suppose un pré-traitement pour la génération des volumes d'ombres et un post-traitement avec un rendu en plusieurs passes. Remarquons que les mécanismes à mettre en place peuvent être partagés avec d'autres taches notamment le rendu CSG ou la transparence (méthode de Mammen).

3.5 Texture

Une *texture* est un ensemble d'informations qui permet de prendre en compte ce que l'on peut regrouper sous le terme de *détails de surfaces*. Ce terme englobe aussi bien l'aspect rugueux d'un objet (par exemple du bois ou de la pierre) que la présence d'un motif détaillé qui se répète. Cette définition se généralise à des images quelconques, des dessins ou des photographies. Dans ce dernier cas, on parle de *photo-texture*.

Le *placage de texture*[1] [Heckbert86] est une opération qui consiste à associer à tout point donné d'un objet l'élément de la texture qui y correspond. Cette opération a pour but de faire varier un des paramètres photométriques de l'objet afin de rendre compte visuellement du détail de surface. Les paramètres sur lesquels on peut jouer sont nombreux. On citera notamment la couleur des objets (le plus utilisé), la normale (technique dite de *bump-mapping* [Blinn78a]), la transparence [Gardner85] et les reflets (*environment mapping* [Greene86]).

L'espace de texture est généralement de dimension deux, typiquement une image plane. La notion de texture a été généralisée aux *textures solides*, tri-dimensionnelle [Peachey85] [Perlin85]. Par ailleurs la texture peut être représentée soit par une fonction mathématique, soit sous forme d'un tableau.

En synthèse d'images temps-réel le placage de textures planes stockées dans un tableau est pratiquement le seul type de placage utilisé. En effet, les textures solides exigent une taille mémoire trop importante pour les accélérateurs d'aujourd'hui. Les textures représentées par une fonction s'adaptent mal au temps-réel car leurs calculs sont trop complexes. Ils sont donc trop longs et de plus ils sont trop difficiles à implémenter matériellement. Toutefois ce dernier point est résolu dans notre système car ces fonctions peuvent être calculées lors du post-traitement.

Par ailleurs, la technique de bump-mapping, basée sur la perturbation de la normale, peut facilement s'adapter à notre système car on dispose de la normale calculée en tout point. L'environment-mapping s'adapte tout aussi bien puisque les informations nécessaires, à savoir la normale et la position du point dans le repère monde sont fournies.

1. texture-mapping en anglais

Nous présentons la technique classique de placage de texture utilisée dans les accélérateurs graphiques. Nous montrons que, bien que la méthode puisse être utilisé pour la quadrique, elle ne convient pas. Nous présentons alors une méthode dite "à deux passes". Nous l'exploitons pour apporter une solution au placage de texture en temps-réel sur des quadriques.

Le placage de texture souffre particulièrement de la nature discrète des algorithmes de rendu. De nombreuses techniques d'anti-aliassage ont donc été développées. Nous présentons les méthodes de pré-filtrage et celles à sur-échantillonnage. Nous montrons que ces deux classes de méthodes s'adapte à la primitive quadrique.

3.5.1 Le placage de texture classique

Nous présentons tout d'abord la technique classique de placage utilisée avec les triangles. Nous expliquons pourquoi nous ne choisissons pas cette technique bien qu'elle s'adapte à la quadrique.

Principe

Le placage de texture se fait en deux étapes. Tout d'abord on passe de l'espace de texture à l'objet. C'est la phase de *paramétrisation*. Ensuite on passe de l'espace objet à l'espace écran. C'est la phase de *projection*. Habituellement, les deux étapes sont regroupées et effectuées dans l'ordre inverse, c'est-à-dire qu'à partir d'un pixel de l'écran, l'élément de texture associé est retrouvé. C'est le *placage inverse*[1].

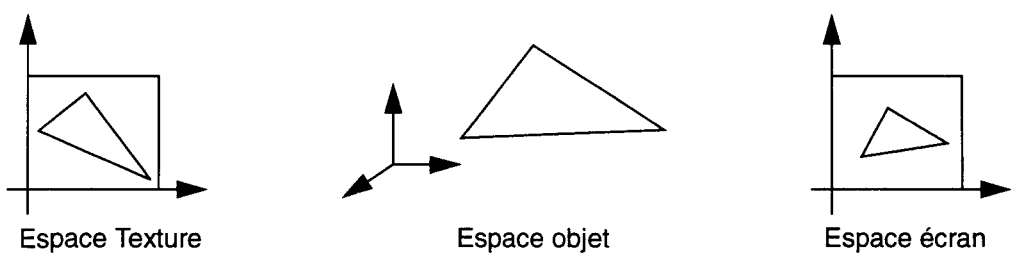


Figure 3.15 ***Le placage de texture***

Pour un triangle la phase de paramétrisation consiste à associer à chaque sommet de la facette un point de l'espace de texture (appelé *texel*), de coordonnées u, v. En général la projection est perspective (la projection orthographique est un cas particulier).

D'un point de vue mathématique, les coordonnées u, v du texel associé au pixel x, y, z peuvent être exprimées par des fonctions homographiques :

$$u = \frac{ax+by+c}{gx+hy+i} \qquad v = \frac{dx+ey+f}{gx+hy+i} \qquad \textbf{(Eq. 3.16)}$$

Dans les accélérateurs graphiques, ces valeurs sont interpolées sur la facette. La technique simple (mais fausse) consiste à ne réaliser qu'une *interpolation linéaire* de u et v. La technique exacte consiste à interpoler les numérateurs et le dénominateur (qui est commun) des équations (Eq. 3.16). Les deux divisions sont ensuite effectuées en chaque pixel (ce qui est coûteux). On parle d'interpolation *rationnelle linéaire* [Heckbert91].

1. inverse mapping

Adaptation à la quadrique

La première idée pour faire du placage de texture sur des quadriques est d'utiliser une paramétrisation de la quadrique. Cette méthode pose deux problèmes majeurs. Tout d'abord les quadriques sont généralement paramétrisées en fonction de leur type, ce qui suppose la connaissance du type de la quadrique. Or nous manipulons les quadriques sous forme algébrique donc le type est inconnu et difficile à déterminer. De plus la paramétrisation est trop complexe pour une implémentation matérielle. Nous avons donc rejeté cette méthode.

3.5.2 Méthode à deux passes

Nous présentons tout d'abord le principe du placage de texture en deux passes. Ensuite nous exploitons le principe pour proposer une méthode de placage de texture sur des objets quadriques. Enfin nous présentons comment le texel est déterminé.

Principe de la méthode

Dans [Bier86] une technique est développée pour plaquer des textures planes sur des surfaces quelconques avec peu de distorsion. Le placage est réalisé en deux temps. Tout d'abord la texture plane est plaquée sur une surface intermédiaire dont on connaît une paramétrisation (*S-mapping*). Ensuite la surface intermédiaire est plaquée sur l'objet (*O-mapping*). Cette dernière étape ne nécessite pas d'avoir une représentation paramétrique de l'objet.

Bier et Sloan ont étudié quatre surfaces intermédiaires (le plan, le cylindre, la sphère et le cube) et trois méthodes pour plaquer la surface intermédiaire sur l'objet (Figure 3.16).

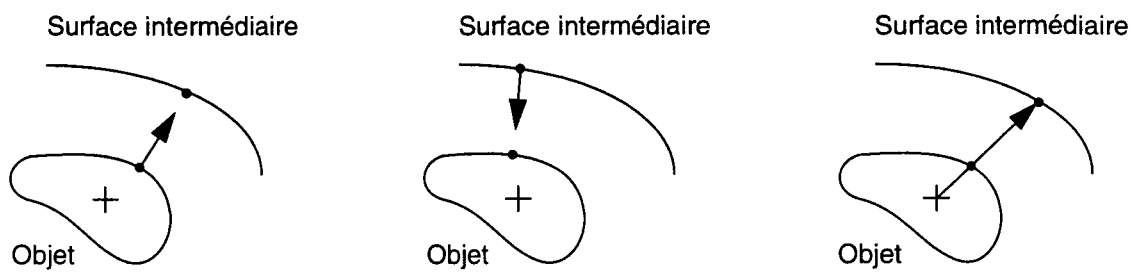


Figure 3.16 ***Placage : en utilisant la normale à l'objet (à droite), la normale à la surface intermédiaire (au centre ou le vecteur formé par le centre de l'objet et le point considéré***

En combinant les surfaces intermédiaires et la méthode de placage ils obtiennent donc douze combinaisons qu'ils évaluent selon la possibilité de trouver le texel correspondant à un pixel donné et la continuité. Ils ne conservent que cinq possibilités (cf. [Bier86] p.46).

L'approche de Bier et Sloan est intéressante car elle fonctionne sur tout type d'objet donc a fortiori sur les quadriques. Toutefois il faut pousser leur étude plus loin pour tenir compte (en plus de la distorsion et de la continuité) de la forme de nos objets ainsi que des contraintes matérielles.

Position du problème

Notre objectif est de trouver la surface intermédiaire et une méthode de placage (c'est-à-dire de S-mapping) qui satisfont à la fois nos contraintes de qualité visuelle et les contraintes d'implémentation matérielle. Deux autres paramètres entrent en ligne de compte. Le type de texture : en effet on remarque qu'une discontinuité dans un motif fort bruité sera peu visible

car "noyé" dans le bruit. La forme des objets sur lesquels on plaque la texture. Nous les classons en deux catégories : les objets *à peu près plats* et les objets *sphéroïdes*.

Ensuite nous en déduisons que deux types de surfaces intermédiaires sont nécessaires, un par type d'objet. Puis nous sélectionnons une surface intermédiaire par type. Nous montrons alors que les contraintes matérielles sont respectées ainsi que les contraintes de qualité visuelle.

Textures planes et objets *à peu près plats*

Dans leur article, Bier et Sloan partent du principe que l'on peut plaquer la texture sur n'importe quel objet. Or dans le cadre d'une utilisation "raisonnable" du placage de texture ce n'est pas vrai. En effet la texture est une image plane. Dans la pratique on plaque cette surface plane sur une surface de même nature, c'est à dire une surface plane.

Observons ce qui se passe pour les facettes. Considérons un objet composé d'un ensemble de facettes. Supposons que l'objet correspond à un objet réel sur lequel on trouve un certain nombre de dessins à restituer avec une technique de placage de textures planes. Pour une facette donnée, on suppose implicitement qu'elle est parallèle au plan de la texture (cette texture correspond à une photo). Le placage est donc parfait. Pour une autre facette, si on utilise le même plan de texture (ie la même photo) des distorsions apparaissent car la facette fait un angle avec le plan de texture. Dans ce cas on utilise une autre photo (ie une autre texture) prise sous un angle différent, de façon à ce que le nouveau plan de texture soit parallèle à la facette.

Si la surface n'est pas plane, on peut néanmoins envisager de plaquer la texture sous certaines conditions. Rappelons que notre texture est une image plane. La plaquer sur un objet représenté sous forme mathématique est équivalent à essayer de coller du papier peint sur l'objet réel correspondant. On peut plaquer aisément l'image sur un objet plat (poser du papier peint sur un mur c'est raisonnable) mais aussi sur un objet *à peu près plat*. Dans ce dernier cas, il n'y a qu'une faible distorsion visuellement acceptable.

Nous définissons un objet à peu près plat comme un objet sur lequel une texture plane peut être plaquée avec une distorsion suffisamment faible pour que le résultat visuel soit acceptable. Par opposition tout objet qui n'entre pas dans cette catégorie est considéré comme sphéroïde. Un objet sphéroïde est un objet dont la forme se rapproche de celle d'une sphère. Par exemple la sphère elle-même mais aussi le cube ou le cylindre.

Objets *sphéroïdes*

On se rend vite compte que bien que cela soit faisable mathématiquement[1], il n'est pas raisonnable de plaquer une image plane sur un objet sphéroïde (on ne pose pas du papier peint sur une voûte en ogive). En effet il y a au moins une discontinuité et une forte distorsion.

Dans certains cas, si l'on veut des effets visuels particuliers, on peut tout de même plaquer une texture plane sur un objet sphéroïde. Mathématiquement cela est possible si l'on a une paramétrisation de l'objet. On a alors beaucoup de déformations (distorsion et éventuellement discontinuité). Mais souvent ces déformations sont effectivement recherchée. Toutefois ces cas particuliers sont rares et sortent du cadre du rendu temps-réel. Ils sont gérés par des moyens logiciels.

Choix de la surface intermédiaire

Puisque l'on a deux types d'objets, il est pertinent de choisir deux surfaces intermédiaires différentes. En effet, tant que l'objet est à peu près plat on peu donc utiliser une surface intermédiaire plane. Quand l'objet n'est plus assez plat (ie il se rapproche d'une forme sphéroïde) la distorsion augmente et les défauts visuels sont apparents. La limite entre les

1. En tout cas pour les surfaces paramétrables et éventuellement avec de fortes distorsions

objet à peu près plats et les objets sphéroïdes est forcément floue dans la mesure où elle dépend de la qualité visuelle qui est une notion subjective. Néanmoins lorsque l'on juge que l'objet n'est plus assez plat il faut changer de surface intermédiaire.

Trois alternatives sont envisageables, le cube, la sphère et le cylindre. On décide d'utiliser le cube. Ce choix est étroitement lié au choix de la méthode de placage. Le cube permet de simplifier considérablement l'algorithme de sélection d'un texel.

Choix de la méthode de placage

Une première idée est d'utiliser la normale à l'objet puisque nous la calculons déjà. Mais cette solution comporte deux inconvénients. Tout d'abord cela suppose le calcul de l'intersection d'une droite (la droite qui passe par le point de l'objet considéré et ayant pour vecteur directeur la normale au point) avec la surface intermédiaire. De plus les quadriques concaves "inversent" la texture.

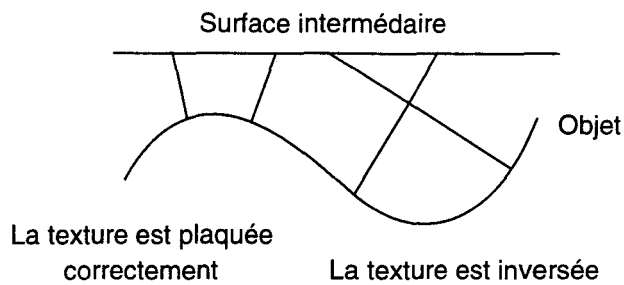


Figure 3.17 ***Inversion de la texture dans les concavités (d'après [Bier86])***

La solution est d'utiliser la normale à la surface intermédiaire. Pour le plan et le cube la normale est constante pour une face donnée. De plus, si l'on travaille dans le repère de la texture, cette normale est parallèle à l'un des axes principaux ce qui simplifie considérablement les calculs. Grâce à l'indexation de la texture par le biais de la normale à la surface intermédiaire, notre algorithme ne dépend pas de la primitive de visualisation et donc ne dépend pas de la quadrique. En fait on travaille uniquement sur des points.

Respect des contraintes visuelles

Préoccupons nous maintenant des problèmes de distorsion et de continuité. Pour les objets à peu près plats la distorsion existe mais elle est faible (d'autant plus faible que l'objet est plat). Il n'y a pas de discontinuité.

Pour les objets sphéroïdes la distorsion induite par le placage de texture est faible (Bier et Sloan remarquent que c'est même la meilleure solution de ce point de vue). Toutefois il y a des discontinuités. Mais celles-ci ne sont gênantes visuellement que sur certains objets et selon le type de texture. Pour un motif irrégulier les discontinuités "se perdent" dans les irrégularités (cf. images en annexe 6.4). Dans le cas d'une image (dessin ou photo) ou un motif régulier, les discontinuités sont visibles.Pour les éviter, la seule solution est de découper l'objet en plusieurs morceaux à peu près plats.

Respect des contraintes matérielles

Le choix des surfaces intermédiaires répond également aux contraintes matérielles. En effet une texture plane ou cubique se stocke facilement en mémoire (plus facilement que pour une sphère par exemple). De plus le plan de texture est dans la pratique limité à un carré et le cube est en fait composé de six carrés.

Comme nous allons le montrer ensuite, on peut utiliser le même algorithme (quelque peu adapté) pour déterminer le texel pour un plan ou un cube. Evidemment cela évite d'avoir à implanter matériellement deux algorithmes différents ce qui serait très coûteux.

On peut éventuellement utiliser une surface intermédiaire qui serait un cube privé de certaines de ces faces pour mieux correspondre à certains objets. Par exemple un cube dont on enlève la face supérieure et la face inférieure pour correspondre à un objet cylindrique.

Le cylindre et la sphère n'ont pas été retenus comme surfaces intermédiaires car leurs normales ne sont pas constantes et donc auraient imposé un calcul supplémentaire. De plus pour la sphère le stockage de la texture en mémoire n'est pas pratique.

Calcul du texel

Dans la mesure où le plan est un cas particulier du cube nous expliquons la méthode pour le cube (Figure 3.18).

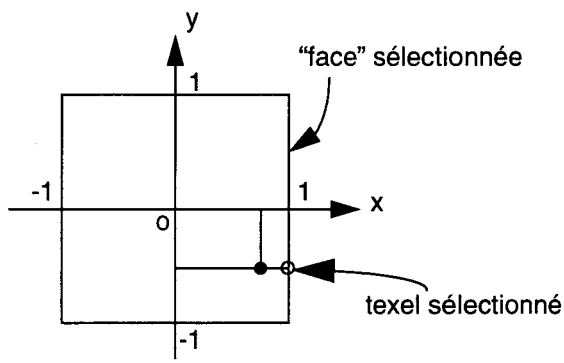


Figure 3.18 ***Détermination de la face et des coordonnées du texel dans la face (sur un exemple 2D)***

Il faut tout d'abord choisir la face dans laquelle on va aller chercher le texel. Pour cela on suppose que le point considéré est exprimé dans le repère de la texture. La face sélectionnée est celle dont le point est le plus proche. Comme nous somme dans le repère de texture on peut utiliser la distance de Manhattan. La valeur absolue et le signe de la coordonnées de plus grande valeur absolue détermine la face (notée f). Les valeurs de u, v sont alors simplement données par les deux autres coordonnées du point.

L'algorithme de détermination de la face et du texel dans la face est donné Figure 3.19. x, y et z sont les coordonnées cartésiennes du point dans le repère texture.

Pour le plan, l'algorithme reste valable dans son principe mais il est encore plus simple. La face étant fixée (par exemple la face 6) les coordonnées u, v sont immédiatement déterminées (pour l'exemple elles sont respectivement égales à x et y).

```
SI ( (|x|>|y|) ET (|x|>|z|) ) ALORS
        SI (x>0) ALORS f=1 SINON f=2
        u=y
        v=z

SI ( (|y|>|x|) ET (|y|>|z|) ) ALORS
        SI (y>0) ALORS f=3 SINON f=4
        u=x
        v=z

SI ( (|z|>|x|) ET (|z|>|y|) ) ALORS
        SI (z>0) ALORS f=5 SINON f=6
        u=x
        v=y
```

Figure 3.19 ***Algorithme de sélection de la face et du texel***

Le passage du point du repère de l'écran vers le repère de texture peut se faire en trois étapes (cf. Figure 3.20). La matrice de passage du repère écran au repère monde est déterminée lors de la phase de préparation du pipeline de rendu. La matrice de passage du repère monde au repère objet est donnée dans la description de la scène. La matrice de passage du repère objet au repère texture doit également être donnée dans la description de la scène.

Cette dernière matrice est facilement connue lors de la phase de modélisation. Pour un objet plat, on le modélise dans son propre repère puis on vient le mettre "en face" de sa texture de façon interactive ou non. De cette opération on déduit facilement la matrice recherchée. Celle-ci peut éventuellement être concaténée à la matrice de passage du repère objet au repère monde si les objets sont par ailleurs décrits directement dans le repère monde. On réduit ainsi le nombre d'informations à stocker dans la scène.

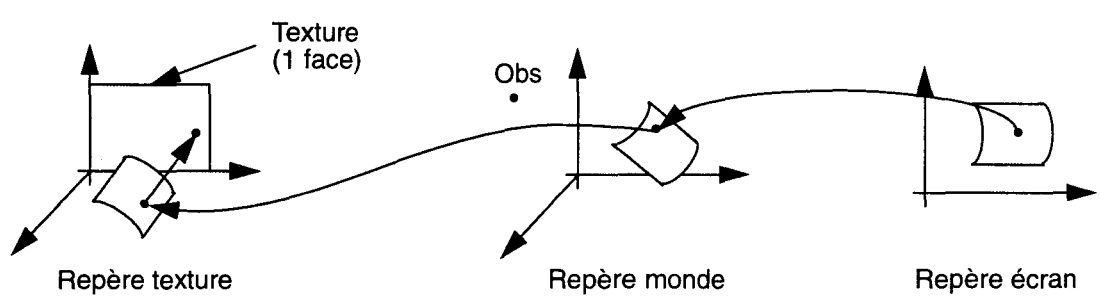


Figure 3.20 ***Passage du repère écran au repère texture (le passage du repère monde au repère objet a été regroupé avec le passage du repère objet au repère texture)***

Grâce aux coordonnées homogènes on peut exprimer les coordonnées (x_t, y_t, z_t, w_t) du point considéré dans le repère de texture en fonction de ses coordonnées $x_e, y_e, z_e, 1$ dans le repère écran. On obtient :

$$\begin{aligned} x_t &= a_1 x_e + b_1 y_e + c_1 z_e + d_1 \\ y_t &= a_2 x_e + b_2 y_e + c_2 z_e + d_2 \\ z_t &= a_3 x_e + b_3 y_e + c_3 z_e + d_3 \\ w_t &= a_4 x_e + b_4 y_e + c_4 z_e + d_4 \end{aligned} \qquad \textbf{(Eq. 3.17)}$$

Pour repasser en coordonnées cartésiennes les trois premières coordonnées sont divisées par la dernière.

3.5.3 L'anti-aliassage de texture

Le phénomène d'aliassage sur les textures se perçoit principalement sous forme de *moiré*. Il est dû à un sous échantillonnage de la texture qui apparaît lorsque la texture est compressée. Sur la Figure 3.21 ce phénomène est présenté pour un exemple mono-dimensionnel. Pour une image on pourra avoir un *taux de compression* selon deux axes.

Pour lutter contre l'aliassage des textures, on utilise les deux grandes techniques présentées au 3.4.1 pour lutter contre le crénelage : le pré-filtrage et le sur-échantillonnage.

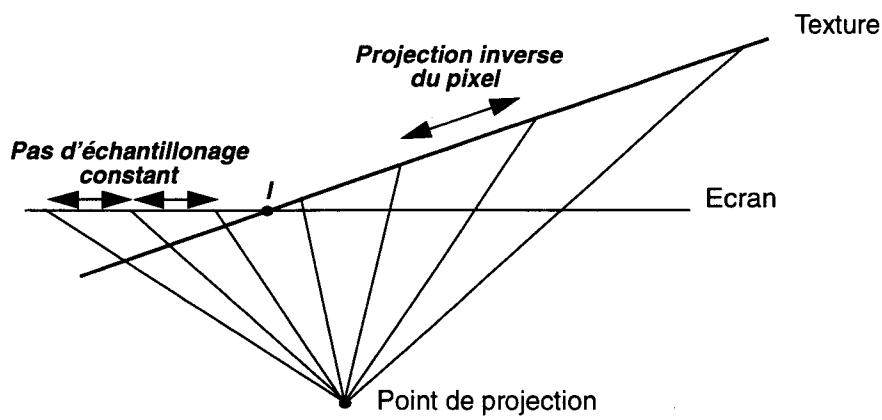


Figure 3.21 ***Compression (à droite du point I) et dilatation (à gauche du point I) d'une texture (d'après [Preux95])***

Le pré-filtrage

On considère le pixel non pas comme un point mais comme une surface rectangulaire. Sa projection inverse sur la texture est un quadrilatère. En théorie la couleur du pixel est donc obtenue en intégrant la couleur de tous les texels du quadrilatères. Toutefois ce calcul est particulièrement onéreux et dépend du taux de compression. Pour simplifier le calcul on effectue un filtrage a priori de la texture en fonction du taux de compression.

La méthode qui fait référence dans le domaine est celle du *mip-mapping* (Figure 3.22) [Williams83]. Le principe consiste à utiliser plusieurs tables de texture à différentes définitions, les plus petites étant obtenues en moyennant les plus grandes. Lors du rendu, le taux de compression est déterminé et l'on obtient la couleur du pixel en interpolant les valeurs extraites des table qui correspondent aux taux immédiatement supérieur et immédiatement inférieur.

Cette méthode offre une très bonne qualité pour un surcoût raisonnable. D'ailleurs elle a été implémenter matériellement dans la *Reality Engine* [SGI92]. Toutefois elle souffre du fait que le filtrage est symétrique. Pour palier cet inconvénient, différentes méthodes ont été proposées. La plus connue [Crow84] tient son nom du codage de la texture dans une *table d'aire sommée (Summed Area Table)*. Elle offre le double avantage de coder tous les taux de compression (on n'a donc plus à interpoler entre deux tables) et de prendre en compte séparément les taux de compression selon les deux axes de la texture.

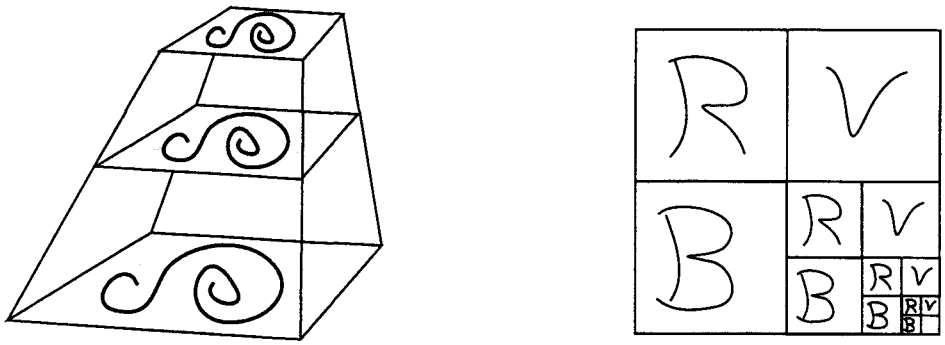


Figure 3.22 ***Mip-mapping : pyramide des tables de texture et représentation mémoire***

Le sur-échantillonnage

Les techniques de sur-échantillonnages sont beaucoup plus systématiques que les techniques de pré-filtrages. Elles fonctionnent aussi bien pour lutter contre le crénelage que pour éviter les moirés dans les textures. On peut donc utiliser les méthodes présentées au paragraphe 3.4.1. pour anti-alliasser les textures.

L'adaptation à la quadrique

La méthode de placage de texture que nous avons proposée au paragraphe 3.5.1 offre un avantage majeur : elle ne dépend pas de la primitive de visualisation. On ne travaille que sur des points. C'est un gros avantage car on peut utiliser n'importe quelle technique classique de lutte contre l'aliassage des textures. On pourra donc utiliser les méthodes de pré-filtrage aussi bien que les méthodes de sur-échantillonnage.

3.6 Bilan

Le but de ce chapitre est la conception algorithmique de notre système de visualisation temps-réel basé sur la quadrique. Avant de conclure sur les études réalisées, nous présentons un récapitulatif des résultats obtenus :

- La définition de notre objet de base est la première étape indispensable à la conception d'un système de visualisation. Notre objet est à la fois simple et suffisamment général pour correspondre à n'importe quelle primitive quadrique fournie par la modélisation.
- Le choix du pipeline de Phong est imposé par la quadrique. En effet celle-ci ne convient pas au pipeline de Gouraud, usuellement utilisé, car il lui impose ses limites. L'emploi du pipeline de Phong est à la fois un avantage et un inconvénient : avantage car il permet un éclairement de meilleur qualité, ce qui va dans le sens de notre recherche de qualité visuelle ; inconvénient car il nécessite une phase de post-éclairement qui augmente le coût général d'un accélérateur graphique.
- Les algorithmes de rendu de base ont été adaptés à la primitive de visualisation.
- L'amélioration de la qualité comprend l'anti-aliassage, la transparence et les ombres portées. Une études des techniques classiques montre que l'on peut facilement les adapter à la quadrique.
- Le placage de texture sur la quadrique a demandé une étude plus approfondie et nous a conduit à des modifications importantes. Nous proposons une méthode de placage originale

qui prend en compte à la fois nos contraintes de temps réel et les spécificités de l'objet de base. De plus cette méthode ne complique pas les algorithmes d'anti-aliassage.

En résumé nous avons montré qu'il est possible de proposer un système de visualisation temps réel basé sur la quadrique. Nous avons choisi l'organisation générale, nous avons clairement défini la primitive de visualisation et nous avons proposé des algorithmes permettant de prendre en compte toutes les caractéristiques des accélérateurs graphiques basés sur la facette (en particulier le placage de texture).

Néanmoins notre étude doit être complétée afin de s'assurer que l'on peut effectivement réaliser un accélérateur. Pour cela une double mise en oeuvre est indispensable. Une mise en oeuvre logicielle permet de valider les algorithmes et d'étudier le système complet. Une mise en oeuvre matérielle consiste à s'assurer que les algorithmes peuvent être implantés matériellement et à évaluer le coût de cette implantation. L'étude d'un système complet basé sur le pipeline de Phong doit ensuite être réalisée. Ce travail fait l'objet du chapitre quatre.

Mise en oeuvre

Dans le chapitre précédent nous avons défini tous les algorithmes nécessaires à la réalisation d'un système de visualisation basé sur la quadrique. Nous avons particulièrement insisté sur les parties originales par rapport à un système classique.

Toutefois trois grandes étapes restent à franchir pour arriver à une réalisation effective de l'accélérateur :

- La conception d'un processeur de rendu d'objet quadrique. Il s'agit d'une part de l'implémentation matérielle des algorithmes proposés et d'autre part de l'estimation de son coût.
- L'étude architecturale d'un système complet intégrant le processeur conçu dans la phase précédente. Les principaux points sont le parallélisme, la communication entre les différents modules et la mémoire.
- Le logiciel de simulation et de validation. Il permet de vérifier que les algorithmes proposés au chapitre précédent fonctionnent bien et que l'implémentation matérielle qui en est faite est correcte.

4.1 Le processeur quadrique

Le processeur quadrique est le coeur de notre système. Il assure le rendu de notre objet de base tel que défini au paragraphe 3.1. Il réalise la conversion objet-pixels et le calcul des différents paramètres en chaque pixel (profondeur, normale, coordonnées du point dans le repère monde et coordonnées de texture). De plus il effectue l'élimination des parties cachées et éventuellement des traitements plus compliqués comme la prise en compte d'arbres CSG et la transparence.

Nous faisons une analyse descendante du processeur quadrique. Ainsi il est décomposé en modules fonctionnels qui sont décrits à partir d'éléments de base. Cette étude nous permet d'évaluer le coût du processeur (en nombre de transistors) dans le paragraphe 4.2.

4.1.1 Présentation générale

Le processeur quadrique s'organise en six modules distincts. Le premier détermine la présence de l'objet de base en chaque pixel et calcule les deux profondeurs de la primitive quadrique ainsi que la profondeur de chaque plan. Le deuxième sélectionne la bonne profondeur et fournit les informations pour le calcul et la sélection de la bonne normale (cf. paragraphe 3.3.3). Le troisième calcule la normale à la primitive quadrique et aux plans. Le quatrième calcule les coordonnées du point dans le repère monde. Le cinquième calcule les coordonnées de texture. Enfin le sixième effectue les traitements inter-objets, typiquement le z-buffer.

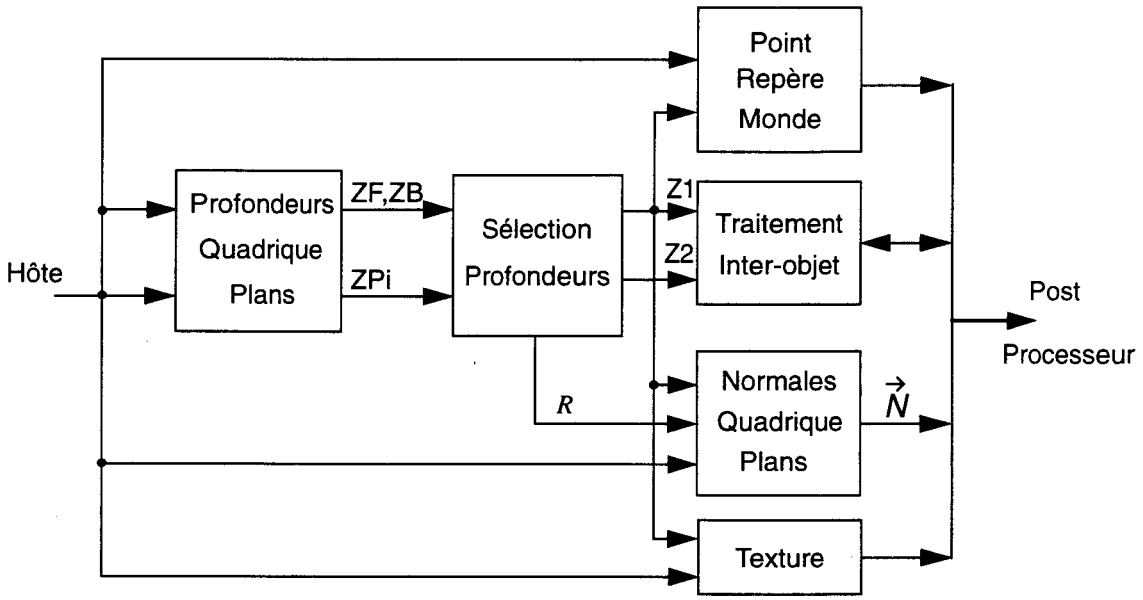


Figure 4.1 ***Le processeur quadrique : schéma général***

4.1.2 Les éléments de base

Tous les éléments tels que additionneurs, comparateurs, registres sont considérés comme triviaux et ne sont pas détaillés. Toutefois certains éléments un peu plus évolués nous sont utiles dans différents modules. Nous décrivons leur rôle et leur structure.

Le PE1

Cet élément, appelé également interpolateur linéaire, permet de calculer de façon incrémentale des expressions du premier degré en x, y. Son fonctionnement est basé sur la méthode des différences finies[1] [Chang89].

Le principe en est le suivant : soit l'expression $R = Ax + By + C$, x et y étant les coordonnées d'un pixel. On veut calculer la valeur de cette expression pour tous les pixels dans une zone rectangulaire de l'écran commençant en x_0, y_0. On initialise le registre R du PE1 de la façon suivante : $R = Ax_0 + By_0 + C$. Cette valeur est également stockée dans un mot mémoire noté M_1. A chaque fois que l'on se déplace d' un pixel sur une ligne on effectue $R = R + A$. A chaque fois que l'on se déplace d'une ligne on recharge M_1 dans R puis l'on effectue $R = R + B$.

1. en anglais forward differencing

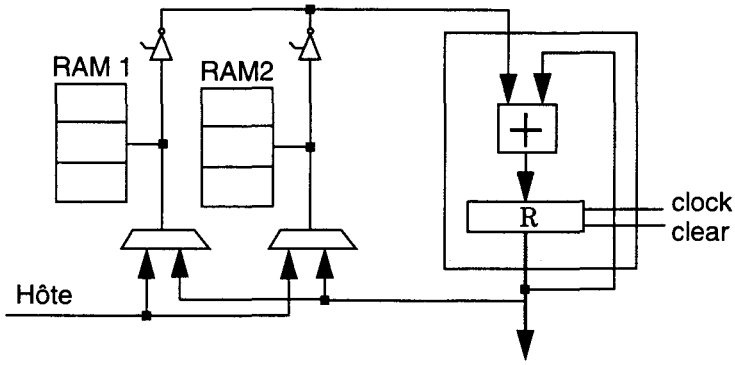


Figure 4.2 ***Schéma du PE1***

Pour l'interpolateur proprement dit un additionneur 24 bits et un registre 24 bits suffisent. Pour stocker les coefficients, deux RAMs sont utilisées. La première fournit les coefficients pour le calcul en cours. Pendant ce temps la deuxième est chargée (en l'occurrence depuis le hôte). A la fin d'un calcul (ie on change d'objet) le rôle des deux mémoires est inversé. Pour le PE1 une mémoire contient trois mots dans lesquels on range A, B et $Ax_0 + By_0 + C$.

Le PE2

Le PE2 ou interpolateur quadratique calcule des expressions du second degré en x, y, de la forme $Q(x, y) = Ax^2 + (Cy + D)x + (By^2 + Ey + F)$. On a donc une expression du second degré en x dont les coefficients sont des expressions de degré zéro, un ou deux en y. A chaque début de ligne il faut donc calculer $L_1 = Cy + D$ et $L_2 = By^2 + Ey + F$. En chaque pixel il faut calculer $Q(x) = Ax^2 + L_1 y + L_2$. Il nous reste à montrer comment calculer une expression du second degré en une seule variable avec un PE2, par exemple $Q(x) \,.\, Q(x+1) = Q(x) + 2Ax + L_1 + L_2$. Si l'on pose $R(x) = 2Ax + L_2 - A$ et $R(x+1) = R(x) + 2A$ alors $Q(x+1) = Q(x) + R(x+1)$. Le PE2 calcule R dans son premier étage et Q au second étage.

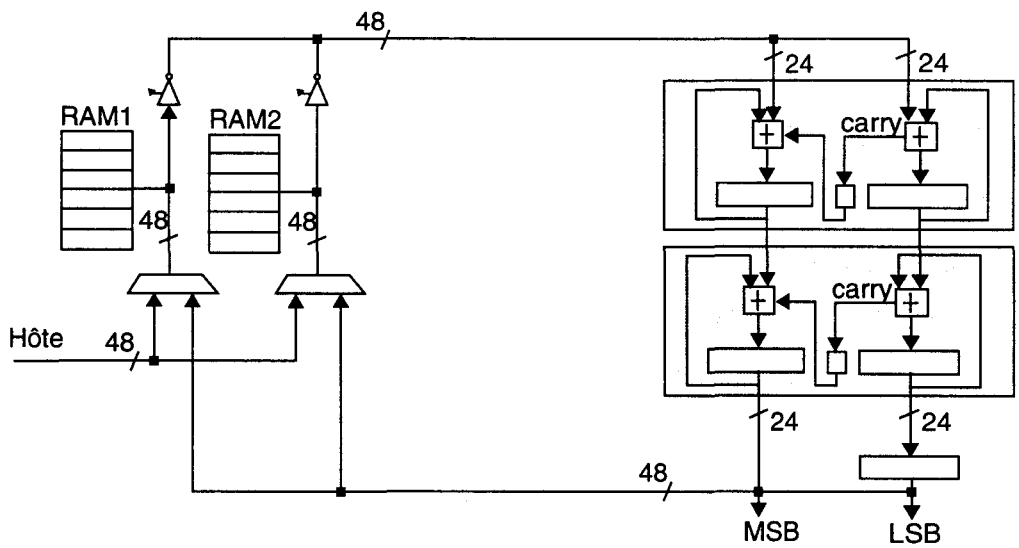


Figure 4.3 ***Schéma d'un PE2 pipeliné***

Le PE2 peut être considéré comme deux PE1 cascadés. Ces deux PE1 ont alors une largeur de donnée double (48 bits). Pour le PE2 il faut deux mémoires de six mots. Pour des raisons de vitesse les additionneurs 48 bits sont pipelinés. Un registre de synchronisation est donc ajouté en sortie.

L'extracteur de racine carrée

Le principe de l'extraction de racine carrée repose sur la technique de division euclidienne apprise à l'école primaire. On calcule donc une division pour laquelle le diviseur n'est pas explicitement connu. Toutefois on sait que le diviseur est également le résultat. La méthode est valable dans n'importe quelle base B [Cowgill64]. Nous la présentons pour des nombres compris entre zéro et un mais elle peut facilement être étendue à des nombres quelconques.

Soit S la racine carrée de N. Soit $p_1 p_2 ... p_n$ la suite de digits qui représente S, p_k étant le digit de poids B^{-k}. La méthode détermine de façon itérative les p_k. A l'étape k on a :

$$S_k = S_{k-1} + p_k B^{-k} \qquad \textbf{(Eq. 4.1)}$$

D'autre part on a :

$$N = S_k^2 + N_k \text{ et } N = S_{k-1}^2 + N_{k-1} \qquad \textbf{(Eq. 4.2)}$$

En élevant l'équation (Eq. 4.1) au carré (de plus afin d'alléger la notation on "oublie" B^{-k}) et en substituant S_k dans les équations (Eq. 4.2) on obtient :

$$N_k = N_{k-1} + p_k(2S_{k-1} + p_k) \qquad \textbf{(Eq. 4.3)}$$

Lorsque l'on effectue une division on cherche à minimiser N_k. Ici cela revient à trouver le plus grand p_k tel que $p_k(2S_{k-1} + p_k) \leq N_{k-1}$.

En base binaire p_k vaut soit zéro soit un. Il suffit donc de supposer $p_k = 1$ et calculer (Eq. 4.3). Si le résultat est positif on a fait la bonne supposition et on peut passer à l'étape suivante. Sinon on s'est trompé, p_k vaut zéro et $N_k = N_{k-1}$.

Cette méthode peut s'implémenter matériellement [Majithia72] au moyen d'un réseau cellulaire. Une cellule est approximativement un additionneur et un multiplexeur.

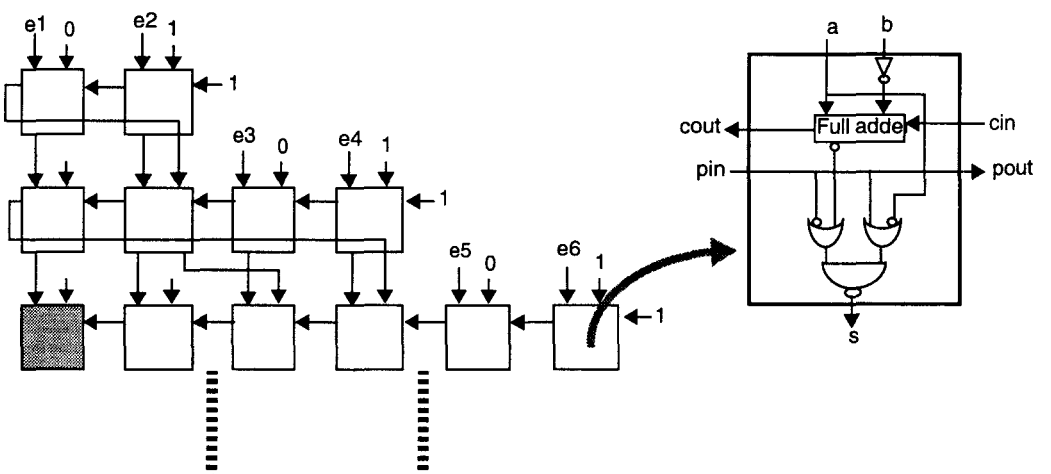


Figure 4.4 *L'extracteur de racine carrée : le réseau et la cellule de base*

On peut réaliser deux optimisations. Tout d'abord la simplification des cellules pour lesquelles une ou deux entrées sont figées. Ensuite on peut supprimer un certain nombre de cellules en remarquant que le reste est borné. En effet $N_k \leq (S_k + 1)^2 - (S_k)^2 - 1$ d'où $N_k \leq 2S_k$. N_k a donc au maximum un bit de plus que S_k. Sur la *ki*ème ligne on a donc besoin de $k+2$ cellules (sauf la première). Ainsi sur la figure ci-dessus la cellule grisée est donc inutile.

Le multiplicateur

Il s'agit d'un simple multiplicateur parallèle. Toutefois comme il sert au calcul de la normale, nous ne nous intéressons qu'au seize bits de poids forts [Lefévère94]. Il nous faut donc une unité de normalisation des deux nombres en entrée afin d'avoir les bits significatifs. Il s'agit de décaler les nombres vers la gauche de façon à amener le premier un (celui de poids le plus fort) tout à gauche. On effectue la multiplication. Le résultat est tronqué pour ne conserver que les seize bits de poids fort. Ensuite il est décalé vers la droite autant de fois que l'on a décalé les nombres en entrée.

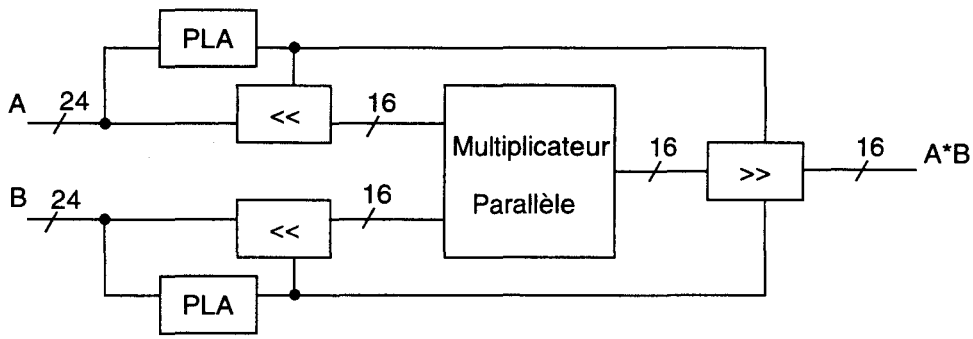


Figure 4.5 ***Le multiplicateur***

La détection du premier un se fait avec une petite PLA qui commande un décaleur variable. Le contrôle des décaleurs variables se fait sur cinq bits. En effet le nombre maximum de décalages est de vingt quatre ce qui se code sur cinq bits.

4.1.3 Les différents modules

Nous détaillons maintenant les six modules présentés Figure 4.1. Notons que le module de placage de texture est optionnel. En effet une machine bas de gamme n'offre pas de placage de texture. Ce caractère optionnel sera repris dans l'évaluation du coût du processeur au 4.2.3.

Calcul des profondeurs de la quadrique

Ce module est l'implémentation matérielle de l'équation (Eq. 3.5). De plus il fournit le signe de $f_2(x, y)$ qui sert pour la conversion objet-pixel. Son schéma bloc est présenté dans la figure ci-dessous. Remarquons que la deuxième profondeur est calculée simplement grâce à un additionneur supplémentaire.

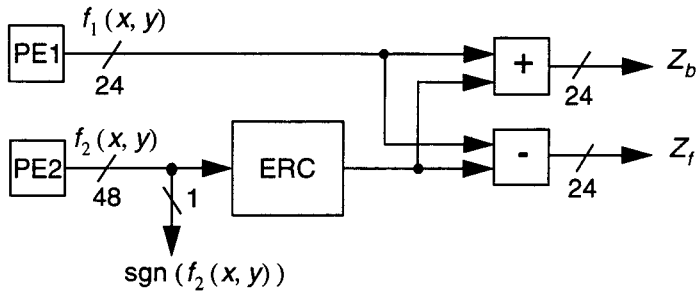


Figure 4.6 ***Calcul des profondeurs de la quadrique***

Le calcul de la profondeur d'un plan est immédiate à l'aide d'un PE1.

Sélection de la bonne profondeur

Nous proposons ici l'implémentation matérielle des algorithmes décrits au 3.3.3. Nous disposons de trois blocs fonctionnels qui correspondent aux trois étapes de l'algorithme. Pour simplifier le schéma nous ne spécifions que les chemins de données pour la profondeur mais il y a également les bits de visibilité et les numéros de primitives.

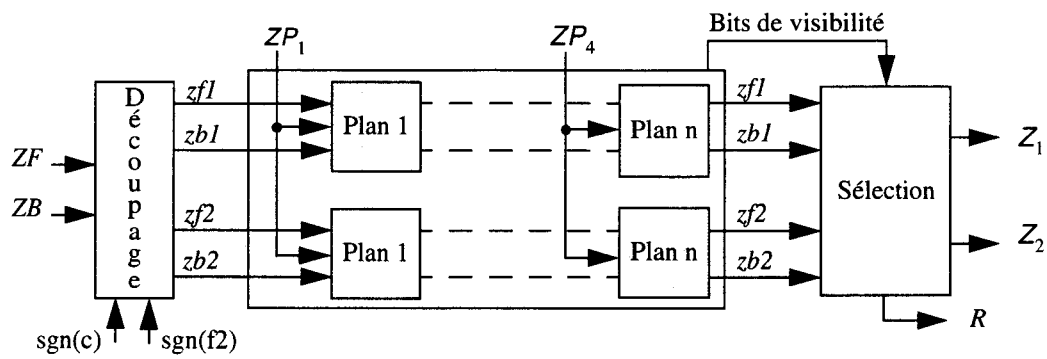


Figure 4.7 ***Sélection de la bonne profondeur : structure générale***

La phase de découpage nécessite simplement quatre multiplexeurs (un pour chacune des quatre profondeurs) contrôlés par le signe de c et le signe de $f_{2n}(x, y)$.

Pour la phase de construction itérative de l'objet, chaque "carré grisé" correspond à l'implémentation de l'algorithme de la Figure 3.10 (pour le plan i et un demi objet). Il s'agit de sélectionner une profondeur selon cinq critères : les deux premiers concernent l'orientation du plan ($\text{sgn}(CP_i)$ et $CP_i = 0$?) ; les trois suivant indiquent la position relative du plan par rapport au demi objet ($\text{sgn}(zf - ZP_i)$, $\text{sgn}(zb - ZP_i)$ et $\text{sgn}(ZP_i)$. On a donc pour chaque triplet profondeur-visibilité-numéro-de-primitive un multiplexeur commandé par une combinaison booléenne simple de ces cinq bits.

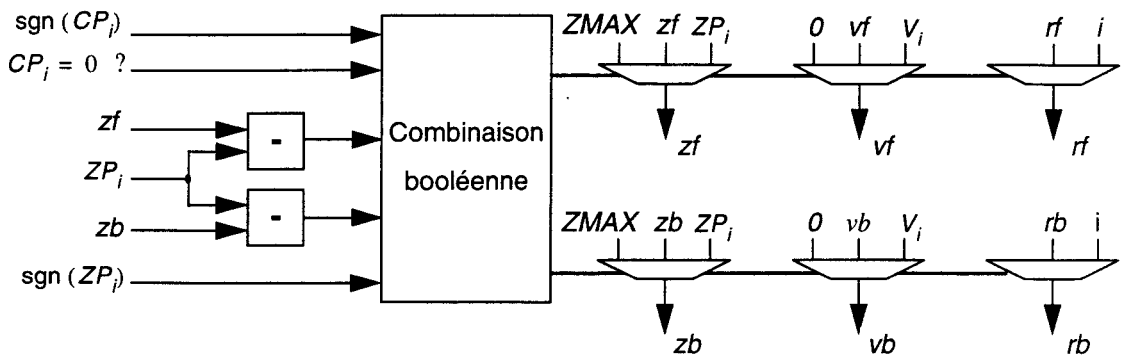


Figure 4.8 ***Construction itérative de l'objet : un plan i coupe un demi-objet***

La phase de sélection est implémentée avec un multiplexeur pour Z_1, un pour Z_2 et un pour R qui sont contrôlés par les quatre bits de visibilité.

Calcul des normales aux plans et à la quadrique

Tout d'abord il faut calculer la normale à la quadrique. Il nous faut par composante un PE1 un multiplicateur et un additionneur. Le PE1 calcule la partie linéaire en x, y. Le multiplicateur multiplie Z_1 avec le bon coefficient. On additionne alors le résultat des deux calculs précédents. Si Z_1 n'est pas une des deux profondeurs de la quadrique le calcul n'a aucun sens. Il est fait tout de même mais il sera ignoré au moment de la sélection de la bonne normale.

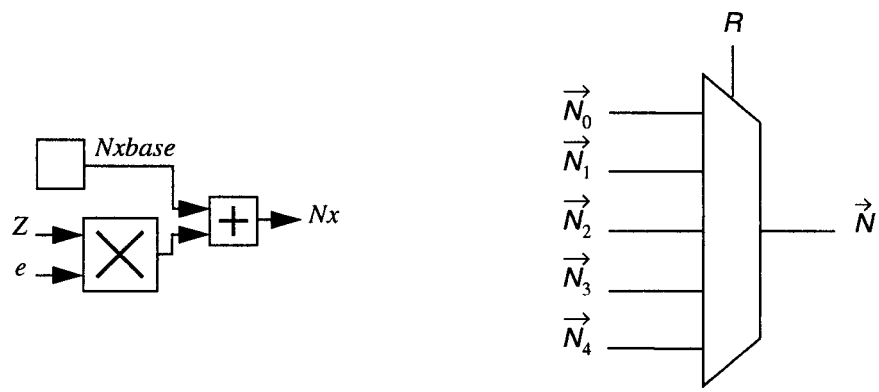


Figure 4.9 ***calcul de la normale : calcul d'une des composantes de la normales au plans (à gauche) et sélection de la bonne normale (à droite)***

Ensuite il faut sélectionner la bonne normale c'est à dire la normale qui correspond à la primitive visible. Son numéro est stockée dans R qui sert à contrôler le multiplexeur.

Si l'on veut interpoler la normale à un plan il faut prévoir trois PE1 par plan (un par composante de la normale). Eventuellement et afin de limiter le coût on peut limiter cette possibilité à un seul plan (ie le plan support de la facette).

Calcul des coordonnées du point dans le repère monde

On cherche les coordonnées du point dans le repère monde en fonction de ses coordonnées dans le repère écran. Les coordonnées homogènes sont des expressions linéaires en x_e, y_e, z_e.

Comme pour la normale on sépare chaque composante en deux parties. Le terme en z_e et le reste de l'expression. Le premier terme est calculé au moyen d'un multiplicateur et est ajouté au deuxième terme qui est interpolé au moyen d'un PE1.

Pour repasser en coordonnées cartésiennes il faut diviser les trois premières coordonnées homogènes par la quatrième. Les diviseurs peuvent être implémentés dans le processeur mais l'on peut également laisser le post-processeur prendre en charge les divisions afin de ne pas trop surcharger le processeur.

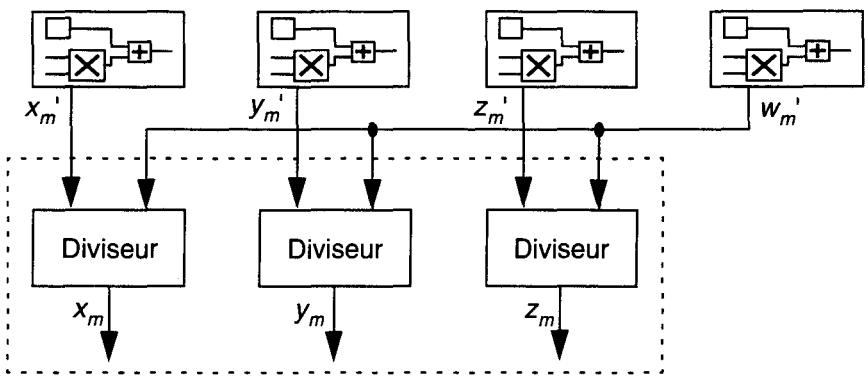


Figure 4.10 ***Calcul des coordonnées cartésiennes du point dans l'espace texture. Les diviseurs (encadrés en pointillés) peuvent être laisser à la charge du post-processeur.***

Calcul des coordonnées de texture

Ce calcul se décompose en deux étapes. Tout d'abord il faut exprimer le point dans le repère de la texture. Ensuite à partir des coordonnées de ce point on détermine les coordonnées du texel, à savoir la face et les coordonnées u, v dans la face.

Le calcul du point exprimé dans le repère de texture s'effectue à partir de l'équation (Eq. 3.17). On a donc quatre expressions linéaires en x_e, y_e, z_e. Ce calcul est tout à fait similaire au calcul du point dans le repère monde.

Le point obtenu est exprimé en coordonnées homogènes. Pour déterminer les coordonnées u, v il faut repasser en coordonnées cartésiennes. Pour cela il faut diviser les trois premières coordonnées homogènes par la quatrième.

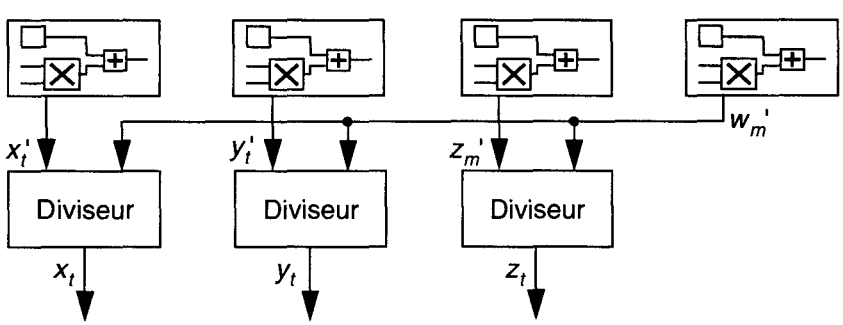


Figure 4.11 ***Calcul des coordonnées cartésiennes du point dans l'espace texture***

Les coordonnées du texel sont déterminées à partir des coordonnées cartésiennes du point dans l'espace texture en utilisant l'algorithme donné dans la Figure 3.19. Pour les comparaisons il faut trois soustracteurs. Ensuite pour sélectionner la face f et les coordonnées u, v il faut trois multiplexeurs. Ces multiplexeurs sont commandés par une combinaison booléenne simple des sorties des comparateurs.

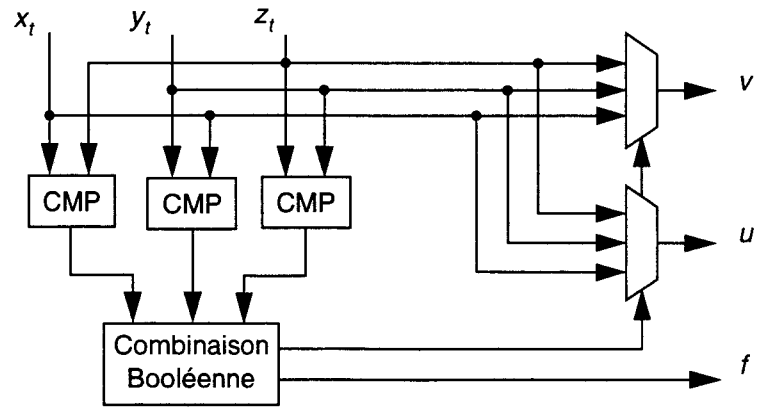


Figure 4.12 ***Détermination des coordonnées du texel***

Remarquons toutefois que l'on peut économiser une division. En effet les comparaisons peuvent être faites sur les coordonnées homogènes aussi bien que sur les coordonnées cartésiennes. Ensuite pour obtenir u et v on a besoin seulement de deux coordonnées cartésiennes. Il suffit donc de sélectionner les deux coordonnées homogènes qui nous intéresse d'abord et de les diviser ensuite. Remarquons que dans ce cas les tests de sélection doivent tenir compte du signe de w_t'. On obtient le schéma synoptique suivant :

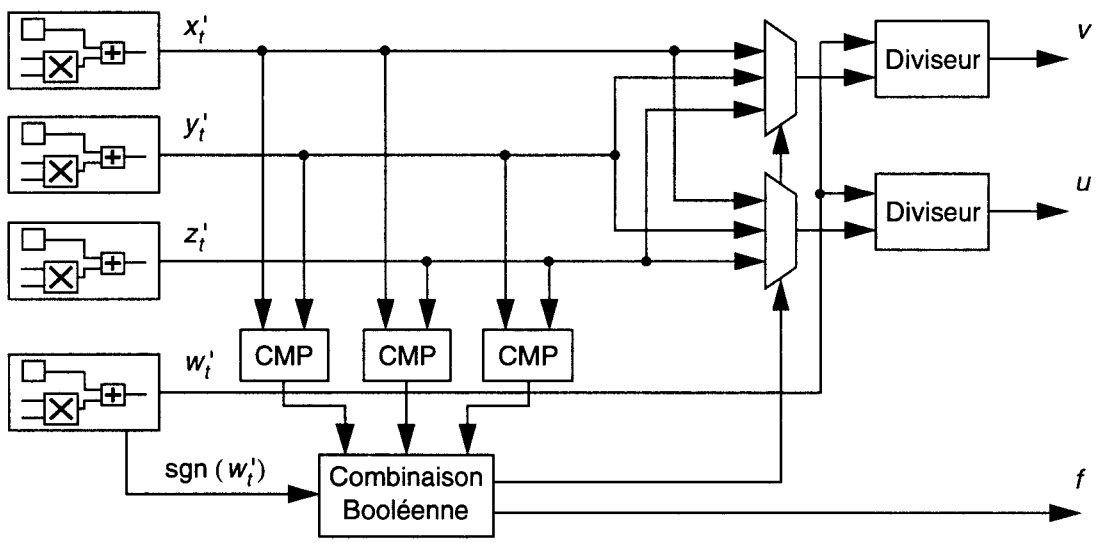


Figure 4.13 ***Calcul du texel : schéma final (optimisé)***

Traitement inter-objets

Revenons sur la dernière tâche du convertisseur, l'élimination des parties cachées. Comme tout accélérateur graphique nous utilisons un mécanisme de z-buffer qui ne dépend pas de la primitive de visualisation tant que l'on est en mesure de fournir pour celle-ci une profondeur en chaque pixel. L'algorithme du z-buffer s'implémente trivialement. Il suffit d'un comparateur et d'un accès mémoire en lecture et éventuellement écriture.

On peut généraliser ce travail à tous les traitements inter-objets effectués au niveau pixel. Ces traitements sont basés sur des comparaisons de profondeurs. C'est à ce niveau que le CSG et la transparence sont gérés.

CSG

Pour le rendu direct CSG on peut utiliser les algorithmes à z-buffer tels que celui implémenté sur la machine Pixel Planes-5 ou le trickle algorithm (cf paragraphe 2.3.2). Dans la mesure où ces algorithmes, à l'instar du z-buffer, travaillent sur des profondeurs, ils sont directement utilisables dans notre machine.

Remarquons d'une part, que lorsque l'on affiche directement des arbres CSG, l'utilisateur doit s'assurer que les objets de bases sont "convexes", c'est à dire qu'ils n'ont que deux profondeurs (une avant et une arrière).

D'autre part, pour savoir si une profondeur (ou plus exactement le point correspondant) est à l'intérieur ou non d'un objet on compare cette profondeur aux deux profondeurs avant et arrière de l'objet. Pour un objet positif (voir le paragraphe 2.3.2 pour l'explication de ce terme) on teste la profondeur avant Z1 et pour un objet négatif on teste la profondeur arrière Z2. On ne calcule qu'une seule normale et qu'un point dans le repère monde, celle et celui associés à la profondeur retenue.

Dans la mesure où la profondeur intervient dans ces calculs il faut un dispositif de sélection de Z1 ou Z2 selon que l'objet est positif ou négatif. Par ailleurs il faut prévoir de la mémoire supplémentaire en chaque pixel. Une profondeur et un bit de drapeau pour l'algorithme de Pixel Planes-5 et deux profondeurs pour le trickle.

Ombre portée

Après le rendu de la scène et l'élimination des parties cachées on rend les volumes d'ombre. Pour chacun d'eux et en chaque pixel ou il est présent on regarde si le point stocké dans le z-buffer est dans le volume ou non. Le résultat est un drapeau d'ombre pour chaque source, qui sera utilisé ensuite dans la formule d'éclairement : au départ l'éclairement de chaque point est la composante ambiante. Pour chaque source si le point est à l'ombre par rapport à cette source son éclairement n'est pas modifié sinon on ajoute la composante diffuse et spéculaire pour cette source.

Le calcul d'ombre portée n'exige que peu de matériel supplémentaire, seulement un bit de mémoire (par pixel) par source. En revanche il faut effectuer le rendu des volumes d'ombres ce qui exige une passe supplémentaire et dégrade les performances.

Transparence

Pour la transparence, on implémente l'algorithme de Mammen (cf paragraphe 3.4.2). Il faut notamment prévoir le stockage d'une profondeur supplémentaire.

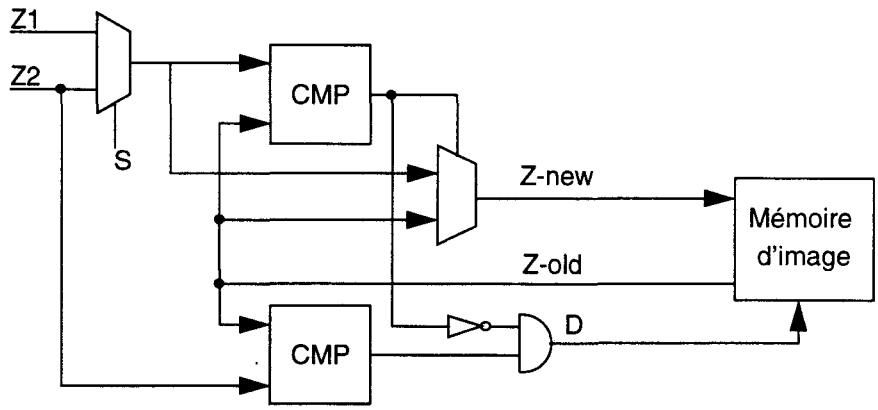


Figure 4.14 ***z-buffer, CSG et transparence (le bit S indique un objet positif ou négatif, D est un drapeau qui sert soit pour indiquer un objet à l'ombre soit pour le traitement CSG)***

Notons que toutes ces tâches ne nécessitent que très peu d'éléments calculatoires supplémentaires. En fait deux comparateurs et deux multiplexeurs suffisent. En revanche elles réclament beaucoup de mémoire supplémentaires. De plus elles nécessitent plusieurs passes et donc on perd le temps réel.

4.2 Evaluation du coût

L'évaluation du coût de notre processeur est particulièrement importante. En effet, le processeur quadrique est l'élément clé de notre système. Cette évaluation nous permet de montrer que le processeur quadrique est réalisable à un prix raisonnable. Nous pouvons alors étudié l'architecture globale de notre système.

Les estimations sont faites à partir de la bibliothèques ECPD15 du logiciel SOLO1400 de la firme European Silicon Structures [ESS89]. La largeur des chemins de données est supposée de 24 bits sauf précision contraire. Toutes les estimations sont données en nombre de transistors.

4.2.1 Les éléments de base

Nous allons tout d'abord donner le coût des éléments "triviaux" tels que les multiplexeurs et les additionneurs. Ensuite nous présentons les éléments de base qui sont le PE1, le PE2, l'extracteur de racine carrée et le multiplicateur.

Elément	Coût
Registre 24 bits	768
Additionneur 24 bits	864
Multiplexeurs 2 vers 1 (48 bits vers 24 bits)	336
Inverseur trois états 24 bits	192

Table 4.1: Eléments "triviaux"

les PE1 et PE2

Les éléments constitutifs d'un PE1 (respectivement d'un PE2) apparaissent clairement sur les Figure 4.2 et Figure 4.3. Leurs coûts sont récapitulés dans les deux tables suivantes. On note qu'un PE2 vaut environ quatre fois plus cher qu'un PE1, ce qui était prévisible.

Elément	Coût
1 registre	768
1 additionneur	864
2 multiplexeurs	672
2 inverseur trois états	384
RAM	1428
Total	4116

Table 4.2: Coût d'un PE1

Elément	Coût
5 registres	3840
4 additionneurs	3456
4 multiplexeurs	1344
4 inverseurs trois états	768
RAM	5712
Total	15120

Table 4.3: Coût d'un PE2

L'extracteur de racine carrée

Pour réaliser cet élément, il faut tout d'abord prendre en compte la partie calculatoire (Figure 4.4) puis il faut pipeliner le réseau.

Le réseau comprend 323 cellules. Chaque cellule nécessite 56 transistors.

Pour simplifier le pipeline nous décidons de placer un étage de registres entre chaque étage de calcul. A chaque étage il faut mémoriser la sortie des multiplexeurs, les bits du résultats déjà calculés et les bits du dividende qui n'ont pas encore été utilisés dans le calcul. Il faut donc mémoriser 49 bits par étage. Nous avons 24 étages. Le coût de l'extracteur est récapitulé dans la table ci-dessous.

Elément	Coût
323 cellules	18088
1176 registres 1 bit	30576
Total	48664

Table 4.4: Coût de l'extracteur de racine carrée

Le multiplicateur

Il comprend le multiplicateur parallèle qui est pipeliné, les PLAs et les décaleurs variables (Figure 4.5). Ceux-ci sont composés de cinq étages de multiplexage. Le coût d'un multiplicateur complet est résumé dans la table suivante :

Elément	Coût
1 multiplicateur parallèle pipeliné	15872
3 décaleurs variables	3360
2 PLAs	4000
Total	23232

Table 4.5: Coût d'un multiplicateur

4.2.2 Les modules

Calcul des profondeurs

Pour la quadrique les éléments constitutifs sont un PE1, un PE2, un extracteur de racine carrée et deux additionneurs (Figure 4.6). Pour calculer la profondeur d'un plan il suffit d'un PE1. Le coût de ce module se résume ainsi :

Elément	Coût
5 PE1	20580
1 PE2	15120
1 extracteur de racine carrée	48664
2 additionneurs	1728
Total	86092

Table 4.6: Coût du module de calcul des profondeurs

Sélection de profondeur

Nous évaluons le coût de ce module phase par phase (Figure 4.7), sans oublier de rajouter les registres de synchronisation.

Pour la phase de découpage il faut trois multiplexeurs (un quatrième est inutile car la profondeur arrière du deuxième demi-objet est toujours ZMAX). Les deux premiers sont des multiplexeurs un parmi trois. Le troisième est un multiplexeur un parmi deux.

Pour la construction itérative il faut par demi-objet et par plan : deux additionneurs, deux multiplexeurs un parmi trois (on néglige le coût des bits de visibilité et du numéro de primitive) et une combinaison booléenne (Figure 4.8).

Pour la sélection de la bonne profondeur il faut un multiplexeur un parmi cinq et un multiplexeur un parmi trois.

De plus on compte un étage de registres par plan pour le fonctionnement en pipeline. Il faut donc ajouter vingt registres.

La table ci-dessous regroupe ces évaluations.

Phase / élément	Coût
Découpage	1344
Construction itérative	22888
Sélection de la bonne profondeur	1344
20 registres	12480
Total	38056

Table 4.7: Coût du module de sélection de profondeur

Calcul des normales aux plans et à la quadrique

Pour les plans on a soit des constantes soit trois PE1 pour interpoler la normale au plan support. Pour chaque coordonnée de la normale à la quadrique on a besoin d'un PE1, d'un multiplicateur, d'un additionneur et d'un registre. Pour sélectionner la bonne normale il faut trois multiplexeurs un parmi cinq. Voici le récapitulatif :

Elément	Coût
6 PE1	24696
3 multiplicateurs	69696
3 additionneurs	2592
3 registres	2304
1 multiplexeur un parmi cinq	840
Total	100128

Table 4.8: Coût du module de calcul des normales

Calcul du point dans le repère monde

Pour l'évaluation de ce module nous ne prenons pas en compte les diviseurs[1]. On a besoin de quatre PE1, de quatre additionneurs, de quatre registres et de quatre multiplicateurs. Le coût est récapitulé dans la table ci-dessous :

Elément	Coût
4 PE1	16464
4 additionneurs	3456
4 registres	3072
4 multiplicateurs	92928
Total	115920

Table 4.9: Coût du module de calcul du point dans le repère monde

Calcul des coordonnées de texture

Ce calcul peut être découpé en trois blocs (Figure 4.13) : tout d'abord le calcul des coordonnées homogènes du point dans le repère de texture ; ensuite la sélection des coordonnées de texture ; enfin les divisions.

Pour calculer les coordonnées homogènes du point dans le repère de texture il faut quatre PE1, quatre additionneurs, quatre registres et quatre multiplicateurs. Pour la sélection il faut trois comparateurs (on considère que le coût d'un comparateur est celui d'un additionneur), deux multiplexeurs un parmi trois et une combinaison booléenne.

On a ensuite besoin d'un diviseur. Un diviseur peut être réalisé de la même façon que l'extracteur de racine carrée. En fait l'extracteur de racine carré est réalisé à partir d'un réseau qui effectue une division. On prendra donc le coût de l'extracteur pour celui d'un diviseur.

On peut récapituler le coût total du module de placage de texture:

Elément	Coût
4 PE1	16464
4 multiplicateurs	92928
4 registres	3072
7 additionneurs	6048
2 multiplexeurs un parmi trois	1008
2 diviseurs	97328
Total (sans les diviseurs)	119520
Total (avec les diviseurs)	216848

Table 4.10: Coût du module de placage de texture

1. Il faut alors passer quatre coordonnées (homogènes) et non trois (cartésiennes) au post-processeur

Notons que comme pour le module de calcul du point dans le repère monde on peut différer les divisions lors du post-éclairement. L'accès au texel se fera donc également à ce moment. Dans ce cas il faut transmettre au module de post-traitement une valeur de plus (en l'occurrence w_t. Le gain est très appréciable dans le cas où le module de post-traitement dispose déjà de diviseurs pour le calcul de l'éclairement de Phong.

Traitement inter-objets

Ce module très simple d'un point de vue calculatoire ne comprend que deux comparateurs et deux multiplexeurs (on néglige les deux portes). Le coût est seulement de 2400 transistors, ce qui est tout à fait négligeable par rapport aux autres modules.

Pipeline et contrôle

Chaque module pris séparément est conçu pour fonctionner selon un mode pipeline. Toutefois il faut prévoir de les synchroniser entre eux. En fait certains modules étant plus profonds que d'autres on les fait commencer avant. Les données utiles pour le fonctionnement des différents modules sont conservées dans une petite fifo externe au processeur.

Le module de contrôle doit gérer la fifo et contrôler les différents éléments des modules. On ne prévoit pas de mécanisme de rupture de séquence dans le pipeline. Lorsqu'une opération est invalide (par exemple on utilise une profondeur de plan pour calculer la normale à la quadrique) le résultat de cette opération est ignoré. Cette opération a été faite pour rien ou plus exactement pour éviter la rupture de séquence.

La logique de contrôle et les registres de synchronisation nécessitent 34020 transistors.

4.2.3 Le processeur quadrique

Pour connaître le prix de notre processeur quadrique nous faisons la synthèse des résultats précédents. Nous proposons deux versions, avec et sans texture. Les coûts de ces deux versions sont récapitulés dans la table ci-dessous :

Module	Coût
Calcul des profondeurs	86092
Sélection de profondeur	38056
Calcul de la normale	100128
Calcul du point dans le repère monde	115920
Calcul des coordonnées de texture	119520
Gestion du pipeline et contrôle	34020
Total (sans texture)	374216
Total (avec texture)	493736

Table 4.11: Coût du processeur quadrique avec et sans texture

Nous pouvons d'ores et déjà faire deux constatations. D'une part le processeur quadrique avec ou sans texture bien que relativement complexe est parfaitement réalisable. D'autre part, le

surcoût engendré par le placage de texture est important. Nous poursuivrons notre analyse au paragraphe 4.5.

4.3 Architecture de notre machine

Notre système ressemble fort à n'importe quel système basé sur le pipeline de Phong comme par exemple [Schneider89] ou [Molnar92]. Sa différence majeure se situe au niveau des processeurs de rendu que nous avons décrit aux paragraphes 4.1 et 4.2. Après un bref rappel sur l'architecture générale de notre système nous détaillons les trois parties principales, le processeur de préparation, le processeur de rendu et le post-processeur. Enfin nous récapitulons les besoins en mémoire et en vitesse de transfert entre chaque partie.

4.3.1 Architecture générale

La Figure 4.15 représente l'architecture matérielle de notre système de visualisation temps-réel basé sur la quadrique. Cette architecture correspond à l'implémentation matérielle usuelle du pipeline de Phong (cf Figure 3.1). Le hôte détient la base de donnée et en assure la traversée. Les processeurs de préparation et de rendu s'occupe respectivement de la préparation de la scène puis de son rendu. Le post-processeur réalise au minimum l'éclairement en chaque pixel.

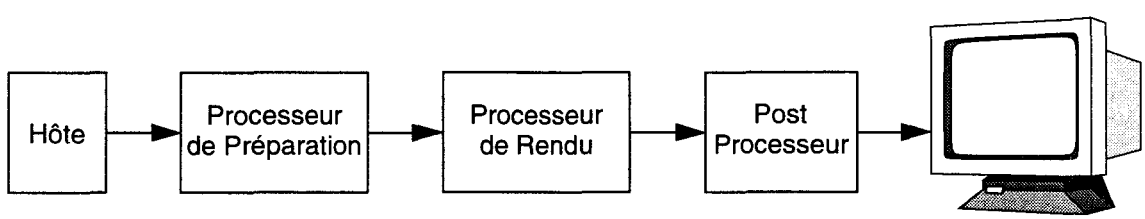


Figure 4.15 ***Schéma synoptique de notre système de visualisation***

Le travail du hôte est réduit à la gestion de la scène et ne sera pas détaillé. En revanche nous présentons les architectures envisageables pour les processeurs de préparation, de rendu et le post-processeur. Puis nous étudions également les problèmes de communication entre les différents modules. En effet les débits entre modules peuvent être considérables et sont même parfois le goulot d'étranglement d'un système. Enfin nous regardons les besoins en mémoire qui sont généralement énormes aussi bien en quantité qu'en vitesse d'accès. Les contraintes mémoires conditionnent fortement l'architecture des machines graphiques.

4.3.2 Le processeur de préparation

Remarquons tout d'abord que qualitativement le travail de préparation pour un système basé sur la quadrique est le même que pour un système basé sur la facette. En effet les tâches à réaliser sont approximativement les mêmes et donc les calculs sont de même nature. En rappelant brièvement le coût des différentes tâches à réaliser nous montrons que quantitativement la charge de travail est comparable. Nous concluons que l'architecture des systèmes actuels convient pour un système basé sur la quadrique.

Le premier processeur conçu spécialement pour la préparation est le célèbre *Geometry Engine* [Clark82]. Plusieurs processeurs de ce type étaient utilisés pour réaliser les transformations géométriques, le fenêtrage et le passage dans le repère de l'écran. Aujourd'hui encore on utilise plusieurs processeurs en parallèle. Nous présentons les architectures possibles.

Charge de travail

Le travail à réaliser lors de la préparation est présenté aux paragraphes 3.2.1 et 3.1.1 : traversée de la scène, transformations géométriques et transformation perspective, fenêtrage. Dans un système matériel il faut rajouter le calcul des coefficients nécessaires au rendu dans le convertisseur.

Pour un accélérateur classique la préparation exige 50 Mflops pour 100.000 facettes [Akeley88]. Pour la quadrique, Eric Nyiri a évalué le coût de préparation d'un patch quadrique [Nyiri94]. On note que la préparation d'un objet quelconque est identique à celle d'un patch. Le coût pour un objet est environ deux fois plus élevé, donc il faut compter 100 Mflops pour 100.000 objets quadriques.

En considérant le fait qu'une quadrique est une primitive de plus haut niveau que la facette et que donc elle remplace plusieurs facettes (jusqu'à plusieurs centaines dans le meilleurs des cas) la préparation d'une scène à partir d'objets quadriques est beaucoup moins coûteuse qu'avec des facettes.

Architecture

Les choix architecturaux pour le processeur de préparation ont évolué avec la technologie. Ce phénomène est parfaitement illustré par l'évolution des machines Silicon Graphics. La 4D/GTX utilise un pipeline, ce qui simplifie l'envoi des objets préparés au processeur de rendu. Plus on pipeline et plus on augmente les performances. Néanmoins cette technique atteint ses limites lorsque le découpage des tâches en sous tâches séquentielles n'est plus réalisable.

La machine 4D/VGX possède une architecture SIMD[1]. Quatre processeurs travaillent sur les quatre sommets d'un objet mais font tous la même chose au même moment. L'avantage de cette architecture est que l'unité de contrôle est unique pour les quatre processeurs. L'inconvénient est la gestions des tests. En effet si lors d'un test un processeur doit effectuer un certain calcul et que les autres n'ont pas à l'effectuer, le processeur occupé bloque les autres qui ne peuvent qu'attendre. Cet inconvénient limite le nombre de processeurs que l'on peut mettre en parallèle. En effet, lorsque ce nombre croît on a de plus en plus de chance pour qu'un processeur travaille et bloque les autres. Le gain de performance n'augmente alors plus.

Pour résoudre ce problème on est passé à une organisation MIMD[2] (comme par exemple sur la Reality Engine, cf [Akeley93]). Cette fois les processeurs en parallèle on chacun leur unité de contrôle et ne sont donc plus bloqués les uns par les autres. Bien sûr cet avantage se paye par une complexité du contrôle plus grande pour répartir les objets sur les différents processeurs.

Un autre paramètre intéressant est le choix de processeurs spécialisés ou à usage général. Il semble qu'il n'y ait pas de situation tranchée aujourd'hui. Il est manifeste que le choix des constructeurs dépend de la technologie dont ils disposent. On risque donc de continuer à voir une alternance entre ces deux solutions. La première offre généralement plus de puissance pour un coût de développement plus élevé. La seconde est plus facile à mettre en oeuvre mais est moins efficace.

4.3.3 Le processeur de rendu

Le processeur quadrique décrit au paragraphe 4.1 peut parfaitement constituer à lui seul le processeur de rendu de notre machine. Néanmoins pour atteindre des performances plus importantes il est possible d'en faire fonctionner plusieurs en parallèle.

1. Single Instruction Multiple Data
2. Multiple Instruction Multiple Data

Aujourd'hui les accélérateurs utilisent un parallélisme faible et cherchent à avoir une vitesse de calcul des pixels qui corresponde à la vitesse d'accès à la mémoire. Pour cela deux approches ont été implémentées.

La première approche se retrouve dans la ZX de chez SUN Microsystems [Deering93]. Elle consiste à avoir des processeurs relativement lents ce qui simplifie leur conception mais oblige à les dupliquer plus pour "suivre" la mémoire.

La deuxième approche consiste à avoir moins de processeurs mais qui vont plus vite. A technologie égale il faut pour cela augmenter le nombre d'étages du pipeline du processeur. La conception d'un processeur est donc légèrement plus complexe. De plus il faut éviter au maximum une rupture de séquence dans le pipeline ce qui est d'autant plus pénalisant que le pipeline est profond. C'est le choix pris pour l'Extreme de chez Silicon Graphics [Harell93].

Pour notre processeur quadrique nous devons tenir compte du fait qu'il a un pipeline très profond (une cinquantaine d'étages). Toutefois ce pipeline permet de travailler à une fréquence élevé (entre 50 et 100 MHz). L'architecture de l'Extreme est donc tout à fait indiquée pour servir de base à notre module de conversion.

Remarquons que ce type d'architecture offre aujourd'hui le meilleur rapport coût-performances.

4.3.4 Le post-processeur

Dans le pipeline de Phong l'éclairement est réalisé par un post-processeur. L'éclairement étant indispensable, la présence du post-processeur s'impose. De plus nous avons vu au paragraphe 3.1.2 qu'il faut une puissance de calcul importante pour soutenir le temps réel.

Nous rappelons quels sont les calculs à faire qui avaient été différés lors du rendu ainsi que les calculs à faire pour l'accès au texel. Puis nous décrivons les propositions d'implémentation matérielle pour répondre au problème d'éclairement.

Les divisions

Il faut effectuer les divisions que nous avons différées lors de la phase de rendu. Il en faut trois pour obtenir les coordonnées cartésiennes du point exprimées dans le repère monde à partir de ses coordonnées homogènes. Il en faut deux autres pour repasser le texels en coordonnées cartésiennes à partir de ces coordonnées homogènes.

L'accès au texel

Une fois que l'on a les coordonnées cartésiennes du texel un accès à la mémoire de texture est effectué. Si l'on anti-alliasse par pré-filtrage (technique du mip-mapping) il faut huit accès à la mémoire de texture plus une interpolation linéaire.

Eclairement

Rappelons que le coût de l'éclairement de Phong sur une image 1024×1024 en temps réel (25 images par seconde) requiert une puissance de calcul de 2,2 Gflops. Différentes architectures ont été proposées :

- La machine GSP-NVS [Deering88] : un pipeline de seize processeurs NVS (Normal Vector Shader) calcule la formule de Phong pour cinq sources directionnelles et un observateur à distance finie. Les opérateurs de racine carrée et de puissance sont réalisés grâce à des ROMs. Ce processeur n'a malheureusement pas été mené à son terme.
- La machine PROOF [Claussen91] : Le processeur de base permet de calculer pour une source la composante diffuse ou la composante spéculaire. Le nombre de processeur est le double du nombre de sources plus un pour la composante ambiante. L'éclairement est

calculé avec la formule de Blinn et l'observateur placé à l'infini. Pour des raisons d'économie la normale n'est pas normalisée. Comme dans GSP-NVS les opérateurs de racine carrée et de puissance sont stockés dans des tables.

- La machine IMOGENE [Lefévère91] [Lefévère92] : la solution retenue est de réduire au maximum la taille des données pour pouvoir utiliser le plus souvent possible des tables pour les opérateurs. Seules les additions sont effectivement calculées. Toutefois V. Lefévère reconnaît que cette limitation n'est acceptable que pour une qualité faible. Pour une meilleure qualité la taille des données augmentent entraînant une explosion exponentielle des tailles des tables. Il faut alors une solution avec des opérateurs câblés. Notons que les sources sont forcément à l'infini.
- Les machines PP4, PP5 et Pixel Flow (cf 2.3.3) : Dans ces machine la formule de Phong est calculée par programme. Le nombre de cycle machine dépend du nombre de source et de leur type, il va de quelques milliers à plusieurs dizaines de milliers, ce qui peut paraître très long. Toutefois ces calculs sont faits sur 128×128 processeurs pixel en même temps. On a donc un temps d'éclairement rapide.
- La machine Verve [Knittel93a] [Knittel93b] : La formule d'éclairement de Phong est implémentée "brutalement" et sans restriction, toutes les opérations sont calculées sauf l'élévation à la puissance qui reste déterminée via une table.

On peut discriminer toutes ces machines selon deux critères principaux : selon les fonctionnalités (type de source, nombre, position de l'observateur) ; selon la manière d'implémenter les différents opérateurs, soit par table, soit par calcul. En règle générale, plus les opérations sont calculées, plus on a de souplesse au niveau des fonctionnalités.

4.3.5 Coût mémoire et bande passante

Base de donnée

La taille des base de données peut être très grande pour des scènes complexes. Notamment dans les systèmes classiques, lorsque l'on veut une image de bonne qualité il faut facettiser très fin, c'est à dire approcher l'objet avec un grand nombre de facettes. Cet inconvénient se traduit en espace disque et en temps d'accès. Une facette est généralement décrite par trois sommets auxquels on associe des informations de normales et de texture. On a donc 24 coefficients.

Pour un objet quadrique on a vingt coefficients pour la primitive quadrique, huit par primitive plane, 16 pour la matrice de passage dans le repère de texture et 26 pour la boite englobante (cf annexe 6.3). Toutefois certains de ces coefficients sont codables sur un seul bit. De plus les informations inutiles (comme par exemple la description d'une primitive si elle n'est pas utilisées) peuvent être omises et la quadrique peut être spécifiée avec seulement dix coefficients. Il est donc difficile d'évaluer la taille d'un objet. On prendra comme référence un patch quadrique qui nécessite 44 coefficients.

Toujours en considérant le fait qu'une quadrique est une primitive de plus haut niveau que la facette et que donc elle remplace plusieurs facettes on peut dire que la description d'une scène à partir d'objets quadriques est beaucoup plus concise qu'avec des facettes.

La texture

La gestion du cube de texture engendre théoriquement une multiplication par six des besoins en mémoire. Néanmoins pour tous les objets à peu près plats on utilise une texture plane. De plus pour les objets sphéroïdes la texture est un motif. Dans la plupart des cas on peut utiliser la même texture plane pour chaque face du cube. En définitive les besoins en mémoire de texture sont donc à peu près les mêmes que dans les systèmes classiques.

La mémoire d'image

La table suivante précise les éléments à mémoriser pour chaque pixel de l'écran. Dans la deuxième colonne on donne le nombre de bits pour stocker l'élément considéré.

Elément	Coût
1 numéro de matière	10
2 profondeurs	2 x 24
1 normale	3 x 16
1 point (coordonnées homogènes)	4 x 24
1 quadruplet RVBα	4 x 8
Total	234

Table 4.12: Taille mémoire pour un pixel

Ce qui fait 29 octets par pixel donc 29 Mo pour un écran 1024×1024. Notons que ce n'est pas la quadrique qui engendre ce besoin de mémoire mais l'utilisation de la méthode de Phong pour le calcul de l'éclairement. Remarquons également qu'il faut doubler cette mémoire si l'on veut utiliser la technique du *double-buffer*.

Entre le module de préparation et celui de conversion

Les informations à passer du module de préparation au module de conversion sont celles de la structure de données objetPrepare (cf annexe 6.3). La table suivante donne le nombre de bits nécessaires pour communiquer une telle structure.

Elément	Coût
numéro de matière	5×10
zone écran	4×10
f1	72
f2	288
coeff. normale quadrique	264
coeff. normale plans	288
coeff. profondeur plans	288
orientation	10
Total	1300

Table 4.13: Nombre de bit à transférer au module de conversion pour un objet

Il faut 1300 bits par objet. Pour traiter 100.000 objets par seconde il faut donc assurer un débit de 130 Mbit/s. Réaliser un réseau ayant ce débit ne posent pas de difficultés technologiques.

Entre le module de conversion et le post-processeur

On se fixe que le module de conversion produit cent millions de pixels à la seconde. Le nombre de bits à transmettre par pixel est récapitulé dans la table suivante :

Elément	Coût
numéro de matière	10
normale	48
profondeurs de l'objet	48
le point dans le repère monde	96
couleur	32
Total	234

Table 4.14: Nombre de bits à transférer par pixel

Pour cent millions de pixels il faut donc prévoir une bande passante de 23,4 Gbit/s. Devant ce nombre considérable et devant le nombre de bits nécessaire par pixel on a intérêt à prévoir un bus large de 234 bits (ce qui est large mais réalisable) pour pouvoir ranger un pixel d'un seul coup et réduire la vitesse d'accès mémoire à 100 MHz (ce qui fait un temps d'accès de 10 ns ce qui est raisonnable).

Remarquons par ailleurs que cette quantité de mémoire qui peut sembler monstrueuse est néanmoins envisageable. En fait on peut déjà trouver de telles quantités de mémoires dans certaines machines (par exemple [Akeley93]).

4.4 Simulation et validation

Au cours de nos travaux nous avons développé un logiciel ayant trois objectifs :

- Validation logicielle : des algorithmes spécifiquement développés pour la quadrique et d'un système complet de visualisation basé sur la quadrique
- Simulation fonctionnelle du processeur quadrique pour s'assurer que l'implémentation matérielle est possible
- Production d'images illustrant les possibilités de notre système. Les images d'illustrations se trouve en annexe 6.4.

Afin d'atteindre ces objectifs nous avons conçu un logiciel relativement gros (7000 lignes de code) que nous présentons brièvement.

4.4.1 Le logiciel

Le logiciel a été développé sur station SUN Sparc Station 2 avec une carte vraie couleur. Le programme est écrit en langage C sous Unix. L'interface graphique a été développé avec la boite à outils Xview et le constructeur d'interface DevGuide.

En dehors de l'aspect validation et simulation, les principales fonctionnalités du logiciel sont :

- Gestion des coefficients photométriques des objets
- Gestion des sources lumineuses
- Gestion de différents types d'objets : quadriques naturelles, patchs quadriques (tels que définis par E. Nyiri) et l'objet de base défini au paragraphe 3.2
- Boites englobantes
- Gestion des fichiers de texture
- Définition du système de visualisation (observateur et fenêtre d'observation)
- Affichage de facettes nff

L'architecture logicielle suit celle du pipeline de rendu (Figure 4.16). Le logiciel est découpé en un ensemble de modules. Un premier module correspond au système de visualisation dans son ensemble. Ensuite on trouve un module pour gérer les bases de données (la scène, les matières, les sources lumineuses) Enfin on associe un module logiciel par module matériel (préparation, conversion, éclairement).

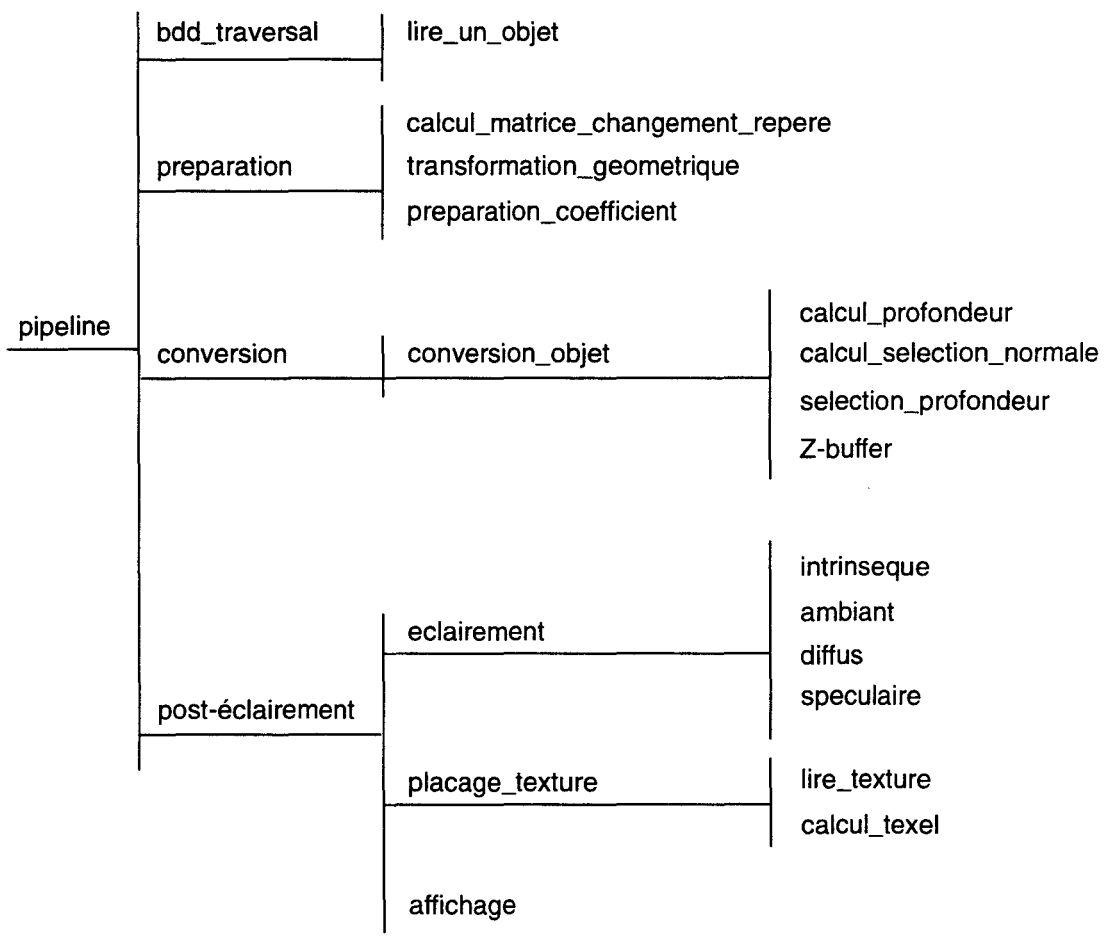


Figure 4.16 ***Architecture logicielle***

4.4.2 Validation logicielle

La validation logicielle de notre système de visualisation basé sur la quadrique repose essentiellement sur les trois modules de préparation, de conversion et d'éclairement. Les algorithmes développés spécifiquement et décrits au chapitre 3 ont pu être vérifiés et mis au point puis intégrés au système complet. Pour cela il nous a fallu définir un certain nombre de structures de données. Celles-ci sont présentées en annexe 6.3. Elles sont également utilisé au paragraphe 4.3.5 pour évaluer les bandes passantes nécessaires entre chaque module.

Le module de préparation ne contient pas de difficultés algorithmiques.

Le module de conversion prend en charge le rendu de base. Pour cela on part des équations (Eq. 3.5), (Eq. 3.9) et (Eq. 3.10) ainsi que des algorithmes donnés au paragraphe 3.3.3. La validité du rendu a été très rapidement confirmée. Les équations et algorithmes ont donc été réécrits pour simuler le fonctionnement du processeur (cf 4.4.3).

Le module d'éclairement comprend la mise en oeuvre de la formule d'éclairement de Phong (cf (Eq. 3.13)) qui ne pose pas de problème algorithmique (le seul problème est lié à la puissance de calcul). Il comprend aussi l'algorithme de placage de texture. Nous avons pu vérifier sa validité ainsi que la faible distorsion introduite par un placage d'une texture plane sur un morceau de surface presque plat.

4.4.3 Simulation du processeur quadrique

Après s'être assuré que les algorithmes proposés au chapitre trois sont valides, il faut vérifier que l'implémentation matérielle du processeur quadrique proposée en début de chapitre corresponde bien fonctionnellement aux algorithmes. Dans le module logiciel de conversion nous avons simulé le processeur quadrique au niveau registre. Certains éléments simples dont le fonctionnement est assuré n'ont pas été simulés. On citera notamment les additionneurs et les multiplicateurs. Les multiplexeurs ont été simulés par des instructions de test simples.

Les premiers éléments qui ont été testés sont les éléments de calcul incrémental, à savoir les PE1 et PE2. les calculs sont effectués en entiers. Les coefficients calculés lors de la préparations sont prévus pour que l'ensemble fonctionne sur 24 bits.

Dans notre programme l'extracteur de racine carrée n'a pas été simulé. Il est réalisé avec l'opérateur proposé par la librairie mathématique du langage C. Néanmoins cet élément a été testé et simulé au niveau transistor par l'auteur lors de son mémoire de DEA [Laporte91]. Son fonctionnement étant relativement lourd à simuler il n'a pas été repris dans le programme pour ne pas trop dégrader les performances.

L'algorithme de sélection de la profondeur a été défini de façon à ne contenir que des opérations élémentaires. Sa simulation est donc directe. Elle a néanmoins permis la mise au point de cet algorithme assez complexe (cf annexe 6.5).

4.5 Bilan

De la conception du processeur et de l'estimation de son coût nous pouvons tirer plusieurs conclusions. La première, sans doute la plus importante, est qu'il est réalisable et que son coût est raisonnable. En effet la technologie VLSI aujourd'hui permet d'intégrer assez facilement quatre ou cinq cent milles transistors dans un même composant.

Comme on pouvait s'y attendre l'ajout du module de calcul de texture augmente notablement la taille du processeur (environ 30%). Cela explique pourquoi le placage de texture est une fonctionnalité qui est longtemps restée l'apanage des accélérateurs haut de gamme.

Notons que les évaluations qui ont été faites ne sont qu'indicatives. En effet on n'a notamment pas tenu compte des chemins de données qui peuvent augmenter considérablement la taille silicium requise. A contrario on a omis certaines optimisations possibles selon la technologie utilisée. En particulier les registres utilisés dans la bibliothèque ecpd15 sont gros. On peut donc espérer un gain substantiel de silicium avec des registres plus petits dans la mesure où l'ensemble du processeur est un pipeline assez profond et donc gourmand en registres de synchronisation.

De l'étude du système complet on peut conclure que celui-ci est réalisable mais qu'il nécessite des développements lourds. En effet la description matérielle du processeur doit être menée à son terme. Cependant le plus gros problème est celui du post-processeur d'éclairement qui est aujourd'hui encore ouvert au niveau de l'implémentation.

La mise en oeuvre logicielle nous a permis d'assurer la validation des algorithmes proposés ainsi que la simulation du processeur. Pour ces deux tâches, le logiciel a fourni un retour qui a favorisé la mise au point des algorithmes. Enfin le logiciel a permis la production d'images afin d'illustrer cette étude, tout spécialement pour les problèmes de placage de textures. Ces images sont en annexe 6.4.

Pour finir ce bilan des travaux de mises en oeuvre nous pouvons dire que nous avons ainsi vérifié que notre système est réalisable. La machine proposée est un croisement entre une architecture d'accélérateur classique (type Silicon Graphics Extreme) et un système basé sur le pipeline de Phong qui de plus prend en compte les spécificités introduites par la primitive quadrique.

Synthèse, conclusion et perspectives

Dans un premier temps nous rappelons la démarche que nous avons suivie pour cette étude, les problèmes auxquels nous avons apporté une réponse et les développements concomitants.

Notre travail se situe dans un contexte de visualisation temps-réel dans lequel s'inscrivent déjà les accélérateurs actuels basés sur la facette. Il semble donc intéressant de positionner notre système par rapport aux autres, d'un point de vue des fonctionnalités, des performances, du coût et de la qualité des images produites.

De 1989 à 1993 une étude avait déjà été menée au LIFL sur la quadrique. L'étude présentée dans ce mémoire en est le prolongement. Une autre étude s'est poursuivie en parallèle, sur la modélisation avec les quadriques. En tenant compte de tous ces travaux nous élargissons notre conclusion. Nous statuons ainsi sur l'avenir d'une machine basée sur la quadrique. Par ailleurs cette étude nous conduit à faire des remarques générales sur l'architecture globale des accélérateurs graphiques, qu'ils soient basés sur la quadrique ou sur la facette.

Pour finir nous présentons les voies de recherches encore ouvertes qui nous semblent intéressantes à explorer.

5.1 Résumé de la proposition

Notre motivation de départ est l'écart remarquable entre primitive de modélisation et primitive de visualisation. En effet alors que les modeleurs travaillent avec des primitives telles que les surfaces de Bézier ou B-spline, les accélérateurs courants ne sont capables d'afficher que des facettes. Cet écart a plusieurs conséquences : d'une part la nécessité des algorithmes de facettisation qui souvent compromettent l'interactivité ; d'autre part une approximation sur les objets effectivement affichés qui entraîne une perte de qualité sur les images calculées. En particulier les objets affichés ont un aspect anguleux dû aux bords des facettes.

Ces constatations nous conduisent à remettre en cause la primitive de visualisation habituelle, à savoir la facette. Pour des raisons de complexité liées à l'usage de matériel spécifique indispensable pour répondre aux contraintes temporelles, notre choix se porte sur la primitive courbe la plus simple : la quadrique. Toutefois une étude préliminaire montre que la quadrique ne peut à elle seule constituer une primitive de visualisation souple et pratique. Nous sommes donc amenés à définir de façon rigoureuse notre primitive de visualisation. Nous utilisons une quadrique et plusieurs plans (en l'occurrence quatre). La définition mathématique est suffisamment générale pour pouvoir gérer les quadriques naturelles, les patchs quadriques, les facettes et les polyèdres.

Nous choisissons ensuite l'architecture du pipeline de Phong pour notre machine. En effet le pipeline de Gouraud habituellement utilisé ne nous convient pas à cause de la faible qualité de l'éclairement. En revanche le pipeline de Phong offre cette qualité. De plus la quadrique s'adapte bien à ce pipeline. En effet celui-ci est basé sur l'interpolation de la normale. Or il est facile de calculer la normale exacte en chaque point d'une quadrique.

L'architecture de notre système de visualisation étant choisie, nous nous focalisons sur la définition des algorithmes de rendu. En effet l'introduction de la quadrique et plus généralement le changement de primitive de visualisation nous contraignent à reprendre ces algorithmes, à les adapter et parfois à en proposer de nouveaux. Les difficultés principales sont la gestion des profondeurs multiples (notre primitive de visualisation comprend six profondeurs : deux pour la quadrique et une pour chacun des quatre plans), l'ombrage (calcul de normale et du point dans le repère monde) et enfin le placage de texture.

La dernière étape dans notre démarche est la conception matérielle de notre système en général et du processeur de rendu en particulier. Pour ce dernier, qui est le point névralgique de notre système, une estimation est réalisée afin de s'assurer de sa faisabilité. Toutes ces études s'appuient sur un logiciel de validation et de simulation.

En conclusion la machine que nous proposons est technologiquement réalisable. De plus elle offre de bonnes capacités d'affichage tant sur le plan fonctionnel que sur le plan des performances.

5.2 Comparaison avec les accélérateurs actuels

Nous récapitulons les avantages et les inconvénients de notre système par rapport aux accélérateurs classiques. Pour cela nous opérons en deux temps. Nous revenons sur le choix du pipeline de Phong dont les caractéristiques sont indépendantes de la primitive de visualisation. Ensuite nous récapitulons les bénéfices et les désavantages dus plus spécifiquement au remplacement de la facette par la quadrique.

Par ailleurs nous réalisons une analyse du coût engendré par la quadrique par rapport à la facette. Cette analyse est intéressante pour comprendre pourquoi la quadrique revient plus cher que la facette. De plus on pourra utiliser cette analyse pour évaluer rapidement toute modification apportée au processeur.

Phong vs. Gouraud

La qualité principale du pipeline de Phong par rapport à celui de Gouraud porte sur l'éclairement qui s'avère bien meilleur. Malheureusement cette amélioration suppose un post-traitement qui est clairement une surcharge de travail. Celle-ci engendre des modifications architecturales qui supposent des développements matériels particuliers.

On peut donc se poser la question de savoir s'il ne vaudrait pas mieux utiliser la quadrique dans une architecture de type Gouraud. Au paragraphe 3.1. nous avons montré qu'il était envisageable de se passer de post-éclairement de Phong soit en calculant directement un éclairement de Lambert (composante ambiante et diffuse), soit en effectuant une interpolation bi-quadratique de couleur (cette dernière voie restant à explorer).

Néanmoins ces deux solutions comportent des limitations et appauvrissent la qualité de l'éclairement. L'emploi de ces méthodes reviendrait à retomber dans les défauts de la facette que l'on cherche justement à éliminer. Le post-éclairement de Phong, malgré ces défauts, nous semble incontournable. Au pire une solution sans post-traitement pourrait constituer une solution dégradée peu onéreuse.

Quadrique vs. facette

Le premier point positif en faveur de notre système est qu'il offre les même fonctionnalités qu'un système classique. On citera bien sûr la conversion objet pixel, l'ombrage et l'élimination des parties cachées. Mais de plus il peut gérer des effets sophistiqués tels que l'anti-aliassage et le placage de texture.

Ensuite on peut profiter des apports de notre primitive. Par exemple sa capacité à gérer les objets volumiques permet de faire du rendu CSG facilement. D'autre part on calcule la vraie

normale et donc la qualité de l'éclairement est meilleure. Au niveau de la qualité visuelle le dernier avantage (et non des moindres) est la qualité géométrique : les objets n'ont plus leur aspect anguleux car la quadrique est une surface courbe.

D'autre part une quadrique peut remplacer de nombreuses facettes : de une dans le cas le plus défavorable à plusieurs centaines dans le cas d'une sphère par exemple. Cette réduction du nombre d'objets se répercute sur la taille des scènes et sur la charge de travail.

Au niveau des performances l'architecture de notre système, proposée au paragraphe 4.3, ne permet qu'un parallélisme faible de quelques processeurs. Elle convient donc uniquement pour des accélérateurs bas et milieu de gamme. Le processeur quadrique décrit au paragraphe 4.1 est en mesure de produire un pixel par cycle d'horloge. Pour une horloge de 50 Mhz et un processeur de rendu composé de deux processeurs quadriques on produira 100 Mpixels/s ce qui est typique d'une petite machine. Pour une machine un peu plus puissante on peut par exemple doubler la vitesse d'horloge et le nombre de processeurs quadriques pour obtenir un débit de 400 Mpixels/s.

Pour espérer obtenir des performances de rendu comparables aux machines haut de gamme il faut repenser l'architecture complète du système. En effet de très hautes performances nécessitent l'emploi d'un parallélisme important voire massif. Cette étude s'inscrit donc dans les perspectives de ce travail.

Pour la préparation on a vu que la charge de travail était réduite par rapport aux systèmes classiques, pour deux raisons. D'une part l'éclairement est déporté après l'élimination des parties cachées. D'autre part, le nombre d'objets à préparer est moindre et donc les calculs sont réduits en proportion.

Pour ce qui est du coût silicium notre processeur est environ dix fois plus cher que le processeur facette. Il est clair que le processeur facette possède là un avantage important. Néanmoins nous considérons que cet avantage est contrebalancé par les gains cités précédemment.

Analyse du coût

Le problème majeur de notre primitive de visualisation est sa ou plus exactement ses profondeurs. D'une part il faut les calculer. D'autre part elles complexifient le calcul des expressions linéaires et rationnelles linéaires que l'on retrouve dans les processeurs de rendu pour effectuer des traitements tels que le calcul d'une normale ou des coordonnées de textures.

Tout d'abord il faut être capable de calculer toutes les profondeurs : six au lieu d'une seule pour la facette. Il faut donc davantage de modules de calcul. De plus pour la quadrique ces modules sont plus compliqués. En effet il faut un interpolateur du premier degré (comme pour la facette) mais il faut en plus un interpolateur du second degré et surtout un extracteur de racine carrée, Lorsque l'on a déterminé toutes les profondeurs, un module spécifique supplémentaire est nécessaire pour sélectionner la bonne profondeur.

En général les expressions à calculer lors du rendu sont linéaires en x, y et z. Le grand point fort de la facette est que sa profondeur est elle-même linéaire en x et y. Dans ce cas toutes les expressions se ramènent à des expressions linéaires en x, y que l'on peut interpoler pour le prix d'un additionneur en utilisant le calcul incrémental. Pour la quadrique la profondeur n'est pas linéairement dépendante des coordonnées x et y comme pour la facette. On doit donc calculer les expressions en x, y **et** z. Le calcul incrémental n'est plus utilisable complètement. On ne l'utilise que sur le terme linéaire en x, y puis l'on ajoute le terme en z qui nécessite une multiplication (la profondeur z multipliée par une constante). Le surcoût est un additionneur et surtout un multiplicateur.

Lorsque l'on a des expressions rationnelles linéaires, comme pour le calcul des coordonnées de texture, on interpole linéairement les numérateurs et le dénominateur de ces expressions et on effectue les divisions ensuite. On limite ainsi le surcoût aux diviseurs. Pour la quadrique on paye l'addition et la multiplication supplémentaire par expression linéaire en x, y, z mais en revanche les divisions sont les mêmes que pour la facette. Sachant que ce sont les diviseurs qui

coûtent le plus chers, on comprend pourquoi le placage de texture est proportionnellement moins cher dans notre processeur que dans un processeur facette.

5.3 Conclusion

Nous donnons tout d'abord notre avis quant à l'avenir de la quadrique dans un système de visualisation temps réel. Pour cela nous utilisons les résultats présentés ci-dessus ainsi que les résultats de Maxime Froumentin. Nous élaborons ainsi une conclusion globale des études menées sur la quadrique au LIFL depuis sept ans.

Par ailleurs l'étude de ce système a fait ressortir des caractéristiques architecturales intéressantes car indépendantes de la primitive de visualisation choisie. Nous revenons donc sur le traitement inter-objet et le post-traitement.

L'expérience quadrique au LIFL

Notre réflexion sur la quadrique au LIFL arrive à son terme. Ce mémoire décrit un système de visualisation basé sur une primitive quadrique. Nous avons montré qu'un tel système est réalisable, qu'il offre un certain nombre d'avantages sur les accélérateurs courants, avantages contrebalancés par un certain nombre d'inconvénients. Notre machine représente une alternative intéressante et réaliste aux accélérateurs usuels.

Le mémoire de Maxime Froumentin présente les possibilités de modéliser avec des quadriques. Il montre que bien qu'il existe un certain nombre de cas où la quadrique se révèle pratique pour la modélisation, elle reste néanmoins une primitive de modélisation assez pauvre dans le cas général.

En conclusion nous pensons qu'il n'est pas pertinent de prolonger l'expérience quadrique jusqu'à une réalisation effective d'une machine. En effet dans le cas général le bilan avantage-inconvénient n'est pas assez positif pour justifier les efforts de développement d'un nouveau système. Pour les cas particuliers nettement favorables tels l'affichage de quadriques naturelles le champ d'application est trop réduit.

Traitement inter-objet

Les études sur la quadrique nous ont très naturellement amenés à réfléchir sur les possibilités de travailler sur les volumes. Nous avons en particulier étudié le rendu direct CSG ainsi que les volumes d'ombres (pour la gestion des ombres portées) et la transparence. L'implémentation des algorithmes relatifs à ces traitements nous a conduits à isoler une partie de notre pipeline de rendu, que nous avons appelée traitement inter-objet. Il s'agit de tous les traitements qui concernent plusieurs objets, effectués au niveau pixel et donc en fonction de comparaisons sur les profondeurs. Le plus simple et le plus connu de ces traitements est l'algorithme du z-buffer.

Ces traitements ne nécessitent que peu de ressources matérielles. En revanche ils exigent beaucoup de ressources en mémoire. De plus ils supposent souvent un rendu en plusieurs passes. Il est donc difficile d'assurer un rendu temps-réel. Toutefois comme ces algorithmes sont en général assez sophistiqués il est tout à fait acceptable d'avoir "simplement" un rendu rapide. Les applications, notamment en CAO pour le CSG sont très importantes. De plus l'intégration de ces algorithmes dans les architectures classiques n'est pas très complexe. Il nous semble donc que les prochains accélérateurs devraient intégrer ces fonctionnalités.

Post-traitement

Par définition un post-traitement est un traitement qui s'effectue sur chaque pixel de l'écran après l'élimination des parties cachées.

Cette distinction est importante. En effet aujourd'hui on calcule typiquement 100 millions de pixels par seconde pour vingt cinq images soit quatre millions de pixels par image. Or un écran

est typiquement composé d'un million de pixels. Si l'on effectue un traitement après l'élimination des parties cachées on ne travaille que sur le million de pixels de l'image au lieu des quatre millions nécessaires à la construction de cette image. On a donc un gain de quatre sur les calculs à effectuer.

Toutefois il est à noter que certaines opérations (typiquement les interpolations) peuvent être réalisées en calcul incrémental si elles sont effectuées lors du rendu. On a donc intérêt à ne pas les différer même en tenant compte du facteur quatre.

Le principal post-traitement est le calcul de l'éclairement de Phong. Dans notre système on peut citer également la prise en compte des drapeaux d'ombre portée pour le calcul de l'éclairement. On peut citer la possibilité de calculer des textures de manière procédurale comme dans la machine Pixel Flow [Molnar92].

Par ailleurs on remarque que plus on augmente le nombre et la complexité des traitements et plus il faut augmenter la puissance du post-processeur ou sacrifier les performances temps-réel. Un choix raisonnable est d'assurer l'éclairement en temps réel et d'assurer les autres fonctionnalités avec une légère dégradation des performances. On obtient ainsi un bon compromis entre performance et puissance de calcul.

A notre avis le post-traitement prendra une place de plus en plus importante dans les architectures des machines futures. En effet la technologie permet de plus en plus d'intégrer des unités de calculs avec la mémoire (d'image ou de texture). Cette même technologie permet d'accroître la taille des mémoires et donc offre la capacité de stockage des informations nécessaires aux calculs.

Dans la mesure où les opérateurs cablés offrent une plus grande souplesse que les tables, nous estimons que les premiers vont faire disparaître les seconds. D'autre part l'architecture des post-processeurs pourrait être un réseau de processeurs suivant deux types de contrôle : SIMD pour simplifier le contrôle ; MIMD pour plus de souplesse et dans la mesure où les traitements sont de type SPMD[1]. En revanche il est difficile de dire si ces processeurs seront spécifiques ou généraux. On risque d'avoir un phénomène déjà existant pour les processeurs de préparation. Le choix est fait selon des considérations technologiques et non architecturales.

5.4 Perspectives

Lorsque l'on reprend le raisonnement du chapitre un sur l'écart existant entre primitive de modélisation et primitive de visualisation, on se rend compte que le choix de la quadrique a été dicté par des contraintes matérielles. Avec l'évolution de la technologie ces contraintes seront de moins en moins gênantes. On en vient donc tout naturellement à se demander si l'on ne pourrait pas choisir une primitive de plus haut niveau que la quadrique.

De la même façon que l'on est passé des surfaces de degré un (la facette) aux surfaces de degré deux (la quadrique), on peut chercher à passer aux surfaces de degré immédiatement supérieur. Les cubiques et les quartiques sont les surfaces respectivement de degré trois et quatre.

Remarquons qu'en modélisation il est intéressant de passer de la quadrique à la cubique et de la cubique à la quartique. En effet les cubiques permettent de résoudre les problèmes rencontrés pour les macro-patchs avec la quadrique. La quartique quant à elle permet de représenter des carreaux de Bézier triangulaires quadratiques (cf [Froumentin96]). Devant les piètres possibilités de la quadrique en modélisation il semble donc intéressant de regarder les possibilités d'afficher les cubiques et les quartiques.

Dans ce sens, remarquons que l'étude qui a été faite pour la quadrique reste valable pour la cubique et la quartique sauf pour la gestion des profondeurs. En effet le nombre de profondeurs

1. Single Program Multiple Processeur. Tous les processeurs exécutent le même programme mais par le jeu des tests ils n'exécutent pas forcément les mêmes morceaux de code

pour la cubique est de trois et pour la quartique de quatre. La difficulté est l'extraction de racine cubique et quartique. On a vu que l'extraction d'une racine carrée est déjà très coûteuse. Ce problème ne peut être résolu qu'avec les progrès de la technologie. Ensuite il faut également adapter l'algorithme de sélection de la bonne profondeur. On peut toutefois conserver l'idée de base qui consiste à travailler au moyen de comparaisons sur les profondeurs.

En revanche le reste de l'étude reste pertinent. La primitive de visualisation est toujours un morceau de surface (a priori soit la cubique soit la quartique) délimité par un volume défini par l'intersection de demi-espaces définis par des plans. Le calcul du point dans le repère monde est inchangé. La normale est toujours obtenue à partir des dérivées partielles qui sont cette fois-ci de degré deux ou trois. Cela suppose des interpolateurs quadratiques et cubiques et des multiplicateurs. Les traitements inter-objet sont identiques dans la mesure où ils fonctionnent sur les profondeurs. Le post-éclairement opère sur des pixels, il n'est donc pas concerné par la nature de la primitive de visualisation. Le placage de texture travaille indépendamment du type de surface (il travaille sur des points), il est donc réutilisable directement. Enfin l'architecture générale peut être conservée.

Annexes

6.1 Modèles d'éclairement

On distingue généralement deux types de modèles d'éclairement, les modèles locaux et les modèles globaux. Les premiers ne prennent en compte que l'éclairement direct des objets par les sources lumineuses. Les seconds tiennent compte en plus des interactions lumineuses entre les objets. Dans cette catégories on citera le modèle proposé par Whitted pour le lancer de rayon [Whitted80] et la radiosité [Goral84].

En rendu projectif on utilise des modèles locaux pour leur simplicité. Nous présentons les formule d'éclairement ambiant, de Lambert, de Phong et de Blinn.

6.1.1 Grandeurs gémétriques et caractéristiques des objets

Pour comprendre les formules d'éclairement présentées ci-dessous, il faut tout d'abord introduire certaines grandeurs utilisées dans les formules. Ces grandeurs ont trait soit à la géométrie, soit au caractéristiques intrinsèques des objets.

Caractéristiques des objets

Les objets sont des sources lumineuses ou des surfaces qui réfléchissent la lumière émise directement par les sources. Les caractéristiques importantes sont les suivantes :

- I_l est l'intensité de la source lumineuse
- K_a, K_d, K_s sont les coefficients de réfection ambiant, diffus et spéculaire de matériau
- s l'exposant spéculaire
- O la couleur intrinsèque de l'objet

Les grandeurs géométriques

La Figure 6.1 représente les grandeurs géométriques qui sont utilisées dans les modèles d'éclairement que nous présentons.

- $\vec{N}$ la normale à l'objet au point considéré
- $\vec{L}$ le vecteur lumière c'est-à-dire le vecteur allant du point à la source lumineuse
- $\vec{O}$ le vecteur observateur c'est-à-dire le vecteur allant du point à l'observateur
- $\vec{R}$ le vecteur réfléchi c'est-à-dire le symétrique du vecteur lumière par rapport à la normale
- $\vec{H}$ le vecteur surbrillance

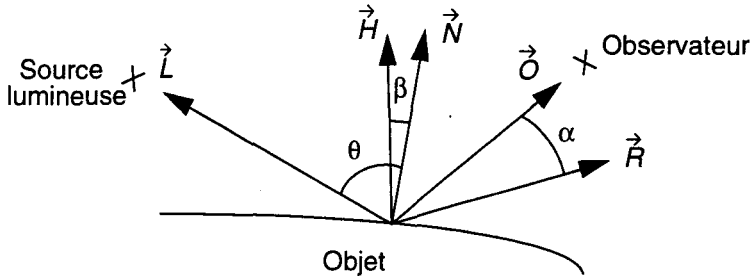


Figure 6.1 ***Grandeurs géométriques utilisées dans les formules d'éclairement***

Remarquons que $\vec{N} \bullet \vec{L} = \cos(\theta)$, les vecteurs $\vec{N}$ et $\vec{L}$ étant normalisés et θ étant l'angle entre la normale et le vecteur lumière. De même $\vec{R} \bullet \vec{O} = \cos(\alpha)$, les vecteurs $\vec{R}$ et $\vec{O}$ étant normalisés et α étant l'angle entre le vecteur réfléchi et le vecteur observateur.

6.1.2 Les formules d'éclairement

Eclairement ambiant

L'éclairement ambiant rend compte de l'illumination globale de la scène (ce n'est qu'une faible approximation comparé à la méthode de radiosité).

$$I = K_a I_l O \qquad \textbf{(Eq. 6.1)}$$

La loi de Lambert

La loi de Lambert rend compte de la réflexion diffuse de la lumière par une surface, c'est-à-dire de la partie de la lumière renvoyée par l'objet dans toutes les directions.

$$I = K_d I_l O\left(\vec{N} \bullet \vec{L}\right) \qquad \textbf{(Eq. 6.2)}$$

En général lorsque l'on utilise la formule de Lambert on y ajoute l'éclairement ambiant.

Eclairement de Phong

B. T. Phong [Phong75] a proposé une formule empirique qui rend compte des taches spéculaires (appelées aussi surbrillance). L'intensité I au point considéré se compose des composante ambiante et diffuse plus une troisième composante, la composante spéculaire, qui rend compte des surbrillances.

$$I = K_a I_l O + K_d I_l O\left(\vec{N} \bullet \vec{L}\right) + K_s I_l \left(\vec{R} \bullet \vec{O}\right)^s \qquad \textbf{(Eq. 6.3)}$$

La lumière est renvoyée de façon privilégiée dans la direction de $\vec{R}$. Lorsque l'observateur se trouve dans cette direction il voit la surbrillance. Lorsqu'il s'en écarte la surbrillance disparaît rapidement. La surbrillance est plus ou moins intense (coefficient K_s dans l'équation (Eq. 6.3)) et disparaît plus ou moins vite (coefficient s dans l'équation (Eq. 6.3)) en fonction du matériau.

La formule de Blinn

Blinn a proposé une amélioration de la formule de Phong qui repose sur le vecteur surbrillance $\vec{H}$ [Blinn78b]. Ce vecteur est le vecteur bissecteur de $\vec{L}$ et $\vec{O}$. La formule (Eq. 6.3) se réécrit ainsi :

$$I = K_a I_l O + K_d I_l O\left(\vec{N} \bullet \vec{L}\right) + K_s I_l \left(\vec{N} \bullet \vec{H}\right)^s \qquad \textbf{(Eq. 6.4)}$$

Cette formule est particulièrement intéressante lorsque la source lumineuse et l'observateur sont à l'infini. En effet dans ce cas le vecteur $\vec{H}$ est constant ce qui allège grandement le calcul. Dans la formule (Eq. 6.3) le vecteur $\vec{R}$ varie de toute façon. En effet $\vec{R} = 2\vec{N}\left(\vec{N} \bullet \vec{L}\right) - \vec{L}$, donc dépendant de $\vec{N}$ qui est calculée en chaque point. Notons que si la source lumineuse ou l'observateur ne sont plus à l'infini, la formule (Eq. 6.4) reste un peu moins coûteuse.

6.2 Post-éclairement de Phong

Pour bien comprendre son impact sur la charge de calcul nous évaluons son coût de calcul selon différents paramètres. Ensuite nous présentons les différentes solutions dégradées qui visent à réduire le coût de l'éclairement sans trop perdre la qualité visuelle.

6.2.1 Coût de calcul

Le calcul de l'éclairement selon la méthode de Phong est coûteux car il impose un calcul assez complexe pour tous les pixel de l'écran. Nous allons maintenant estimer ce coût[1] afin d'illustrer ce fait d'une part et pour donner une idée des moyens à mettre en oeuvre d'autre part. Dans le chapitre suivant (cf. paragraphe 4.3.4) nous présenterons les propositions existantes pour réaliser ce calcul en temps réel et nous essayerons d'en tirer des conclusions pour l'avenir.

Avant d'estimer le calcul remarquons que ce calcul n'est effectué qu'après l'élimination des parties cachées. On borne ainsi le nombre de pixels à éclairer par le nombre de pixels de l'écran. Par ailleurs, ce calcul peut être grandement simplifié si l'observateur et la source sont à l'infini. Nous proposons donc une estimation lorsque l'on est dans ce cas et une lorsque l'on est dans le cas général. Enfin les calculs sont estimés pour une seule source ponctuelle. Le coût pour n sources est multiplié par n à ceci près que certains calculs peuvent être communs à toutes les sources (notamment le calcul de l'intensité ambiante, la normalisation de la normale et le calcul du vecteur observateur).

Reprenons l'équation (Eq. 3.13). Dans cette équation seuls les produits scalaires sont à calculer pour chaque pixel. Nous allons donc nous y intéresser de plus près. La normale est calculée dans le convertisseur objet pixel, mais il faut la normaliser.

Pour une source et un observateur situés à distance infinies tous les vecteurs autres que la normale sont constants. Il suffit de les normaliser une fois par image et donc nous négligerons ce coût. Il ne reste donc qu'à calculer les produits scalaires et la formule complète.

Pour une source et un observateur situés à distance finie il faut calculer les vecteurs observateur et lumière, qu'il faut ensuite normaliser. Après il faut calculer le vecteur surbrillance et le normaliser. Finalement on calcule les produits scalaires et l'éclairement au point considéré.

1. Une étude plus détaillée se trouve dans [Lefévère94]

La table ci-dessous résume le coût approximatif des deux calculs pour un pixel. La première colonne indique le type d'opération, la deuxième le nombre d'opérations pour un observateur et une source à l'infini et la troisième le nombre d'opérations pour une source et un observateur à distance finie. On peut donner un total en nombre d'opérations de base en utilisant les données de [Hennessy90] p.43. En considérant l'addition comme opération de base, une multiplication correspond à une additions, une division ou une racine carrée correspond à quatre additions et une élévation à la puissance correspond à huit additions.

Addition / soustraction	12	25
Multiplication	16	30
Division	1	3
Extraction de racine carrée	1	3
Elévation à la puissance	1	1
Total	44	87

Table 6.1: Coût de calcul de la formule de Phong

Si l'on veut afficher des images sur un écran d'un million de pixels à la vitesse de vingt-cinq images par seconde, on a donc un besoin en puissance de calcul de 1,1 Gflops dans le premier cas et de 2,2 Gflops dans le cas général, ce qui est considérable.

6.2.2 Méthodes de réduction du coût

Un certain nombre de travaux ont visé à réduire le coût du calcul d'éclairement de Phong.

- Calcul de la formule de Blinn sans normaliser la normale [Claussen89]. La qualité n'est pas excellente mais Claussen montre qu'elle est meilleure qu'avec l'ombrage de Gouraud.
- L'interpolation des produit scalaires $\vec{N} \bullet \vec{L}$ et $\vec{N} \bullet \vec{H}$. Cette méthode évite la normalisation de la normale.
- la technique appelée *fast Phong shading* [Bishop86]. La formule de Phong est calculée au moyen d'une série de Taylor du second degré. Le calcul incrémental permet de ramener le coût à deux additions par pixels. Toutefois il faut développé une série pour chaque source lumineuse. De plus on est limité aux sources directionnelles.
- L'interpolation angulaire ou *faster Phong shading*, proposée par Kuijk et Blake [Kuijk89]. On interpole la normale en coordonnées polaires ce qui permet de conserver une norme unitaire. On gagne ainsi la normalisation de la normale.

On voit donc que l'on peut réduire le coût de la méthode d'éclairement de Phong au prix d'une perte de qualité et/ou de limitations (sur le nombre ou le type des sources lumineuses ou sur la position de l'observateur).

Néanmoins il est clair que pour garder une qualité optimale et la généralité de la formule il est nécessaire de faire le calcul sans transiger sur le coût. Les solutions algorithmiques sont connues. En général on calcule la normale en tout point (pour des facettes on peut l'interpoler) puis on la normalise puis on utilise la formule de Blinn.

En définitive le problème se situe au niveau de l'implémentation. Aujourd'hui aucun accélérateur graphique réalisé n'offre un tel type d'éclairement. Toutefois quelques propositions ont été faites, que nous présentons brièvement au paragraphe 4.3.4.

6.3 Formats de données

Notre logiciel de validation et simulation comprend trois structures de données principales. La première concerne le stockage des objets lus dans une base de données sur disque. La seconde correspond aux informations échangées entre le module de préparation et celui de conversion. La troisième correspond aux informations échangées entre le module de conversion et le module de post-éclairement. Nous présentons successivement ces trois structures. Mais tout d'abord nous décrivons le format de fichier utilisé pour la description des scènes.

6.3.1 Description de scène

Afin de produire des images nous avons du définir un format de scène en conformité avec la définition de notre objet de base du chapitre trois. Les informations de lumière et le système de visualisation sont stockés dans des fichiers séparés. Dans le fichier de description de scène on ne trouve donc que les objets de la scènes, placés dans le repère monde.

Le fichier commence par un mot clé qui indique le type d'objet qui s'y trouvent. En effet pour des raisons de compatibilité avec les travaux d'Eric Nyiri et de Maxime Froumentin nous avons trois type d'objets :

- TETRAEDRE : correspond au patch quadrique
- MACROPATCH : correspond au macropatch a quatre patchs
- GENERIQUE : correspond à notre objet de base

Notons que les deux premiers ne sont que des cas particuliers du troisième. On pourrait en envisager d'autres (par exemple les facettes). On pourrait également s'en passer. Toutefois ils sont pratiques. Tout d'abord nous décrivons le format GENERIQUE. Ensuite nous précisons les particularités des autres formats.

Le format GENERIQUE

Au niveau de la description de l'objet le nombre de primitives est variable. On a zéro ou une quadrique et de zéro à huit plans. Pour spécifier une quadrique on a la ligne suivante :

Q a v c

Avec a le bit d'activité qui indique si la primitive participe effectivement à la définition de l'objet. v est le bit de visibilité. c indique le numéro de couleur de la primitive et plus généralement le numéro de matière. Si la quadrique est active (a=1) alors on donne ensuite la définition de la quadrique sous forme d'une matrice 4×4. La matrice est omise dans le cas contraire.

Un plan est spécifié par une ligne de la forme :

P a v c

Si le plan est actif on trouve ensuite sa définition au moyen d'un vecteur de quatre réels. Dans le cas contraire le vecteur est omis et de plus on considère que toutes les primitives de l'objet ont été spécifiées.

Pour finir on peut rajouter une boite englobante :

E x

x indique l'existence ou non de la boite englobante. Si elle existe (x=1) alors on donne huit points selon le schéma de la Figure 6.2.

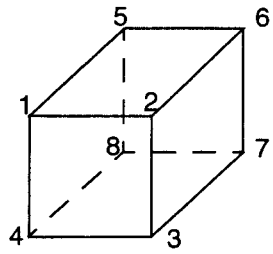


Figure 6.2 ***Boite englobante***

Le format TETRAEDRE

Un patch est spécifié au moyen d'une quadrique et de quatre points qui sont les sommets du tétraèdre. Ces points servent à la fois pour déterminer les plans de coupes et comme boite englobante.

Le format MACROPATCH

Un macropatch est spécifié avec quatre patchs et un polygone de contrôle composé de six points (Figure 6.3).

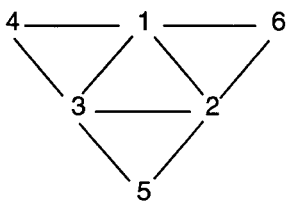


Figure 6.3 ***Polygone de contrôle***

6.3.2 L'objet à afficher

Lors de la traversée de la base de donnée on lit les objets dans le fichier de description de scène selon le format spécifié au paragraphe précédent. Les objets sont ensuite stockés dans une structure de données. Avant de la présenter nous définissons les deux types suivants :

```
typedef float mat44[4][4];/* défini une matrice 4 x 4 */

typedef float vect4[4];/* défini un point en coordonnées homogènes */


typedef objetDeBase * objPtr;/* pointeur sur un objet */


typedef struct {
|     mat44 q; /* la quadrique dans le repere écran */
|     vect4 plan[4];/* les plans dans le repere écran*/
|     int ba[5]; /* les bits d'activite */
|     int bv[5];/* les bits de visibilite */
|     int num_mat[5];/* les numeros de matiere pour chaque primitive */
```

```
|   vect4 pbe[8];/* les sommets de la boite englobante */
|   int be; /* boite englobante existante ou non */
|   struct objetDeBase *suivant;/* pointeur sur l'objet suivant */
} objetDeBase;
```

La scène est rangée dans une liste de objetDeBase.

6.3.3 L'objet préparé

Lors de la phase de préparation on fait subir différentes opérations aux objets lus depuis la base de données. On calcule notamment les coefficients qui serviront lors de la phase de conversion. Pour stocker ces nouvelles informations on utilise une seconde structure de données.

```
typedef objetPrepare * pretPtr; /* pointeur sur un objet prepare */


typedef struct {
|   int num_mat[5];/* les numeros de matiere pour chaque primitive */
|   int xmin,xmax,ymin,ymax;/* zone ecran ou se trouve l'objet */
|   int f1[3];/* coeff. de f1(x,y) pour les prof. de la quadrique */
|   long int delta[6];/* coeff. de f2(x,y) pour les prof. de la quadrique */
|   int a,b,d,e,f,g,h,i,cp,ep,fp;/* coeff pour les normales de la quadriques */
|   int czp[4][3];/* coeff. pour la profondeur des plans */
|   int np[4][3];/* coeff. pour la normale des plans */
|   int orient[5];/* orientation de la quadrique et des plans */
|   struct objetPrepare *suivant;/* pointeur sur le suivant */
} objetPrepare;
```

6.3.4 Information pixel

Dans un système classique on ne stocke, au niveau du pixel, que la profondeur de l'objet le plus proche et sa couleur. Dans notre cas nous devons stocker au minimum les deux profondeurs, la normale, le point exprimé dans le repère monde en coordonnées homogènes qui sert pour le post-éclairement. Il faut également stocker le numéro de matière. Ce numéro sert pour récupérer les coefficients photométriques de la matière utiles lors du calcul de l'éclairement, il est donc indispensable. Enfin il faut stocker la couleur au pixel.

On aboutit à la structure suivante.

```
typedef struct {
|   int num;/* numéro de matière */
|   int nx,ny,nz;/* normale */
|   int z1,z2;/* profondeurs de l'objet */
|   float xm,ym,zm,wm;/* point dans le repère monde */
|   unsigned char r,v,b,alpha;/* couleur et transparence du pixel */
} buffer_pixel;
```

6.4 Images

Les images présentées dans cette annexe ont été produites avec le logiciel de validation et simulation développé au cours de mes travaux. Elles illustrent les raisonnements tenus dans ce rapport ainsi que les possibilités que pourrait avoir un système de visualisation basé sur la quadrique.

6.4.1 Contour facétisé

Le principal défaut dû à l'utilisation de la facette comme primitive de visualisation est l'aspect facétisé des objets, notamment sur les contours. Pour diminuer le problème on peut facétiser plus fin. Le résultat est meilleur mais le problème n'est pas résolu il est simplement repoussé. Pour obtenir une qualité acceptable on peut avoir besoin d'un très grand nombre de facettes. A ce titre, l'exemple de la sphère est particulièrement éloquent.

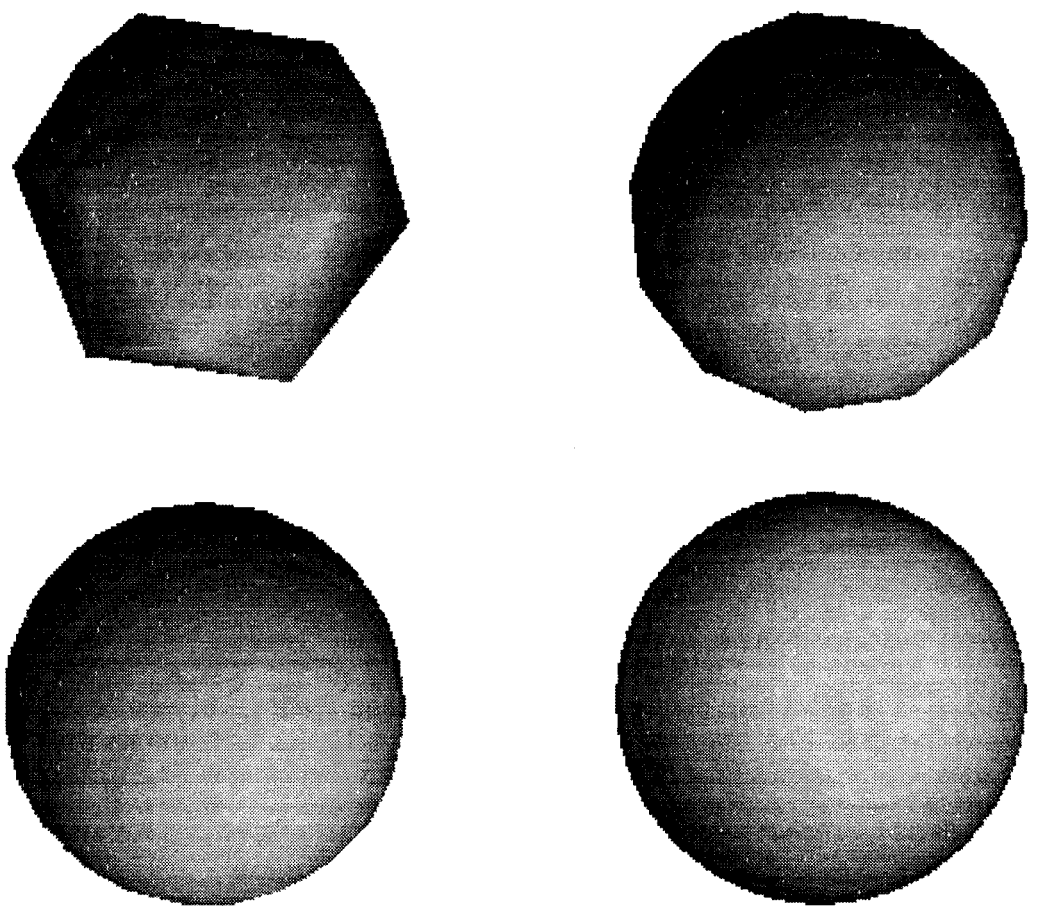


Figure 6.4 *Sphères facétisées : trois sphères avec 32 facettes (en haut à gauche) 128 facettes (en haut à droite) et 512 facettes (en bas à gauche). La quatrième sphère est une vraie sphère (ie une quadrique affichée directement)*

6.4.2 Objet CSG

Un des traitements inter-objets intéressant est l'affichage direct d'objet CSG. Nous avons implémenté un algorithme non optimisé mais qui permet toutefois de valider le principe. On

présente dans la figure suivante un premier exemple où l'objet est l'intersection de deux ellipsoïdes. Dans le deuxième exemple, l'objet est un ellipsoïde coupé par un autre ellipsoïde.

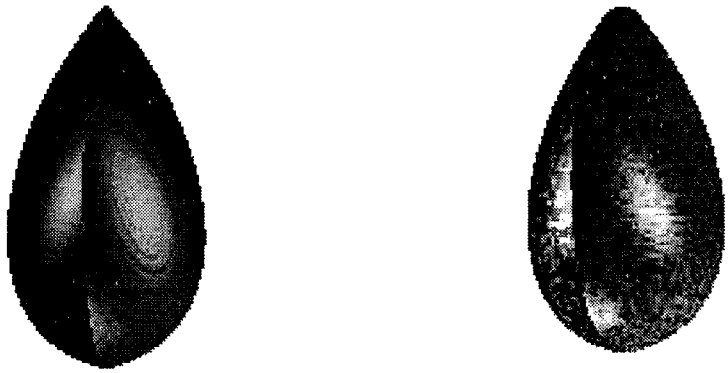


Figure 6.5 *Objet CSG : intersection de deux ellipsoïdes*

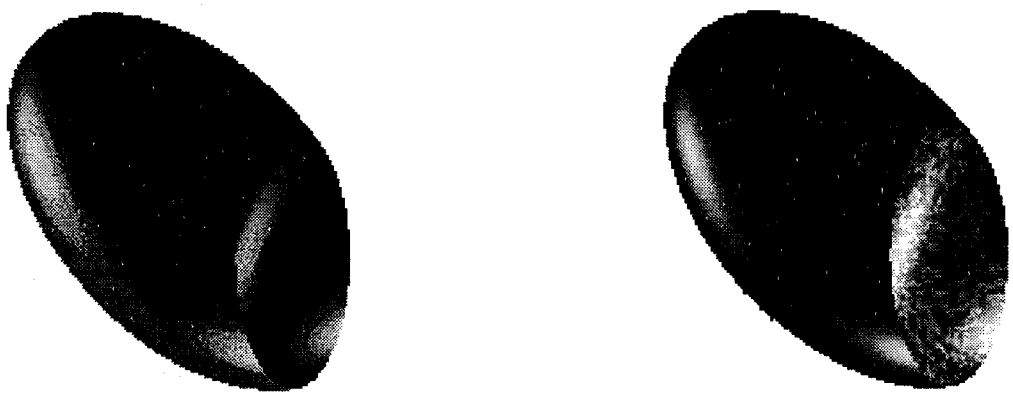


Figure 6.6 *Objet CSG : un ellipsoïde coupé par un autre*

6.4.3 Le placage de texture

La série de photos présentées maintenant illustre le paragraphe 3.5. En particulier on montre pourquoi nous avons séparé les objets en deux catégories, les objets à peu près plats et les objets sphéroïdes ainsi que l'adéquation respectivement du plan et du cube comme surfaces intermédiaires pour ces deux catégories d'objets.

Dans figure suivante un motif à damier est plaqué sur un paraboloïde hyperbolique (une selle de cheval). Sur la photo de gauche, la surface est considérée comme à peu près plate, la surface intermédiaire est un plan.

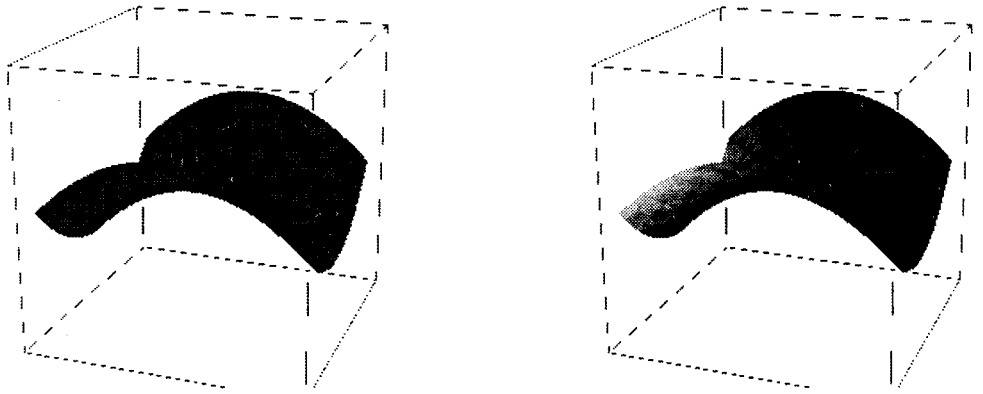


Figure 6.7 ***Placage de texture : choix de la surface intermédiaire (illustration 1)***

On remarque que pour une surface à peu près plate on a intérêt à utiliser une surface intermédiaire plane. En effet sur la deuxième image, pour laquelle le placage est fait avec un cube, on voit les discontinuités. De plus rappelons que le cube exige six fois plus de mémoire pour le stockage de la texture.

Dans la figure ci-dessous, on montre que pour un objet sphéroïde (en l'occurrence une sphère) il vaut mieux utiliser un cube (à gauche) comme surface intermédiaire qu'un plan (à droite). On note également qu'avec un motif régulier les discontinuités se voient.

Figure 6.8 ***Placage de texture : choix de la surface intermédiaire (illustration 2)***

Sur la figure suivante on montre que les discontinuités ne sont toujours gênantes. En effet pour une texture fortement bruitée, les discontinuités sont "noyées dans le bruit". Pour les deux photos la surface intermédiaire est le cube, puisque l'objet est sphéroïde. Les deux textures sont bruitées. Les discontinuités, notables sur la photo ci-dessus à gauche, disparaissent complètement.

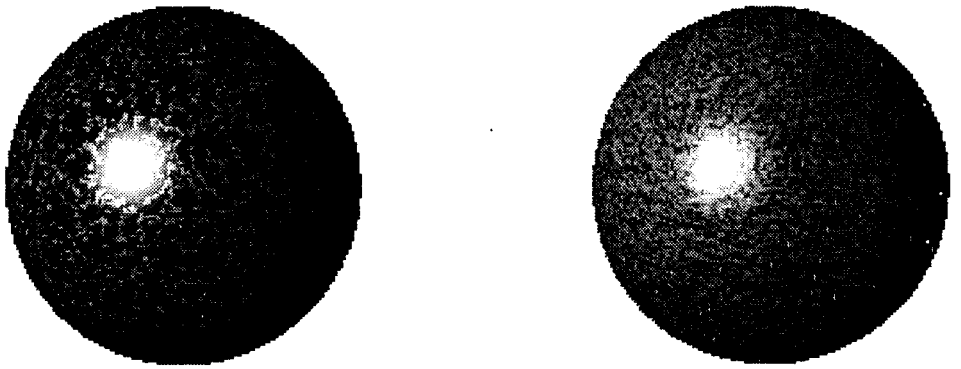


Figure 6.9 ***Placage de texture : problème de discontinuités***

Illustrons maintenant les problèmes de distorsions lorsque l'objet est plus ou moins plat. Pour les trois photos dela figure ci-dessous, la surface intermédiaire est plane. Au début on plaque sur une demi-sphère, on a donc beaucoup de distorsion. Ensuite on ne garde que la calotte supérieure d'un ellipsoïde de plus en plus plat. Sur la deuxième photo on note encore un peu de distorsion au niveau des yeux. Sur la dernière, presque plane, on n'a plus de distorsion notable.

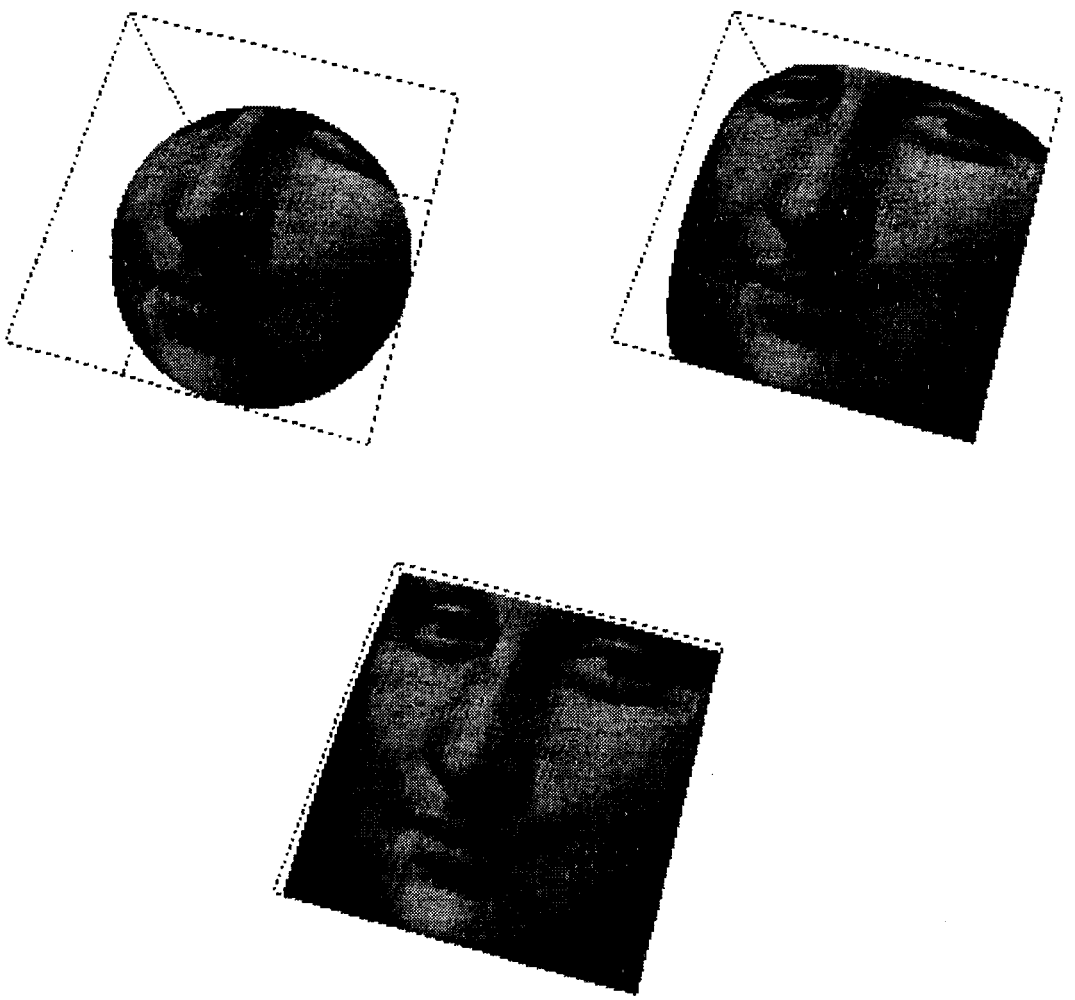


Figure 6.10 ***Placage de texture : problème de distorsion***

6.4.4 Divers

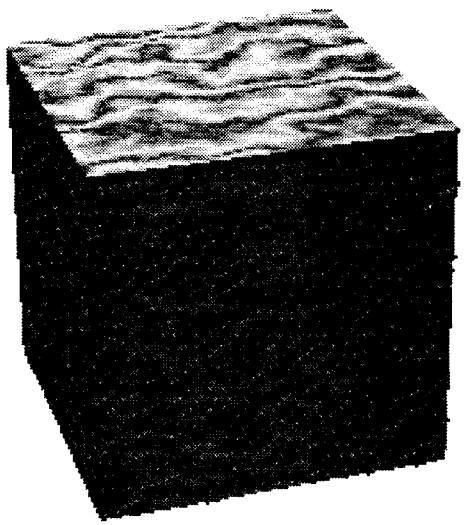


Figure 6.11 *Un cube "sculpté" dans le marbre*

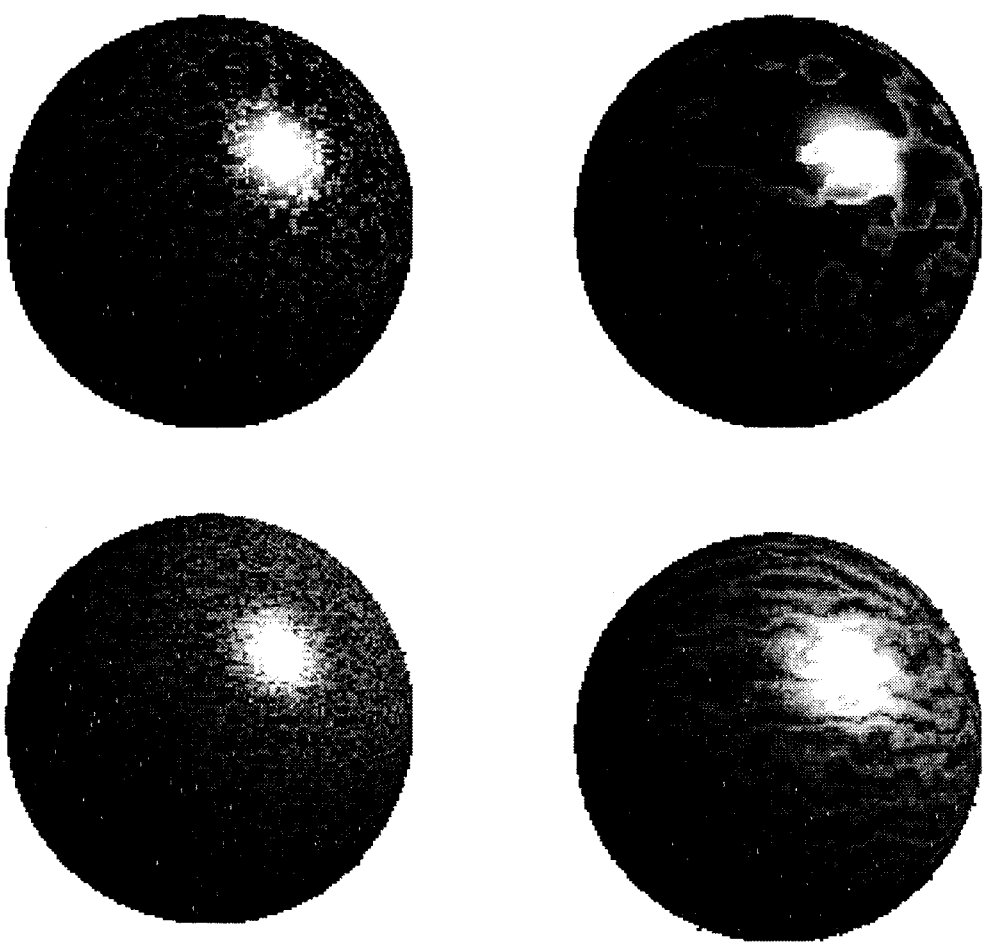


Figure 6.12 *De jolies planètes*

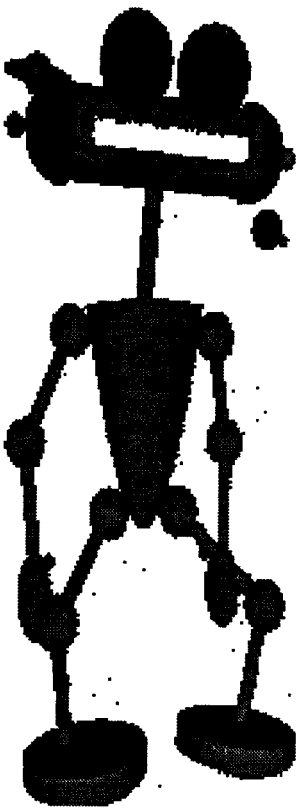


Figure 6.13 *Le robot Micro One*

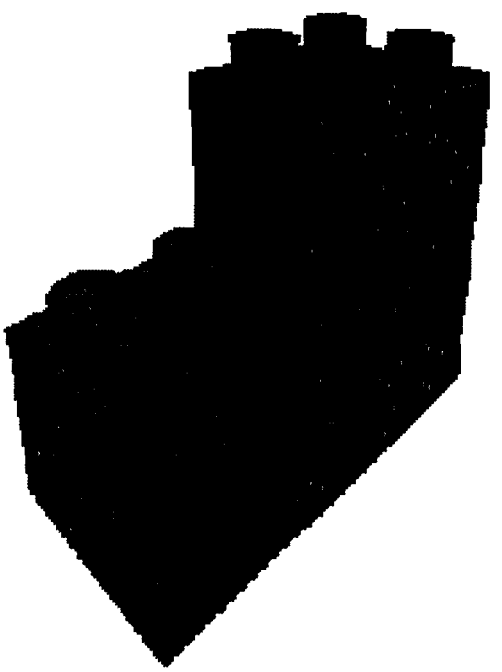


Figure 6.14 *Tout pour faire un simulateur de Lego*

6.5 Algorithme de sélection de profondeurs

Cet algorithme a été particulièrement délicat à mettre au point. Une description fonctionnelle en est faite au paragraphe 3.3.3. Voici le code en langage C, tiré du logiciel de validation et simulation qui a été développé. On retrouve bien les trois phases qui le compose. On remarque que le deuxième demi-objet n'est pris en compte que lorsque cela est nécessaire, ce qui ne pourrait être le cas dans le processeur quadrique. Cette optimisation logicielle n'a été faite qu'après s'être assuré que la prise en compte systématique du deuxième demi-objet (simulation de l'implémentation matérielle) fonctionnait correctement.

```
void lego_objet()
{
|   int zf1,zb1,vf1,vb1,zf2,zb2,vf2,vb2,nuf1=0,nub1=0,nuf2=0,nub2=0,i;
|
|   /*********************************************************/
|   /**** 1e phase : on decoupe en deux demi-objets ****/
|   /*********************************************************/
|   if ((ORIENT[0]==0)&&(F2>0))
|   {
|   |   zf1=ZMIN;vf1=BV[0];zb1=Zf;vb1=BV[0];
|   |   zf2=Zb;vf2=BV[0];zb2=ZMAX;vb2=BV[0];
|   |   /****
|   |   |   le deuxieme demi-objet NE SERT QUE dans ce cas, on effectue donc
|   |   |   la deuxieme phase (ie la construction iterative) immediatement
|   |   |   sur ce deuxieme demi-objet
|   |   ****/
|   |   for(i=1;i<=NBPA;i++)
|   |   {
|   |   |   if (ORIENT[i]==0) /**** plan perpendiculaire ****/
|   |   |   {
|   |   |   |   if (ZP[i] > 0) { zf2=ZMAX; zb2=ZMAX; vf2=0; vb2=0; }
|   |   |   }
|   |   |   else /**** plan non perpendiculaire ****/
|   |   |   {
|   |   |   |   if (ORIENT[i] < 0) /**** plan avant ****/
|   |   |   |   {
                        if (ZP[i]>zb2) { zf2=ZMAX; zb2=ZMAX; vf2=0; vb2=0; }
                        else if (ZP[i]>zf2) { zf2=ZP[i]; vf2=BV[i]; nuf2=i; }
|   |   |   |   }
|   |   |   |   else /**** plan arriere ****/
|   |   |   |   {
```

```
                                    if (ZP[i]<zf2) { zf2=ZMAX; zb2=ZMAX; vf2=0; vb2=0; }
                                    else if (ZP[i]<zb2) { zb2=ZP[i]; vb2=BV[i]; nub2=i; }
|   |   |   |   }
|   |   |   }
|   |   }
|   }
|   else if ((ORIENT[0]==0)&&(F2<0))
|   |   { zf1=ZMIN;vf1=BV[0];   zb1=ZMAX;vb1=BV[0];   vf2=vb2=ZMAX;  }
|   else if ((ORIENT[0]==1)&&(F2<0))
|   |   { zf1=ZMAX;vf1=0;       zb1=ZMAX;vb1=0;       vf2=vb2=ZMAX;  }
|   else if ((ORIENT[0]==1)&&(F2>0))
|   |   { zf1=Zf;vf1=BV[0];     zb1=Zb;vb1=BV[0];     vf2=vb2=ZMAX;  }

|   /*************************************************************************/
|   /**** 2e phase : construction iterative de l'objet plan par plan ****/
|   /*************************************************************************/
|   for(i=1;i<=NBPA;i++)
|   {
|   |   /**** pour le premier demi-objet ****/
|   |   if (ORIENT[i]==0) /**** plan perpendiculaire ****/
|   |   {
|   |   |   if (ZP[i] > 0) { zf1=ZMAX; zb1=ZMAX; vf1=0; vb1=0; }
|   |   }
|   |   else /**** plan non perpendiculaire ****/
|   |   {
|   |   |   if (ORIENT[i] < 0) /**** plan avant ****/
|   |   |   {
|   |   |   |   if (ZP[i]>zb1) { zf1=ZMAX; zb1=ZMAX; vf1=0; vb1=0; }
|   |   |   |   else if (ZP[i]>zf1) { zf1=ZP[i]; vf1=BV[i]; nuf1=i; }
|   |   |   }
|   |   |   else /**** plan arriere ****/
|   |   |   {
|   |   |   |   if (ZP[i]<zf1) { zf1=ZMAX; zb1=ZMAX; vf1=0; vb1=0; }
|   |   |   |   else if (ZP[i]<zb1) { zb1=ZP[i]; vb1=BV[i]; nub1=i; }
|   |   |   }
|   |   }
|   }
|
```

```
|       /*************************************/
|       /* 3e phase : on recupere le bon z */
|       /*************************************/
|       if (vf1==1) { Z1=zf1; Numero=nuf1; }
|       else if (vb1==1) { Z1=zb1; Numero=nub1; }
|       else if (vf2==1) { Z1=zf2; Numero=nuf2; }
|       else if (vb2==1) { Z1=zb2; Numero=nub2; }
|       else { Z1=ZMAX; }/* Rem : ici le numero de la primitive ne sert a rien */
|
}
```

Bibliographie

[Akeley88] K. Akeley et T. Jermoluk
High Performance Polygon Rendering
Computer Graphics, vol. 22, num. 4, pp. 239-246, August 1988

[Akeley93] K. Akeley
Reality Engine Graphics
Computer Graphics, Siggraph'93 proceedings, pp. 109-116, August 1993

[Atherton83] Peter R. Atherton
A Scan-line Hidden Surface Removal Procedure for Constructive Solid Geometry
Computer Graphics, vol. 17, num. 3, pp. 73-82, July 1983

[Bajaj92] C. Bajaj, I. Ihm
Algebraic Surface Design with Hermite Interpolation
ACM Transactions on Graphics, vol. 11, num. 1, January 1992, pp. 61-91

[Baumgart72] B. G. Baumgart
Winged-Edge Polyhedron Representation
Technical Report STAN-CS-320, Computer Science Department, Stanford University, Palo Alto, CA, 1972

[Bergeron86] Philippe Bergeron
A General Version of Crow's Shadow Volume
IEEE CG&A, September 1986, pp. 17-28

[Bézier70] Pierre Bézier
Emploi des Machines à Commande Numérique
Masson et Cie, Paris, 1970

[Bézier74] Pierre Bézier
Mathematical and Ptractical Possibilities of UNISURF
Barnhill & Riesenfeld Eds, Computer Aided Geometric Design, Academic Press, New York, 1974

[Bier86] Eric A. Bier, Kenneth R. Sloan
Two-Part Texture Mappings
IEEE CG&A, September 1986, pp. 40-53

[Bishop86] G. Bishop et D. M. Weimer
Fast Phong Shading
Computer Graphics, Siggraph'86, vol. 20, num. 4, pp. 103-106, August 1986

[Blinn78a] James F. Blinn
Simulation of Wrinckled Surfaces
Computer Graphics (Siggraph'78), vol. 12, num. 3, pp. 286-292, August 1978

[Blinn78b] James F. Blinn
Computer Display of Curved Surfaces
PhD Thesis, University of Utah, Department of Computer Science, December 1978

[Bloomenthal90] Jules Bloomenthal, Brian Wyvill
Interactive Techniques for Implicit Modeling
Computer Graphics, vol. 24, num. 2, pp.109-116, March 1990

[Boyse82] John W. Boyse, Jack E. Gilchrist, General Motors
GMSolid: Interactive Modeling for Design and Analysis of Solids
IEEE CG&A, March 1982, pp. 27-40

[Bronsvoort86] Willem F. Bronsvoort
Techniques for Reducing Boolean Evaluation Time in CSG Scan-line Algorithms
Computer Aided Design, vol. 18, num. 10, pp. 533-538, December 1986

[Catmull75] E. Catmull
Computer Display of Curved Surfaces
in Proceedings of IEEE Conference on Computer Graphics, Pattern Recognition and Data Structure, May 1975

[Chaillou91] Christophe Chaillou
Etude d'un Processeur de Visualisation d'Images de Synthèse en Temps Réel Exploitant un Parallélisme Massif Objet : le Projet IMOGENE
Thèse de Doctorat, LIFL, Univ. de Lille I (France), Janvier 1991

[Chang89] L. S. Chang, M. Shantz et R. Rocchetti
Rendering Cubic Curves and Surfaces with Integer Adaptative Forward Differencing
Computer Graphics, vol. 23, num. 3, pp. 157-166, July 1989

[Clark82] James Clark
The Geometry Engine: a VLSI Geometry System for Graphics
Computer Graphics, vol. 16, num. 3, pp. 127-133, July 1982

[Claussen89] U. Claussen
Reducing the Phong Shading Method
Proceedings of Eurographics'89, Elsevier Science Publisher, 1989

[Claussen91] U. Claussen
Real Time Phong Shading
Advances in Computer Graphics Hardware V, Springer Verlag, 1991

[Cohen85] M. Cohen et D. Greenberg
The Hemi-Cube, a Radiosity Solution for Complex Environments
ACM Computer Graphics vol. 19 num. 3, july 1985, pp 31-40

[Cowgill64] D. Cowgill
Logic Equations for a Built-in Root Method
IEEE Transactions on Electronic Computers, April 1964, pp. 156-157

[Crow77] Franklin C. Crow
Shadow Algorithms for Computer Graphics.
ACM Computer Graphics, vol. 11 num. 3, july 1977, pp 242-248

[Crow84] Franklin C. Crow
Summed-Area Table for Texture Mapping
Computer Graphics, Siggraph'84 Proceedings, vol. 19, pp. 207-212, July 1984

[Dahmen89] Wolfgang Dahmen
Smooth Piecewise Quadric Patches
chap. 23, pp. 181-193, Academic Press, 1989

[Deering88] M. Deering, S. Winner, B. Schediwy et al.
The Triangle Processor and Normal Vector Shader : A VLSI System for High Performance Graphics
ACM Computer Graphics, vol. 22, num. 4, pp. 21-30, August 1988

[Deering93] Michael F. Deering et Scott R. Nelson
Leo: A System for Cost Efective 3D Shaded Graphics
Computer Graphics, Siggraph'93 proceedings, pp. 101-108, August 1993

[Ellis91] J.L. Ellis, G. Kedem et al.
The RayCasting Engine and Ray Representations
Proceedings ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, june 1991, pp 255-267

[Epstein89] David A. Epstein, Frederik W. Jansen, Jaroslaw R. Rossignac
Z-Buffer Rendering from CSG : The Trickle Algorithm
RC 15182, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York 10598

[ESS89] European Silicon Structures
ECPD15 Process Databook
ES2 Publication Unit, November 1989

[Eyles88] J. Eyles, J. Austin, H. Fuchs, T. Greer, J. Poulton
Pixel-Planes 4 : A Summary
Advances in Computer Graphics Hardware II, Springer Verlag, pp. 183-208, 1988

[Farin92] Gerald Farin
Curves and Surfaces for CAGD
Academic Press, 1992, Third Edition

[Foley90] J. Foley, A. Van Dam, S. Feiner et J. Hughes
Computer Graphics: Principles and Practice (second edition)
The systems Programming Series, Addison Wesley 12110, 1990

[Froumentin94] Maxime Froumentin et Christophe Chaillou
Déformation Interactive de Carreaux Quadriques
Revue Internationale de CFAO et d'infographie, vol. 9, num. 6, pp. 753-765, 1994

[Froumentin96] Maxime Froumentin
Modélisation à l'aide de surfaces quadriques dans le cadre de la synthèse d'images
Thèse de Doctorat, LIFL, Univ. de Lille I (France), Juin 1996

[Fuchs85] H. Fuchs, J. Goldfeather, J. Hultquist et al.
Fast Spheres, Shadows, Textures, Transparencies and Image Enhancement in Pixel-Planes
ACM Computer Graphics, vol.19 num.3, july 1985, pp 111-120

[Fuchs89] H. Fuchs, J. Poulton, J. Eyles et al.
Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories.
ACM Computer Graphics, vol. 23 num. 3, july 1989, pp 79-88

[Gardner84] G. Y. Gardner
Simulation of Natural Scenes Using Textured Quadric Surfaces
Computer Graohics, vol. 18, num. 3, pp. 11-20

[Gardner85] G. Y. Gardner
Visual Simulation of Clouds
Computer Graohics, vol. 19, num. 3, pp. 297-303

[Glassner89] A. S. Glassner
An Introduction to Ray-Tracing
Academic Press, London, 1989

[Goldfeather86a] J. Goldfeather, H. Fuchs
Quadratic Surface Rendering on a Logic-Enhanced Frame-Buffered Memory System
IEEE CG&A, vol. 6, num. 1, pp. 48-59, January 1986

[Goldfeather86b] J. Goldfeather, J. Hultquist, H. Fuchs
Fast Constructive Geometry Display in the Pixel-Powers Graphics System
ACM Computer Graphics, vol. 20 num. 4, august 1986, pp 107-116

[Goldfeather89] Jack Goldfeather, Steve Molnar, Greg Turk and Henry Fuchs
Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning
IEEE CG&A, May 1989, pp.20-28

[Goldman83] Ronald N. Goldman, Control Data Corporation
Two Approaches to a Computer Model for Quadric Surfaces
IEEE CG&A, September 1983, pp.21-24

[Goral84] C. Goral, K. Torrance, D. Greenberg, B. Battaile
Modeling the Interaction of Light Between Diffuse Surfaces
ACM Computer Graphics, vol. 18 num. 3, july 1984, pp 213-222

[Gouraud71] H. Gouraud
Continuous Shading of Curved Surfaces
IEEE Transaction on Computers, vol. C-20 num. 6, june 1971, pp 623-629

[Greene86] Ned Greene
Environment Mapping and Other Applications of World Projections
IEEE CG&A, vol. 6, num. 11, pp. 21-29, November 1986

[Guo93] Baining Guo
Representation of Arbitrary Shapes Using Implicit Surfaces
The Visual Computer, vol. 9, pp. 267-277, 1993

[Haeberli90] P. Haeberli et K. Akeley
The Accumulation Buffer: Hardware Support for High-Quality Rendering
ACM Computer Graphics, vol. 24, num. 4, august 1990, pp. 309-318

[Harell93] Chandlee B. Harell et Farhad Fouladi
Graphics Rendering Achitecture for High Performance Desktop Workstation
Computer Graphics, Siggraph'93 proceedings, pp. 93-100, August 1993

[Hashemian90] R. Hashemian
Square Rooting Algorithms for integer and Floating Point Numbers
IEEE Transactions on Computer, vol. 39 num. 8, august 1990, pp 1025-1029

[Heckbert86] Paul S. Heckbert
Survey of Texture Mapping
IEEE CG&A, November 1986, pp. 56-67

[Heckbert91] Paul S. Heckbert, Henry P. Moreton
Interpolation for Polygon Texture Mapping and Shading
State of the Art in Computer Graphics, Visualization and Modeling, Rogers and Earnshaw Eds, Springer-Verlag, pp.101-111

[Hennessy90] John L. Hennessy, David A. Patterson
Computer Achitecture, A Quantitative Approach
Morgan Kaufman Publisher Inc., 1990

[Herbison82] D. Herbison-Evans
Real-Time Animation of Human Figure Drawings with Hidden Lines Omitted
IEEE CG&A, vol. 2, num. 9, pp. 27-33

[Jansen86a] Frederik W. Jansen
CSG Hidden Surface Algorithms for VLSI Hardware System
Advances in Computer Graphics Hardware I, pp. 75-82, 1986, Proceedings of the First Eurographics Workshop on Graphics Hardware,

[Jansen86b] Frederik W. Jansen
A Pixel-Parallel Hidden Surface Algorithm for Constructive Solid Geometry
Proceeding of the Eurographics'86 Conference, pp29-40, Elsevier Science Publisher

[Jansen90] Frederik W. Jansen, Arno N. T. van der Zalm
A Shadow Algorithm for CSG
Proceedings of Eurographics'90, pp. 51-61

[Jansen91] Frederik W. Jansen
Depth-Order Point Classification Techniques for CSG Display Algorithm
ACM Transactions on Graphics, vol. 10, num. 1, pp. 40-70, January 1991

[Karpf92] Sylvain Karpf, Christophe Chaillou
An Efficient Massively Parallel Rasterization Scheme for a High Performance Graphics System
Seventh Eurographics Wokshop on Graphics Hardware, Cambridge, Septembre 1992

[Karpf93] Sylvain Karpf
Architectures Massivement parallèles pour la Synthèse d'Images Temps Réel
Thèse de doctorat, LIFL, Université de Lille 1 (France), Janvier 1993

[Kaufman91] A. Kaufman
Volume Visualization
IEEE Computer Society Press, 1991

[Kedem89] G. Kedem et J.L. Ellis
The Ray Casting Machine.
Parallel Processing for Computer Vision and Display, Addison Wesley, pp 378-401

[Knittel93a] Günter Knittel
VERVE : Voxel Engine for Real-Time Visualisation and Examination
Eurographics'93 proceedings, Blackwell Publishers, pp. C37-C48, 1993

[Knittel93b] Günter Knittel
A VLSI Design for Fast Vector Generation
Eighth Eurographics Workshop on Graphics Hardware, pp.1-14, Septembre 1993

[Kleij93] Reiner van Kleij
Display of Solid Models with Quadratic Surfaces
PhD Thesis, 26 April 1993, Delft University of Techmology, The Netherlands

[Kuijk89] A. A. M. Kuijk et E. H. Blake
Faster Phong Shading via Angular Interpolation
Computer Graphics Forum, vol. 8, num. 4, pp.315-324, 1989

[Laporte91] 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

[Laporte93] H. Laporte, E. Nyiri, M. Froumentin et C. Chaillou
Direct Visualisation of Quadrics
Proceedings of Eighth Eurographics Workshop on Graphics Hardware, Barcelona, September 1993

[Laporte95a] H. Laporte, M. Froumentin et C. Chaillou
Quadric Surfaces for Real Time Visualisation
Proc. of First Eurographics Workshop on Implicit Surfaces, Grenoble, April 1995

[Laporte95b] H. Laporte, E. Nyiri, M. Froumentin et C. Chaillou
A Graphics System Based on Quadrics
Computers and Graphics, vol. 19, num. 2, pp. 252-260, March-April 1995

[Lefévère91] V. Lefévère, S. Karpf, C. Chaillou et M. Meriaux
The I.M.O.G.E.N.E Machine: Some Hardware Elements.
Sixth Eurographics Workshop on Graphics Hardware. Advances in Computer Graphics Hardware VI, Springer Verlag

[Lefévère92] V. Lefévère et C. Chaillou
Low Cost Hardware for Real Time Phong Lighting
Proceedings Graphics Interface 92 Workshop on Local Illumination, may 1992, pp 23-30

[Lefévère94] Vincent Lefévère
Architecture Spécialisée pour l'éclairement de Phong en Temps Réel
Thèse de doctorat, LIFL, Université de Lille I (France), Février 1994

[Levin76] Joshua Levin
A Parametric Algorithm for Drawing Pictures of Solid Composed of Quadric Surfaces
Communication of the ACM, vol. 19, num. 10, October 1976 pp. 555-563

[Loutrel70] P. Loutrel
A Solution to the Hidden-Line Problem for Computer-Drawn Polyhedra
IEEE Transaction on Computers, vol. c-19, 1970, p. 205

[Mahl72] Robert Mahl
Visible Surface Algorithms for Quadric Patches
IEEE Transactions on Computers, vol. c-21, num. 1, January 1972, pp.1-4

[Majithia72] J. C. Majithia
Cellular Array for Extraction of Squares and Square Roots of Binary Numbers
IEEE Transactions on Computers, September 1972, pp. 1023-1024

[Mammen89] Abraham Mammen
Transparency and Anti-Aliasing Algorithms Implemented with the Virtual Pixel Maps Technique
IEEE CG&A , July 1989, pp. 43-55

[Mantyla88] Martti Mantyla
An Introduction to Solid Modeling
Computer Science Press, 1988

[Max83] N. L. Max
Computer Representations of Molecular Surfaces
IEEE CG&A, vol. 3, num. 5, pp. 21-29

[Max90] N. L. Max
Cone-Spheres
Computer Graphics, Vol. 24, num. 4, pp. 59-62, August 1990

[Menon94] Jai P. Menon
Constructive Shell Representations for Freeform Surfaces and Solids
IEEE CG&A, March 1994, pp. 24-36

[Miller87] James R. Miller
Geometric Approches to Nonplanar Quadric Surface Intersection curves
ACM Transaction s on Graphics, October 1987, vol. 6, num. 4, pp. 274-307

[Molnar92] Steven Molnar, John Eyles et John Poulton
PixelFlow: High Speed Rendering Using Image Composition
Computer Graphics, Siggraph'92, vol. 26, num. 2, pp. 231-240, July 1992

[Ning93] Paul Ning et Jules Bloomenthal
An Evaluation of Implicit Surface Tilers
IEEE CG&A, November 1993, pp. 33-41

[Nishimura85] Hitoshi Nishimura, Makoto Hirai, Toshyuki Kawai, toru Kawata, Isao Shirakawa and Koishi Omura
Object Modeling by Distribution Function and a Method of Image Generation
Proceedings of Electronic Communication Conference'85 (in japanese)

[NVidia95a] NVidia Corporation
Informations commerciales

[NVidia95b] Joanna McMahon
Communication Personelle

[Nyiri90] E. Nyiri
Modélisation et Simulation d'Objets 3D a l'Aide d'Expressions du Second Degré
Mémoire de DEA, Université de Lille, septembre 1990

[Nyiri92a] E. Nyiri et C. Chaillou
Aspect Logiciel du Projet IMOGENE
Actes de MICAD 92, Editions Hermès, pp 201-217

[Nyiri92b] E. Nyiri, C. Chaillou et M. Froumentin
Le Polyèdre et la Quadrique comme primitives d'affichage
Actes des Journées GROPLAN 92, novembre 1992

[Nyiri94] Eric Nyiri
Etude de la Quadrique comme Primitive d'Afichage en Synthèse d'Images
Thèse de Doctorat, LIFL, Univ. de Lille I (France), Février 1994

[Peachey85] Darwyn R. Peachey
Solid Texturing of Complex Surfaces
Computer Graphics (Siggraph'85), vol. 19, num. 3, pp. 279-286, July 1985

[Perlin85] Ken Perlin
An Image Synthesizer
Computer Graphics (Siggraph'85), vol. 19, num. 3, pp. 287-296, July 1985

[Phong75] B.T. Phong
llumination for Computer Generated Pictures.
Communications of tht ACM, vol. 18 num. 18, june 1975, pp 311-317

[Porter78] T. K. Porter
Spherical Shading
Computer Graphics vol. 12, num. 3, pp. 282-285

[Pratt87] Vaughan Pratt
Direct Least-Squares Fittinf of Algebraic Surfaces
Computer Graphics (Siggraph '87 Proceedings) July 1987, pp. 145-152

[Preparata88] Franco P. Preparata, Michael Ian Shamos
Computational Geometry, an Introduction
Springer Verlag New Yprk Inc., Second Edition, 1988

[Preux95] Alain Preux, Christophe Chaillou
Anti-Aliassage en Synthèse d'Images
Techinque et Sscience Informatique, vol. 14, num. 10, pp. 1257-1290, 1195

[Requicha82] A. A. G. Requicha and H. B. Voelcker
Solid Modeling: A Historical Summary and Contemporary Assessment
IEEE CG&A, March 1982, pp.9-24

[Rogers90] David F. Rogers, J. Alan Adams
Mathematical Elements for Computer Graphics
Mac Graw-Hill Publishing Company, Second Edition, 1990

[Rossignac86] J. R. Rossignac, A. A. G. Requicha
Depth Buffering Display Techniques for Constructive Solid Geometry
IEEE CG&A, September 1986, pp. 29-39

[Rossignac89] J. R. Rossignac, H. B. Voelcker
Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection and Shading Algorithms
ACM Transactions on Graphics, vol. 8, num. 1, pp. 51-87, 1989

[Roth82] Scott D. Roth
Ray Casting for Solid Modeling
Computer Graphics and Image Processing, Vol. 18, pp. 109-144, 1982

[Sarraga83] Ramon F. Sarraga
Algebraic Methods for Intersections of Quadric Surfaces in GMSolid
Computer Vision, Graphics and Image Processing, Vol. 22, pp. 222-238

[Sato85] Hiroyuki Sato, Mitsuo Ishii, Keiji Sato, Morio Ikesaka
Fast Image Generation Of Constructive Solid Geometry Using a Cellular Array Processor
ACM Siggraph'85 Proceedings, vol. 19, num. 3, pp. 95-102, July 1985

[Schneider89] Beng Olaf Schneider et Ute Claussen
PROOF: an Architecture for Rendering in Object Space
Advances in Graphics Hardware, pp. 121-137, Springer Verlag, Berlin, 1989

[Sederberg85] Thomas W. Sederberg
Piecewise Algebraic Surface Patches
Computer Aided Geometric Design, vol. 2, pp. 53-59, 1985

[Segal92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, Paul Haeberli
Fast Shadow and Lighting Effects Using Texture Mapping
Computer Graphics, vol. 26, num. 2, pp. 249-252, July 1992

[SGI92] Silicon Graphics.
RealityEngine in Visual Simulation
Technical Report, 1992.

[Sillion94] François Sillion et Claude Puech
Radiosity & Global Illumination
Morgan Kaufmann Publishers, Inc., San Francisco, California, 1994

[Sims96] Dave Sims
Putting the Visible Human to Work
IEEE CG&A, January 1996, pp. 14-15

[Stolte95] Nilo Stolte et René Caubet
Discrete Ray-Tracing of Huge Voxel Spaces
Proc. of Eurographics'95, Maastricht (The Netherlands), vol. 14, num. 3, pp. 383-394

[Sutherland74] I. E. Sutherland, R. F. Sproull, R. A. Schumacker
A Characterization of Ten Hidden-Surface Algorithms
ACM Computing Surveys, Vol. 6, num. 1, pp. 1-55, March 74

[Tilove80] Robert Bruce Tilove
Set Membership Classification : A Unified Approach to Geometric Intersection Problems
IEEE Transactions on Computers, Vol. C-29, num. 10, October 1980

[Watkins70] G. S. Watkins
A Real-Time Visible Surface Algorithm
Ph. D. Dissertation, Division of Computer Science, Univ. of Utah, Salt Lake City, Technical Report UTEC-CSc-70-101, July 1970

[Weiss66] Ruth A. Weiss
BE VISION, A Package of IBM 7090 FORTRAN Programs to Draw Orthographic Views of Combinations of Plane and Quadric Surfaces
Journal of the ACM, vol.13, num. 2, April 1966, pp.194-204

[Whitted80] T. Whitted
An Improved Illumination Model for Shaded Display
Communication ACM, vol. 23, 1980, pp 343-349

[Williams78] L. Williams
Casting Curved Shadows on Curved Surfaces
ACM Computer Graphics, vol. 12 num. 3, july 1978, pp. 270-274

[Williams83] Lance Williams
Pyramidal Parametrics
Computer Graphics, Siggraph'83 proceedings, vol.17, pp. 1-11, July 1983

[Woo90] Andrew Woo, Pierre Poulin, Alain Fournier
A Survey of Shadow Algorithms
IEEE CG&A, November 1990, pp.13-32

[Woon71] Peter Woon and H. Freeman
A procedure for Generating Visible-Line Projections of Solids Bounded by Quadric Surfaces
Information Processing'71, vol. 2, North-Holland Publication Company, Amsterdam, 1971, pp. 1120-1125