NUMERO D'ORDRE : 1282

ANNEE : 1994

Laboratoire d'Informatique Fondamentale de Lille

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

Vincent LEFEVERE

Architectures spécialisées pour l'éclairement de Phong en temps réel

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

Président :	V. CORDONNIER	USTL, Villeneuve d'asq
Directeur de Thèse :	M. MERIAUX	EUDIL, USTL, Villeneuve d'asq
Rapporteurs :	R. CAUBET	IRIT, Toulouse
	A. FOURNIER	UBC, Vancouver Canada
Examinateurs :	C. CHAILLOU	ENIC, LIFL, Villeneuve d'asq
	P. MATHERAT	E.N.S. Paris

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

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M. H. LEFEBVRE, M. PARREAU

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

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

PROFESSEUR EMERITE

M. A. LEBRUN

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

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

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

M. P. LOUIS

PROFESSEURS - CLASSE EXCEPTIONNELLE

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

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

PROFESSEURS - 1ère CLASSE

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

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

PROFESSEURS - 2ème CLASSE

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

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

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

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

Remerciements

Je tiens à adresser mes remerciements à :

- - Monsieur le Professeur Vincent CORDONNIER qui me fait l'honneur de présider ce jury,
- - Monsieur Michel MERIAUX, Professeur à l'EUDIL, qui m'a proposé ce sujet de thèse, extrêmement intéressant, et qui m'a soutenu tout au long de la rédaction de mon mémoire,
- - Messieurs René CAUBET, Professeur à l'IRIT de Toulouse et Alain FOURNIER, Professeur à l'UBC de VANCOUVER qui ont accepté d'être les rapporteurs de ce travail,
- - Monsieur Philippe MATHERAT, Professeur à l'ENS de PARIS, ainsi que Monsieur Christophe CHAILLOU, Maître de Conférences à l'ENIC, pour leur participation au jury.

Je remercie aussi :

- - Samuel DEGRANDE pour ses conseils, sa gentillesse, sa disponibilité pour échanger des idées,
- - Bernard SZELAG et ses collègues qui gèrent nos stations de travail et sauvegardent nos précieux fichiers,
- - Monsieur Henri GLANC pour sa contribution à la reproduction de ce mémoire.

CHAPITRE I: Introduction à l'infographie. 7

1•1 Les méthodes de synthèse d'images. 7
1•1•1 Le rendu par projection ou rendu géométrique. 8
1•1•2 Le lancer de rayons. 10
1•1•3 La synthèse d'images par méthode globale ou radiosité. 10
1•1•4 Le rendu volumique ou rendu voxel. 11
1•2 Les formules d'éclairement. 11
1•2•1 Les lois physiques élémentaires. 11
1•2•2 Les modèles de couleurs. 12
1•2•3 Le rayonnement spéculaire. 14
1•3 Les méthodes d'ombrage de facettes. 16
1•3•1 La méthode d'ombrage de Gouraud. 16
1•3•2 La méthode d'ombrage de Phong. 19
1•3•3 Les techniques d'accélération de l'ombrage de Phong. 20
1•3•4 Comparaison critique des méthodes de Phong et de Gouraud. 21
1•4 L'ombrage de quadriques. 21
1•5 Les textures. 22
1•5•1 Le "mappage" de textures 2D. 22
1•5•2 La perturbation de normales. 23
1•5•3 Les textures 3D. 23
1•6 Les architectures dédiées. 23
1•6•1 Les architectures à parallélisme pixels. 23
1•6•2 Les systèmes à parallélisme objet. 26
1•7 Cahier des charges. 30
1•7•1 Poser le problème. 30
1•7•2 L'intérêt des méthodes de Phong. 31
1•7•3 Fixer les contraintes. 31
1•7•4 Les points devant être examinés. 31

CHAPITRE II: La complexité de l'éclairement de Phong et son optimisation. 33

2•1 Scène en projection orthogonale. 34
2•1•1 Rappel sur la projection orthogonale. 34
2•1•2 Avec sources de lumière à l'infinie. 34
2•1•3 Avec sources de lumière à distance finie. 37
2•2 Scène en projection perspective. 40
2•2•1 Rappel sur la projection perspective. 40
2•2•2 Avec sources de lumière à l'infini. 42
2•2•3 Avec sources de lumière à distance finie. 47
2•3 Synthèse et conclusion. 51
2•3•1 La puissance requise. 51
2•3•2 Les opérations et fonctions souhaitées. 52

CHAPITRE III: Proposition d'implémentation matérielle. 53

3•1 Rappel de l'architecture générale. 54
3•2 L'Unité de Normalisation. 54
3•2•1 Nécessité de l'opération de Normalisation. 54
3•2•2 Le coût de la Normalisation. 56
3•2•3 Architecture de l'unité de normalisation. 56
3•2•4 Conclusion 58

3•3	**La composante spéculaire.**	**58**
3•3•1	Les solutions de base.	58
3•3•2	Proposition d'une solution plus simple.	60
3•3•3	Reculer les limites.	65
3•3•4	Des méthodes d'approximation.	69
3•3•5	Intégration des deux méthodes.	74
3•4	**La première version du processeur d'éclairement.**	**77**
3•4•1	L'architecture des unités de calcul des produits scalaires.	78
3•4•2	Détail de la structure de l'unité d'éclairement.	80
3•4•3	Conclusion.	81
	CHAPITRE IV: Développements mathématiques pour l'architecture.	**83**
4•1	**La transmission théorique de l'erreur.**	**83**
4•1•1	Au cours de la normalisation d'un vecteur.	84
4•1•2	La transmission de l'erreur dans le produit scalaire.	85
4•1•3	Au cours du calcul du diffus.	87
4•1•4	Au cours du calcul du spéculaire par le vecteur réfléchi.	87
4•1•5	Au cours du calcul du spéculaire par le vecteur réfléchi inverse.	89
4•1•6	Au cours du calcul du spéculaire par le vecteur surbrillance.	90
4•2	**L'Unité de Pré-Normalisation.**	**91**
4•2•1	Sa nécessité.	91
4•2•2	Définition formelle.	92
4•2•3	Transmission théorique de l'erreur dans l'unité de pré-normalisation.	93
4•3	**Les formats de données dans l'unité de normalisation.**	**93**
4•3•1	Le choix de la précision d'entrée de l'unité de normalisation.	93
4•3•2	Les simplifications induites par la pré-normalisation.	97
4•3•3	Etude théorique des formats de l'unité de normalisation de base.	98
4•3•4	Etude théorique des formats de l'unité de normalisation en S./L.N.S.	103
4•4	**Conclusion.**	**106**
	CHAPITRE V: Application au processeur de l'unité d'éclairement proposée.	**107**
5•1	**L'architecture générale.**	**107**
5•2	**L'architecture de la pré-normalisation.**	**108**
5•2•1	Algorithme et fonctionnement de la pré-normalisation.	108
5•2•2	L'architecture d'une unité de pré-normalisation.	109
5•2•3	La place de l'unité de pré-normalisation dans le pipeline de rendu.	110
5•3	**L'architecture de la normalisation.**	**112**
5•3•1	Coût de l'unité de normalisation de base en binaire naturel.	112
5•3•2	Coût de l'unité de normalisation en S./L.N.S.	115
5•3•3	Simulation des unités de normalisation.	117
5•3•4	Un exemple de solutions technologiques pour un VLSI.	118
5•3•5	L'utilisation de processeurs de traitement de signal.	120
5•4	**Le processeur d'éclairement.**	**120**
5•4•1	La version pour le binaire naturel.	121
5•4•2	La version pour la notation S./L.N.S.	122
5•4•3	Le chargement des RAMs.	123
5•4•4	Synthèse d'un V.L.S.I pour l'automate de chargement.	124

5•5 Bilan Global. 125

Conclusion. 127

ANNEXE A: L'optimisation des algorithmes d'éclairement. 129

A•1 Légende. 129
A•1•1 Les fonctions. 129
A•1•2 Les variables. 129
A•1•3 Les colonnes. 131
A•2 Scène en projection orthogonale. 131
A•2•1 Une source placée à l'infini. 131
A•2•2 Plusieurs sources placées à l'infini. 132
A•2•3 Une source placée à distance finie. 132
A•2•4 Plusieurs sources placées à distance finie. 133
A•3 Scène en projection perspective. 135
A•3•1 Une source placée à l'infini. 135
A•3•2 Plusieurs sources placées à l'infini. 137
A•3•3 Une source placée à distance finie. 139
A•3•4 Plusieurs sources placées à distance finie. 141

ANNEXE B: Les méthodes d'approximation de l'éclairement spéculaire. 143

B•1 Les approximations polynomiales. 143
B•1•1 Linéaire par morceaux. 143
B•1•2 Quadratique par morceau. 144
B•2 Les approximations rationnelles. 145
B•2•1 Fonction homographique. 145
B•2•2 Fonction rationnelle de degré deux. 147
B•3 Utilisation de fonction rationnelle par morceaux. 150
B•3•1 Fonction homographique. 150
B•3•2 Fonction rationnelle de degré deux. 150

ANNEXE C: Algorithmes de chargement pour l'éclairement spéculaire . 153

C•1 Elaboration du premier algorithme de chargement. 153
C•2 Etude du chargement d'une fonction homographique. 155
C•3 Etude du chargement d'une fonction rationnelle. 157
C•4 Application à l'approximation homographique. 159
C•5 Application à l'approximation rationnelle. 160

ANNEXE D: Les configurations de l'unité de normalisation. 161

D•1 Architecture avec unités séparées. 161
D•2 Architecture avec unité d'inversion de la racine carrée. 162
D•3 Architecture avec unités de division. 164

Introduction

La synthèse d'image par ordinateur est de plus en plus utilisée dans divers domaines depuis l'animation moléculaire jusqu'à la création de spots télévisés. Le besoin grandissant, il devient de plus en plus important d'obtenir rapidement des images de qualité. On trouve toute une série d'applications, par exemple les simulateurs, qui nécessitent le calcul d'images de synthèse en temps réel. En produisant des images de qualité sommaire, il est possible de nos jours de satisfaire assez facilement à de telles exigences. Cela s'avère plus difficile quand on désire utiliser le modèle de Phong pour calculer l'éclairement d'une scène complexe.

L'objet de cette thèse est donc de concevoir et d'étudier la faisabilité d'un processeur d'éclairement en temps réel intégrant l'éclairement de Phong.

Dans le premier chapitre, nous évoquons les techniques existantes pour créer des images de synthèse. Le temps réel faisant partie de nos objectifs principaux, nous limitons, par la suite, notre étude à la plus simple d'entre elles: la méthode de rendu par projection. Nous présenterons ensuite les formules d'éclairement. Cela inclut celles établies de façon empirique pour permettre la représentation des surfaces brillantes. Puis nous évoquons les modèles d'ombrage permettant le calcul de l'éclairement sur les facettes ainsi que sur d'autres primitives plus complexes. L'intérêt de différer le calcul de l'éclairement après la conversion des objets en pixels sera mis en évidence. Nous concluons notre introduction de l'infographie en examinant les architectures dédiées, et plus particulièrement le projet I.M.O.G.E.N.E., support de notre unité d'éclairement de Phong en temps réel dont nous précisons finalement le cahier des charges.

Au chapitre II, en vue d'une réalisation, nous estimons, en nombre d'opérations, le coût du calcul de l'éclairement. Nous examinerons, tour à tour, chacun des cas selon la position de la source, la position de l'observateur et la formule d'éclairement spéculaire utilisée. Une optimisation est, à chaque fois, proposée pour une ou plusieurs sources. Cette première étude indique ainsi la puissance de calcul requise et montre combien elle est importante. Afin d'aboutir à une proposition d'implémentation matérielle, nous limitons donc, par la suite, notre étude en plaçant l'observateur et les sources à l'infini.

Nous consacrons tout le chapitre III, à l'introduction d'une architecture générale de l'unité de post-éclairement. Elle permet l'obtention de la puissance requise pour un nombre de sources modulables. C'est d'abord l'unité de normalisation qui retiend notre attention. Elle donne lieu à deux architectures, l'une utilisant le calcul en binaire naturel et l'autre, le calcul en S./L.N.S. Puis nous nous intéressons au problème central du calcul de l'éclairement spéculaire: l'élévation à la puissance. Nous étudions les techniques et les restrictions acceptables des performances permettant d'obtenir l'intensité spéculaire pour un faible coût. Afin de ne pas rester prisonnier des limites de notre solution, nous examinons les possibilités d'accroître pour un coût dérisoire, les performances, de façon significative. Il résulte de cette étude la présentation d'un module optionnel permettant l'utilisation, par notre architecture, d'exposants spéculaires plus élevés. Nous orientons, ensuite, l'examen du calcul de l'éclairement spéculaire vers l'étude de méthodes d'approximation. Nous terminons ce chapitre en proposant des solutions permettant de tirer avantage des deux approches du problème.

Au cours de cette présentation, un point reste malgré tout ignoré: la taille des données qui sont manipulées au cours des calculs. Ce point est important et peut conditionner notre choix. Le chapitre IV est donc consacré à l'étude mathématique de ce problème. Nous commençons par établir les formules qui régissent la transmission de l'erreur au cours des calculs, puis l'erreur maximale admissible afin de s'assurer une qualité suffisante de l'image. Nous examinons, par la suite, la transmission théorique de l'erreur dans une unité idéale de normalisation, et montrons l'utilité d'introduire une unité de pré-normalisation. Nous terminons en calculant le cumul des erreurs introduites successivement, au cours de la normalisation par chaque bloc composant l'unité. Ainsi nous établissons un compromis sur la taille de chaque bloc permettant de diminuer globalement la complexité de l'architecture.

Nous terminons ce travail, en décrivant, au chapitre V, l'unité de pré-normalisation puis nous chiffrons la complexité de chacune des deux architectures de l'unité de normalisation. Nous présentons, alors, les résultats de leurs simulations fonctionnelles. Ensuite, nous examinons un exemple de solution technologique pour la réalisation matérielle dans un VLSI. Nous conclurons en détaillant les algorithmes câblés dans chacun des blocs de l'unité d'éclairement.

CHAPITRE I: Introduction à l'infographie.

Pour créer des images de synthèse, il existe plusieurs techniques que nous évoquerons successivement au début de ce chapitre. Le temps réel faisant partie de nos objectifs principaux, nous limiterons par la suite notre étude à la plus simple d'entre elles: la méthode de rendu par projection.

Notre but étant l'étude d'une unité d'éclairement en vue de sa réalisation, nous présenterons ensuite les formules d'éclairement. Cela recouvrira l'examen des lois physiques régissant la propagation de la lumière, mais aussi la décomposition spectrale de la lumière et les modèles de couleurs les plus couramment utilisés en infographie. Les formules empiriques permettant la représentation des surfaces brillantes seront à leur tour exposées.

Les bases de données étant habituellement converties en facettes pour faciliter la synthèse d'images, nous évoquerons les différents modèles d'ombrage permettant le calcul de l'éclairement sur les facettes. Nous envisagerons ensuite le cas où d'autres primitives plus complexes sont utilisées. Nous terminerons la présentation des techniques de l'infographie en évoquant la possibilité d'ajouter des textures aux objets.

Puis, nous nous intéresserons aux architectures dédiées, en évoquant tour à tour les différents types de parallélisme. Nous examinerons d'abord plusieurs propositions utilisant le parallélisme pixel, puis celles utilisant le parallélisme objet. Dans cette dernière catégorie, nous examinerons en détail le projet I.M.O.G.E.N.E. Enfin nous préciserons le cahier des charges de notre unité d'éclairement de Phong en temps réel.

1•1 Les méthodes de synthèse d'images.

La synthèse d'images utilise de nos jours diverses méthodes qui diffèrent tant par la complexité des algorithmes utilisés que par la qualité d'image obtenue. On dispose principalement de quatre méthodes :

- le rendu par projection appelé aussi rendu géométrique; c'est le plus simple de tous.
- le lancer de rayons, qui ajoute le réalisme des miroirs.
- la synthèse d'images par méthode globale qui procède par l'étude des échanges de flux lumineux.
- le rendu volumique.

Plus la complexité des méthodes augmente, plus les images sont de qualité, et plus la notion de temps réel devient hors d'atteinte.

1•1•1 Le rendu par projection ou rendu géométrique.

C'est la méthode la plus simple et la plus utilisée. Elle peut facilement être parallélisée et constitue la seule méthode permettant d'obtenir le temps réel en admettant certaines restrictions concernant la taille de la base de données et la taille de l'écran. Certaines stations de travail haut de gamme disposent d'unités spécialisées pour la synthèse d'images par rendu géométrique.

1•1•1•1 Conversion Objet-Pixel.

• Cette méthode de rendu consiste simplement à déterminer la couleur de chaque élément de l'écran (ou pixel) en calculant quel objet est vu par l'observateur par la petite fenêtre que constitue le pixel. Il faut donc convertir la base de données des objets en pixels dans lesquels on obtiendra la couleur. Cette méthode fait intervenir deux opérations étroitement liées que l'on peut dissocier sous les appellations "la conversion des objets en pixels" et "l'élimination des parties cachées".

• La conversion des objets en pixels consiste simplement à calculer les paramètres de profondeurs, de couleurs ou de normales relatives à l'objet pour chaque pixel par lequel on pourrait voir un élément de l'objet sans tenir compte des autres éléments. C'est une opération que l'on peut donc aisément effectuer en parallèle sur plusieurs objets différents pour un même pixel ou inversement. La parallélisation de la conversion objet en pixel introduit un choix entre un parallélisme pixel et un parallélisme objet [36] [37]. Par exemple, pour les projets GSP-NVS [13], PROOF [50] et I.M.O.G.E.N.E [7] la solution du parallélisme objet a été choisie, par contre les projets Pixel-Plane 4 [15], Pixel-Plane 5 [18] ont opté pour le parallélisme pixel. Karpf [28] a étudié les performances que l'on pouvait obtenir suivant la méthode choisie quand on l'utilise sur des portions réduites de l'écran. Sa conclusion est en faveur du parallélisme pixel qui selon lui permet d'obtenir le meilleur rapport coût-performance avec sa solution multi-pipeline.

• En un même pixel pourraient être calculées plusieurs valeurs relatives à des objets différents. L'élimination des parties cachées va consister à conserver uniquement la valeur appartenant à l'objet visible, c'est à dire appartenant à l'objet situé à la distance la plus proche de l'observateur.

1•1•1•2 L'élimination des parties cachées.

• Cette opération a historiquement été le sujet de grandes discussions, différents algorithmes, que l'on peut retrouver dans le livre "Algorithmes pour l'infographie" [47], étant proposés. Dorénavant, la discussion semble close, la méthode du tampon en profondeur (ou Z-buffer) est la plus utilisée, en particulier dans les architectures spécialisées pour la synthèse d'images et dans les projets précédemment cités.

• La méthode consiste à utiliser un tampon dans lequel on conserve pour chaque pixel la couleur ou tout autre paramètre permettant d'obtenir la couleur (comme les normales quand on utilise l'éclairement de Phong (voir paragraphe 1•3•2) et la profondeur Z).

Pour chaque pixel de chaque objet, on vérifie si la profondeur calculée est inférieure à la profondeur précédemment stockée dans le tampon. Si c'est le cas, les nouvelles valeurs sont substituées aux anciennes. Dans le cas contraire, le contenu du tampon reste inchangé.

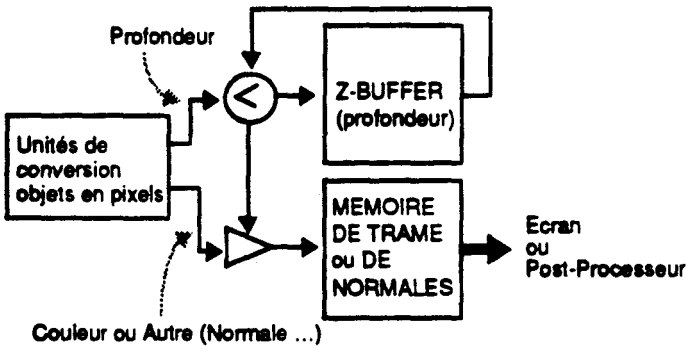


Figure 1•1 L'architecture du tampon en profondeur.

1•1•1•3 Le cas de la projection perspective.

• La projection orthogonale manque parfois de réalisme car il n'est pas toujours possible de placer l'observateur suffisamment loin de la scène pour qu'il puisse être considéré comme étant à l'infini. Dès lors que l'observateur est relativement proche des objets, il faut effectuer une projection en perspective.

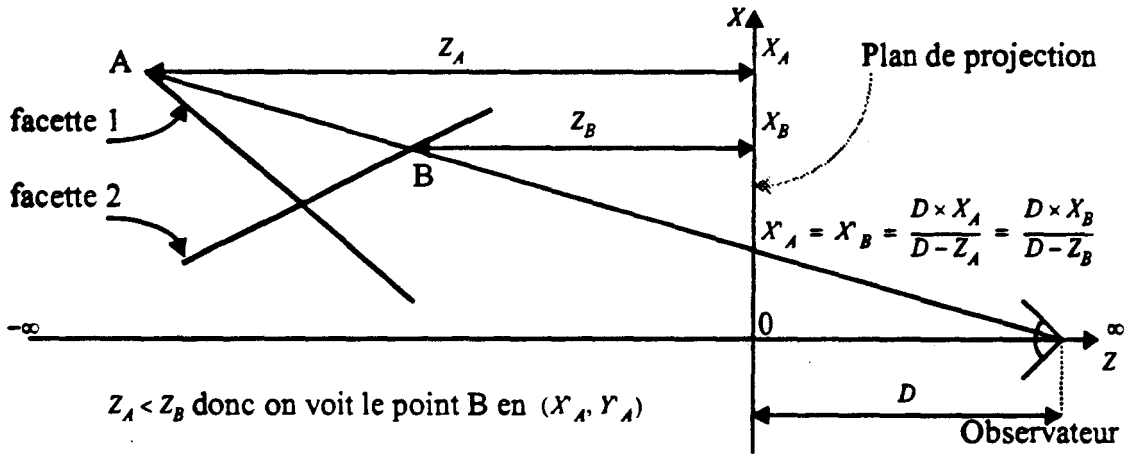


Figure 1•2 L'élimination des parties cachées pour des scènes en perspective.

L'une des solutions peut consister à calculer les coordonnées (X', Y') à partir des coordonnées (X, Y) puis d'effectuer l'élimination des parties cachées en utilisant le tampon en profondeur avec les coordonnées (X', Y') sur le plan de projection et en comparant les vraies profondeurs. Pour chaque élément de surface des objets, de coordonnées (X, Y, Z), il faut déterminer les coordonnées du pixel (X', Y') en calculant $\frac{D}{D-Z}$ et en multipliant X et Y par cette valeur. C'est un calcul qui coûte cher.

• Mais, la solution qui s'adapte le mieux à l'utilisation d'un tampon en profondeur pour l'élimination des parties cachées est la transformation perspective. Cela consiste à transformer l'espace réel de la scène vers un espace virtuel dans lequel l'application correspondant à la projection perspective est une projection orthogonale. Ainsi on peut effectuer aisément l'élimination des parties cachées en utilisant le tampon en profondeur dans l'espace virtuel.

La transformation perspective est définie comme ceci : $Tp : \begin{cases} \Re^3 \to \Re^3 \\ \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \to \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} \end{cases}$

avec $X' = (D \times X) / (D - Z)$
$Y' = (D \times Y) / (D - Z)$ et D la distance entre l'écran de projection et l'observateur.
$Z' = (D \times Z) / (D - Z)$

• L'image d'une facette triangulaire par la transformation Tp est aussi une facette triangulaire. Pour effectuer la conversion des facettes en pixels, dans l'espace virtuel de la transformation perspective, il suffira donc d'effectuer la conversion des facettes images obtenues par transformation des coefficients de la base de données des facettes.

Soit une facette triangulaire (A, B, C), son image dans l'espace virtuel est la facette $(Tp(A), Tp(B), Tp(C))$. Pour visualiser une facette triangulaire en perspective, il suffira donc de calculer les nouvelles coordonnées des trois sommets de la facette après transformation. On réduit donc les calculs propres à la mise en perspective au nombre de trois par facette au lieu d'un par élément de surface des facettes se projetant sur un pixel.

1•1•2 Le lancer de rayons.

• Cette méthode [57] est basée sur la simulation du trajet suivi par les rayons lumineux jusqu'à l'oeil de l'observateur. Chaque rayon atteignant l'oeil passe par un pixel de l'écran. Le principe consiste donc à lancer depuis l'observateur vers chaque pixel de l'écran un rayon primaire qui va permettre de parcourir le trajet inverse suivi par la lumière jusqu'aux sources.

On détermine le point d'intersection de chaque rayon primaire avec le premier objet rencontré (appartenant à la base de données). Issus de ces points, de nouveaux rayons vont être calculés. Premièrement, un rayon sera envoyé vers chaque source afin de déterminer s'il existe un obstacle entre la source et ce point et donc de savoir si cette source contribue, ou non, à l'éclairement de l'objet. On crée ainsi les ombres portées. Deuxièmement, un rayon réfléchi est émis dans la direction symétrique de celle du rayon primaire par rapport à la direction de la normale de l'objet au point considéré. On modélise ainsi les objets de type miroir. Troisièmement un rayon transmis est émis si l'objet est transparent en prenant en compte les indices des milieux et en utilisant la loi de Descartes.

La méthode consiste ensuite à renouveler l'opération effectuée sur les rayons primaires, avec les rayons réfléchis et transmis. On obtient alors un arbre de rayons dont on va limiter arbitrairement la profondeur.

• Cela permet d'obtenir des images de qualité. Mais, malgré les différents travaux de recherche effectués pour accélérer l'algorithme et le paralléliser, cela reste une méthode très gourmande en temps de calcul. Elle ne peut être utilisée pour la synthèse d'images en temps réel. Les diverses stratégies pour accélérer le processus sont proposées dans [21], [33] et [43].

1•1•3 La synthèse d'images par méthode globale ou radiosité.

La radiosité, qui a initialement été proposée par Goral et al. [23], calcule les images en procédant à l'étude des interactions lumineuses entre les différents objets composant la scène. Au lieu de considérer les échanges lumineux comme une propagation sous la forme de rayon, on modélise des échanges de flux lumineux entre des facettes.

La première opération va donc consister à représenter la base de données des objets comme un ensemble de facettes. Ensuite vient le calcul des interactions lumineuses et enfin on positionne l'observateur dans la scène puis on projette les facettes et leur radiosité (c'est à dire l'intensité du flux reçu par la facette) sur l'écran de visualisation lié à l'observateur. On utilise alors une projection perspective. (Voir paragraphe 1•1•1•3.)

Le calcul des radiosités (des interactions lumineuses), qui revient en fait à l'évaluation d'une intégrale, réclame la quantité de calculs la plus importante. La première implémentation efficace de ce calcul a été proposée sous l'appellation "méthode de l'hémicube" par Cohen et al en 1985 [11]. Depuis, de nombreux travaux ont abouti à diverses propositions permettant d'optimiser le temps de calcul [46] ou d'y adjoindre un éclairement spéculaire [51].

1•1•4 Le rendu volumique ou rendu voxel.

• Le rendu voxel, qui découle principalement de l'imagerie médicale (scanner et RMN), utilise une représentation de l'image sous forme d'un ensemble de points repérés par leurs coordonnées (x, y, z). Les points représentent des éléments de volume de l'espace et sont appelés ainsi des "voxels". Ils sont une extension 3D de la notion de "pixel". Un aperçu des principaux travaux de recherche effectués dans ce domaine est présenté dans [30]. Cette méthode fait appel à des éléments de géométrie discrète lors de la transformation des objets en voxels dont on peut avoir un aperçu en consultant [1], [14], [38] et [54].

1•2 Les formules d'éclairement.

Pour calculer des images de synthèse, il est nécessaire de respecter dans une certaine mesure les lois physiques qui régissent la lumière, le but étant d'obtenir des images suffisamment réalistes. Plus les lois seront fidèlement simulées, plus les images calculées pourront être proches d'une photographie de la réalité.

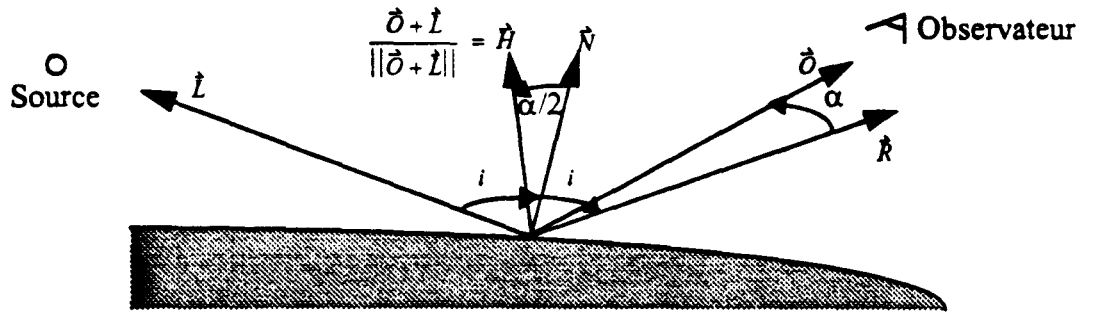


Figure 1•3 Les vecteurs utilisés pour l'éclairement.

1•2•1 Les lois physiques élémentaires.

1•2•1•1 La loi de Lambert.

• C'est la loi physique qui régit la réémission de la lumière pour les surfaces diffuses. Elle suppose une réémission uniforme de la lumière dans toutes les directions. L'intensité de lumière émise dépend de l'intensité de lumière reçue qui elle même va dépendre de l'intensité I de la source et de l'angle d'incidence i. Toute l'énergie reçue n'étant pas forcément intégralement réémise, la constante K_D, pour la matière utilisée, correspond au rapport entre l'énergie reçue et l'énergie réémise.

La formule régissant l'intensité de la lumière diffuse est : $I_D = K_D \times I \times \cos(i) = K_D \times I \times (\vec{N} \cdot \vec{L})$.

• Si le résultat du cosinus ou du produit scalaire est négatif, la source éclaire la facette (ou l'objet) par le côté opposé à l'observateur. Il n'y a pas réémission d'une intensité négative de lumière. Cela n'aurait aucun sens physique. L'intensité réémise est en fait nulle.

1•2•1•2 Les lois de Descartes.

• Les lois de Descartes donnent des indications sur la trajectoire de la lumière lors du passage de deux milieux d'indice différent. Après l'intersection du rayon incident et de l'interface, un rayon réfracté est produit dans le second milieu et un rayon réfléchi est produit dans le premier milieu. Les lois de Descartes fixent la direction de ces deux rayons et les équations de Maxwell déterminent l'intensité de chacun d'eux.

• Dans le cas d'une surface de type miroir, le modèle utilisé suppose une réémission intégrale de la lumière reçue dans la direction symétrique par rapport à la normale (c'est à dire la direction du seul rayon réfléchi). Si le vecteur $\hat{O}$ et le vecteur $\hat{R}$ sont colinéaires l'observateur reçoit toute l'intensité lumineuse de la source. Par contre, si les vecteurs ne sont pas colinéaires, l'observateur ne reçoit aucune lumière issue de cette source.

1•2•2 Les modèles de couleurs.

1•2•2•1 La répartition spectrale de la lumière.

• La couleur est une interprétation visuelle faite par le cerveau humain. L'oeil reçoit la lumière visible qui est composée d'un ensemble de longueurs d'onde comprises entre 400 nm et 700 nm. Un prisme permet d'observer le spectre de la lumière sous forme d'un arc en ciel ou de raies dans le cas d'une lumière composée d'un nombre limité de longueurs d'onde. Les couleurs dites fondamentales sont celles qui ne sont composées que d'une seule longueur d'onde.

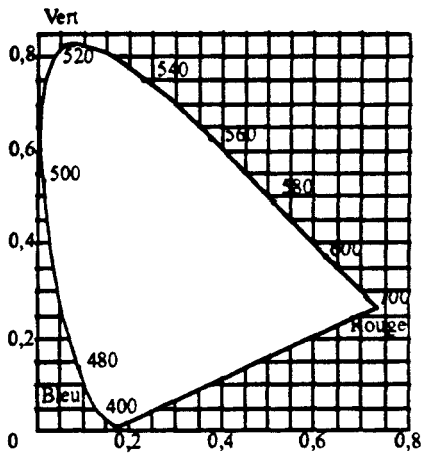


Figure 1•4 Le diagramme de chromaticité.

Le cerveau ne distingue pas la lumière sous sa forme spectrale mais simplement sous la forme d'une sensation colorée. Sur la Figure 1•4, les couleurs que peut voir l'être humain sont situées dans la zone blanche. Sur le contour de cette zone, sont situées, les couleurs fondamentales. A l'intérieur de la zone, on trouve les couleurs obtenues par mélange des couleurs fondamentales. L'intensité lumineuse constitue la troisième dimension du graphique (qui n'est pas représenté ici).

• Chaque point de la zone blanche est le barycentre d'un ensemble de points du contour. Le poids d'un de ces points est l'intensité de la couleur fondamentale associée, celui du barycentre est l'intensité de la couleur obtenue par ce mélange de couleurs fondamentales. Un même point peut être le barycentre de plusieurs ensembles. La couleur choisie sur l'exemple ci-contre peut être obtenue en mélangeant les couleurs de longueur d'onde 500 nm et 700 nm ou celles de longueur d'onde 480 nm et 560 nm.

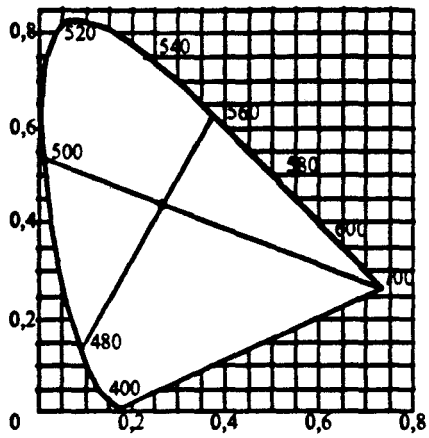


Figure 1•5 Exemple de mélange de couleurs.

1•2•2•2 Le modèle R.V.B.

• En prenant trois couleurs et en les mélangeant ont peut obtenir toutes les couleurs dont le lieu, sur le diagramme de chromaticité, est situé dans le triangle grisé. Les sommets de ce triangle sont les lieux des trois couleurs de base du modèle. C'est ce principe qui est utilisé pour reproduire les couleurs sur un téléviseur couleur ou un écran couleur d'ordinateur. Les trois couleurs choisies théoriquement sont le Rouge CIE, le Vert CIE et le Bleu CIE mais les couleurs réelles des chromatophores, des tubes cathodiques, diffèrent quelque peu des couleurs CIE.

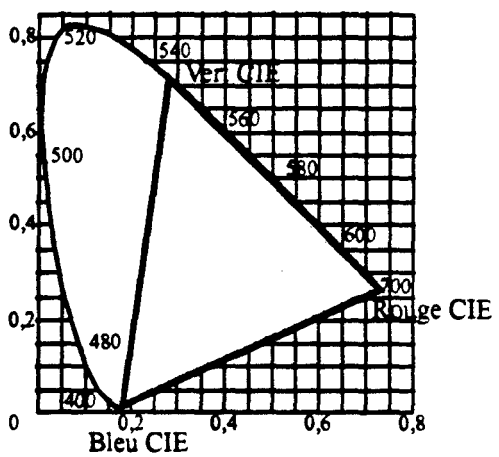


Figure 1•6 Diagramme du modèle R.V.B.

• Ce système permet de reproduire une grande partie des couleurs visibles bien que d'autres ne puissent être reproduites ainsi. C'est ce que l'on appelle le modèle R.V.B. Chaque couleur est obtenue par un mélange unique des trois couleurs de base. Pour mélanger deux couleurs E et F dont la décomposition en R.V.B. est respectivement $[E_R\ E_V\ E_B]$ et $[F_R\ F_V\ F_B]$, il suffit d'ajouter, les intensités des composantes de base.

$$(E+F)_R = E_R+F_R \qquad (E+F)_V = E_V+F_V \qquad (E+F)_B = E_B+F_B$$

1•2•2•3 Calcul de l'éclairement diffus en R.V.B.

• Dans le cas de l'utilisation du modèle R.V.B., on calcule l'éclairement diffus en adaptant la loi de Lambert. Soit $[C_R\ C_V\ C_B]$ la décomposition R.V.B. de la couleur de l'objet. Soit $[I_R\ I_V\ I_B]$ la décomposition R.V.B de la couleur de la source. Soit K_D la constante fixée par la matière pour l'éclairement diffus.

On a alors : $I_{DR} = (K_D \times C_R \times I_R) \times (\hat{N} \cdot \hat{L})$, $I_{DV} = (K_D \times C_V \times I_V) \times (\hat{N} \cdot \hat{L})$ et

$I_{DB} = (K_D \times C_B \times I_B) \times (\hat{N} \cdot \hat{L})$.

1•2•3 Le rayonnement spéculaire.

Dans la nature, on ne trouve pas uniquement des objets réémettant la lumière suivant le modèle diffus ainsi que d'autres réémettant la lumière tel un miroir. De nombreuses matières répondent à des modèles plus complexes. Pour prendre en compte l'éclairement de ce genre de surface, il faut utiliser une B.R.D.F.[1] telle que l'a définie Nicodemus en 1977. [39]

Lors de l'utilisation de la B.R.D.F., on calcule l'intensité lumineuse $L_r(\theta_r, \phi_r)$ émise par la facette dans la direction polaire (θ_r, ϕ_r) grâce à la formule :

$$L_r(\theta_r, \phi_r) = \int_0^{2\pi} \left(\int_0^{\frac{\pi}{2}} [L_i(\theta_i, \phi_i) \rho_{bd}(\theta_i, \phi_i, \theta_r, \phi_r) \cos\theta_i \sin\theta_i] \, d\theta_i \right) d\phi_i$$

Cette approche du calcul de l'éclairement permet de plus la représentation de surfaces non-isotropes dans les scènes [45].

• $\rho_{bd}(\theta_i, \phi_i, \theta_r, \phi_r)$ est la B.R.D.F. qui caractérise la surface de la facette.

• $L_i(\theta_i, \phi_i)$ est l'intensité reçue par la facette dans la direction polaire (θ_i, ϕ_i).

• Chaque matière a sa propre B.R.D.F. qui peut être obtenue grâce à des relevés physiques. Des travaux de recherches sont menés dans ce domaine au sein du Lauwrence Livermore Laboratory à Berkeley pour mesurer les différentes valeurs des B.R.D.F.[55].

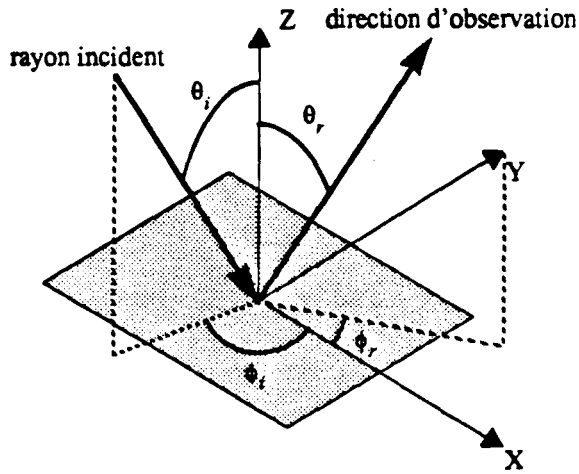


Figure 1•7 Les paramètres d'une B.R.D.F.

• Malgré le coût élevé de l'utilisation de B.R.D.F., il existe des méthodes qui ont été récemment introduites pour optimiser les calculs. Certaines de ces méthodes[2] proposent la décomposition de la B.R.D.F. suivant trois formes de rayonnement :

$$\rho_{bd} = \rho_{bd,sp} + \rho_{bd,dd} + \rho_{bd,ud}.$$

Le terme $\rho_{bd,sp}$ correspond à un éclairement de type spéculaire, le terme $\rho_{bd,dd}$ à un éclairement de type diffus directionnel et le terme $\rho_{bd,ud}$ à un éclairement de type diffus.

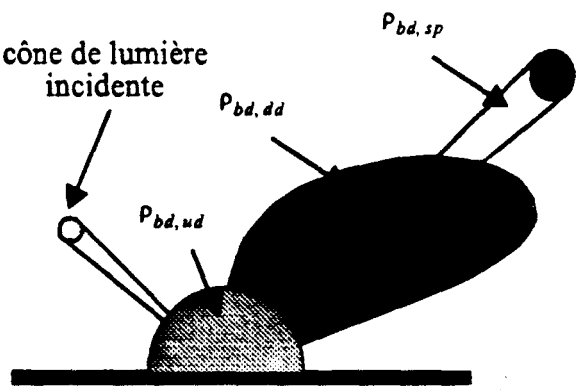


Figure 1•8 Décomposition d'une B.R.D.F.

1. B.R.D.F signifie en anglais "a bidirectionnal reflectance distribution function" c'est à dire une fonction bidirectionnelle de distribution de la réflectance.
2. voir la version multi-média sur CDROM de [26] ainsi que l'article [56]

• Toutefois ces méthodes d'optimisation ne sont pas encore suffisantes, aussi les anciennes formules empiriques sont encore grandement utilisées notamment lorsqu'il s'agit d'obtenir des images en temps réel. Ces formules empiriques sont certes plus simples, mais ne permettent pas la représentation des matières non-isotropes. Ces modèles ont pour principe de dissocier pour une même matière un éclairement de type diffus respectant la loi de Lambert et un éclairement de type spéculaire ajoutant l'effet de brillance. L'éclairement diffus directionnel n'est pas pris en compte par ces méthodes, celui-ci faisant tout au plus partie intégrante du spéculaire.

1•2•3•1 La formule empirique de Phong.

• Phong a proposé un modèle empirique pour reproduire le spéculaire. Dans le cas d'un miroir, la lumière est totalement renvoyée dans la direction du vecteur réfléchi $\vec{R}$. Cela reste la direction privilégiée de réémission de la lumière incidente. Mais ce modèle autorise en plus une certaine dispersion autour de cette direction. L'ouverture du "cône" de dispersion est fixée par l'exposant spéculaire e dans la formule suivante :

$$I_S = I \times K_S \times (\vec{R} \cdot \vec{O})^e = I \times K_S \times (\cos(\alpha))^e$$

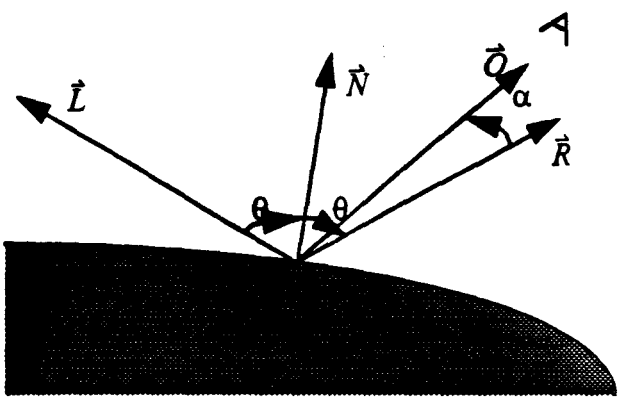


Figure 1•9 La formule de Phong.

Plus l'exposant spéculaire est élevé, plus l'angle entre la direction de l'observateur et la direction du vecteur réfléchi doit être faible, pour recevoir une énergie substantielle de lumière spéculaire. La constante K_S détermine la proportion de lumière réémise suivant le modèle spéculaire. Pour calculer l'éclairement en utilisant ce modèle, on ajoute l'intensité calculée pour la lumière diffuse à celle calculée pour la lumière spéculaire.

1•2•3•2 La formule empirique de Blinn.

• Une autre formule empirique a été proposée par Blinn pour reproduire le spéculaire [4]. Il ne s'agit plus d'utiliser le vecteur réfléchi $\vec{R}$, mais le vecteur surbrillance $\vec{H}$. Ce vecteur est défini comme étant un vecteur unitaire ayant pour direction celle de la bissectrice de la droite vers la source et de la droite vers l'observateur. La formule permettant d'obtenir le vecteur $\vec{H}$ est donc la suivante :

$$\vec{H} = \frac{\vec{O} + \vec{L}}{\|\vec{O} + \vec{L}\|}$$

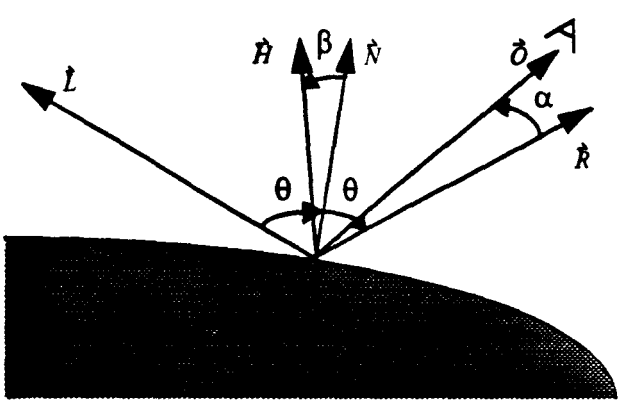


Figure 1•10 La formule de Blinn.

La formule permettant de calculer la composante spéculaire de l'éclairement est : $I_S = I \times K_S \times (\hat{N} \cdot \hat{H})^e = I \times K_S \times (\cos(\beta))^e$.

L'exposant spéculaire e garde son office, il règle l'ouverture du "cône" de dispersion, mais sa valeur n'est pas la même dans le modèle de Phong et dans celui de Blinn. Quand le vecteur $\hat{O}$ se rapproche du vecteur $\hat{R}$ le vecteur $\hat{H}$ se rapproche du vecteur $\hat{N}$. La direction privilégiée de réémission de la lumière spéculaire reste donc la direction du vecteur $\hat{R}$.

1•3 Les méthodes d'ombrage de facettes.

Afin de simplifier les calculs, lors des différentes phases de la synthèse d'images effectuée notamment par la méthode du rendu géométrique, la base de données de la scène est découpée en petites facettes. Une sphère, par exemple, pourra être découpée en 128 facettes ou plus, suivant la qualité désirée. Plus les facettes seront petites, plus les surfaces courbes seront reproduites fidèlement, mais le nombre de facettes sera plus important.

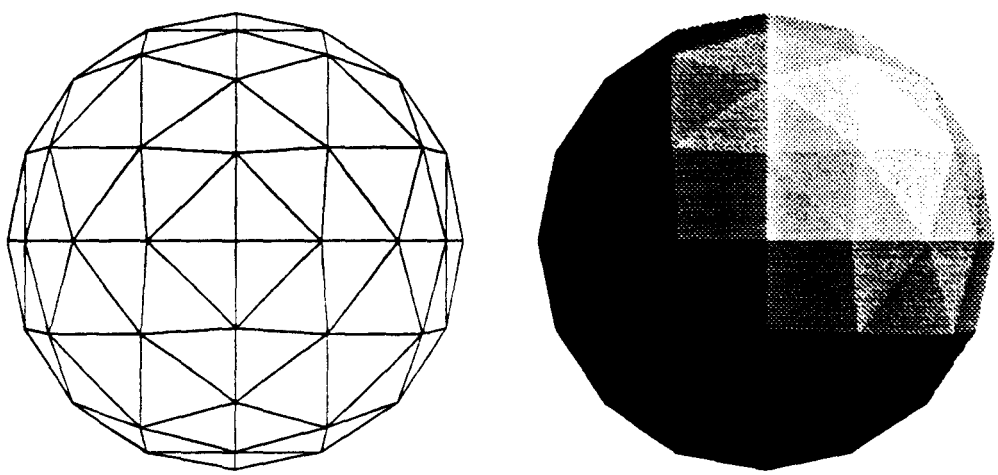


Figure 1•11 Sphère représentée par des facettes visualisée en "fil de fer" et en "ombrage constant".

Quand on représente une scène en "ombrage constant"[1], on calcule la couleur grâce aux formules d'éclairement au centre de chaque facette puis on associe cette couleur à l'ensemble de la facette.

Comme on le voit sur la figure ci-dessus (chaque facette ayant un ombrage constant c'est à dire une couleur constante) l'effet de dégradé de couleur n'est pas fidèle à la réalité. Pour remédier à ce problème et donc obtenir un véritable dégradé de couleur, deux méthodes ont successivement été proposées.

1•3•1 La méthode d'ombrage de Gouraud.

• La méthode de Gouraud est la première mais aussi la plus simple [24]. Elle consiste, à calculer pour chaque facette la couleur en ses sommets puis à effectuer une interpolation de la couleur sur la facette. Si on utilise les nuances de gris, on interpole l'intensité lumineuse. Si on utilise le modèle R.V.B., on effectue l'interpolation pour chacune des trois composantes du modèle (c'est à dire en rouge, en vert puis en bleu).

1. L'ombrage constant se dit "flat shading" en anglais.

• Lorsqu'on effectue l'interpolation, deux cas peuvent se présenter suivant la position de l'observateur. Il peut s'agir d'une projection orthogonale, avec l'observateur rejeté à l'infini ou bien d'une projection perspective (ou d'une transformation perspective), si l'observateur est à distance finie.

1•3•1•1 Interpolation lors d'une projection orthogonale.

• Il s'agit simplement de déterminer les coefficients a, b et c pour que la fonction $V(x,y) = a \times x + b \times y + c$ vérifie $V(A_X, A_Y) = V_A$, $V(B_X, B_Y) = V_B$ et $V(C_X, C_Y) = V_C$.

Ces relations s'écrivent sous forme matricielle : $\begin{bmatrix} V_A \\ V_B \\ V_C \end{bmatrix} = \begin{bmatrix} A_X & A_Y & 1 \\ B_X & B_Y & 1 \\ C_X & C_Y & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix}$. On détermine donc les coefficients de la forme bilinéaire en évaluant la relation matricielle : $\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} A_X & A_Y & 1 \\ B_X & B_Y & 1 \\ C_X & C_Y & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} V_A \\ V_B \\ V_C \end{bmatrix}$.

La matrice $M = \begin{bmatrix} A_X & A_Y & 1 \\ B_X & B_Y & 1 \\ C_X & C_Y & 1 \end{bmatrix}^{-1} = \dfrac{\begin{bmatrix} (B_Y - C_Y) & (C_X - B_X) & (B_X \times C_Y - B_Y \times C_X) \\ (C_Y - A_Y) & (A_X - C_X) & (A_Y \times C_X - A_X \times C_Y) \\ (A_Y - B_Y) & (B_X - A_X) & (A_X \times B_Y - A_Y \times B_X) \end{bmatrix}}{\begin{vmatrix} A_X & A_Y & 1 \\ B_X & B_Y & 1 \\ C_X & C_Y & 1 \end{vmatrix}}$ existe dès lors que le déterminant $\begin{vmatrix} A_X & A_Y & 1 \\ B_X & B_Y & 1 \\ C_X & C_Y & 1 \end{vmatrix}$ n'est pas nul c'est à dire dès lors que la facette n'est pas dégénérée.

• C'est une méthode qui fonctionne très bien quand le modèle d'éclairement utilisé est purement diffus. C'est à dire si la valeur de K_S est nulle.
La figure ci-contre illustre l'ombrage de Gouraud calculé pour la sphère décomposée en 128 facettes triangulaires. En comparaison de l'ombrage constant présenté dans la Figure 1•11, c'est une amélioration importante de la qualité de l'image.

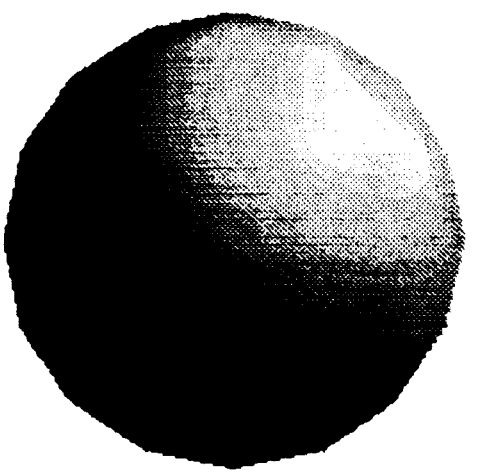


Figure 1•12 Ombrage de Gouraud d'une sphère sans éclairement spéculaire.

• Par contre, des défauts visuels importants peuvent apparaître avec la méthode de Gouraud quand le modèle d'illumination utilise la formule de Phong ou de Blinn pour calculer un éclairement spéculaire.

• La variation importante et extrêmement rapide de l'intensité au coeur de la tache spéculaire ne peut être fidèlement reproduite par l'interpolation de l'intensité lumineuse.
On constate sur la figure ci-contre la maigre qualité, lors du rendu de l'éclairement spéculaire pour la sphère découpée en 128 facettes. Les paramètres, pour la matière qui compose la sphère, sont :
$K_D = 110$ $K_S = 185$ et $e = 15$. Le spéculaire est calculé avec la formule de Phong.

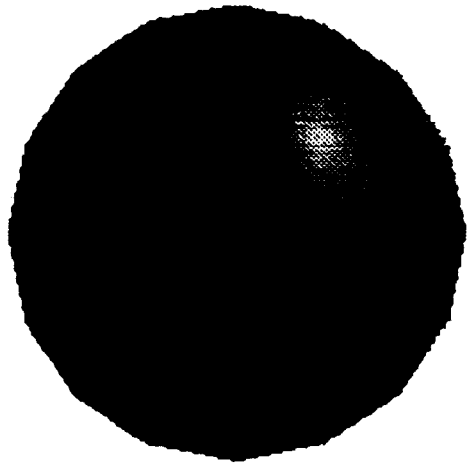


Figure 1•13 Ombrage de Gouraud avec éclairement spéculaire.

1•3•1•2 Interpolation lors d'une projection perspective.

• La projection orthogonale n'est pas suffisante pour imiter la vision du monde réel car on a l'habitude de voir en perspective. Plusieurs méthodes peuvent être envisagées pour résoudre le problème. Il est possible d'utiliser une projection perspective. Il faut alors effectuer l'interpolation dans l'espace réel. La couleur est obtenue pour chaque élément de surface de la facette par interpolation, comme dans le cas de la projection orthogonale. Puis elle est associée au pixel projeté dont on obtient les coordonnées comme il est expliqué au paragraphe 1•1•1•3 en calculant : $\frac{D \times x}{D - Z}$ et $\frac{D \times y}{D - Z}$.

• Par contre si on aborde le problème par le biais de la transformation perspective, l'interpolation doit être effectuée dans l'espace virtuel. Il faut calculer l'interpolation en évaluant en chaque sommet le terme $(D \times I)/(D - Z)$. Le résultat de l'interpolation doit donc être multiplié par $(D - Z)/D$ pour obtenir l'intensité. Mais lors de la transformation perspective on ne dispose pas de la valeur de Z (la profondeur dans l'espace réel) mais de la valeur de Z' (la profondeur dans l'espace virtuel).
Or $Z' = (D \times Z)/(D - Z)$ donc $Z = (D \times Z')/(D + Z')$ et $(D - Z)/D = D/(D + Z')$. L'interpolation s'avère donc coûteuse dans le cas de la transformation perspective puisqu'elle nécessite le calcul d'une division en chaque pixel pour obtenir le résultat final.

• Pour remédier à ce problème, on pourrait établir une approximation de la division qui soit une fonction du second degré, mais il y a plus simple. Si les facettes sont petites la variation sur la facette de la profondeur Z dans l'espace réel ou de la profondeur Z' dans l'espace virtuel sera faible. Dans ce cas, on peut effectuer l'interpolation dans l'espace réel sans se préoccuper de la faible erreur que cela occasionne sur le calcul de l'éclairement. On préfère donc régler la taille de la facette utilisée suivant la précision visuelle voulue.

1•3•1•3 Conclusion.

• Que l'observateur soit ou non à l'infini, plus la transformation des objets produira de petites facettes, meilleure sera la qualité de l'image. On résout, en outre, l'utilisation du spéculaire. Mais malheureusement le nombre de facettes à traiter devenant très grand, il devient de plus en plus difficile d'obtenir le temps réel pour les scènes contenant beaucoup d'objets. Pour améliorer le rendu du spéculaire, Phong proposa une autre méthode d'ombrage.

1•3•2 La méthode d'ombrage de Phong.

• La méthode de Phong[44] consiste pour chaque facette à calculer le vecteur normal $\hat{N}$ en chacun de ses sommets puis à effectuer une interpolation bilinéaire des trois composantes du vecteur $\hat{N}$ sur la facette. Ensuite pour chaque pixel de la facette, on calcule l'éclairement en utilisant la valeur de $\hat{N}$ obtenue. Le vecteur est normalisé (c'est à dire que sa norme est rendue unitaire) en chaque sommet, mais il doit l'être aussi en chaque pixel car l'interpolation bilinéaire des composantes séparées n'assure pas une variation linéaire de la norme.

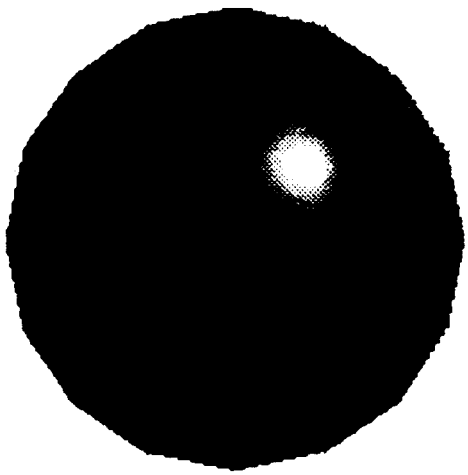


Figure 1•14 Ombrage d'une sphère par la méthode de Phong.

Cette méthode permet non seulement un rendu correct de l'éclairement diffus mais aussi de l'éclairement spéculaire.

• Quand l'observateur et la source sont situés à une distance finie de l'objet, il est nécessaire de recalculer en chaque pixel le vecteur $\hat{L}$ vers la source pour déterminer l'éclairement diffus et le vecteur surbrillance $\hat{H}$ pour l'éclairement spéculaire. T. B. Phong proposa parallèlement d'utiliser des interpolations bilinéaires des composantes pour obtenir ces deux vecteurs à partir de leurs valeurs aux sommets de la facette.

1•3•2•1 Interpolation lors d'une projection orthogonale.

• Il s'agit simplement de déterminer les trois triplets de coefficients pour chacune des trois formes bilinéaires.

$$N_{x(x,y)} = ax + by + c \ , \ N_{y(x,y)} = dx + ey + f \text{ et } N_{z(x,y)} = gx + hy + i$$

Les neufs coefficients, déterminés pour les trois formes bilinéaires, donnent aux sommets de la facette les composantes du vecteur normal normé. On aboutit ainsi à la relation matricielle suivante :

$$\begin{bmatrix} N_{xA} & N_{yA} & N_{zA} \\ N_{xB} & N_{yB} & N_{zB} \\ N_{xC} & N_{yC} & N_{zC} \end{bmatrix} = \begin{bmatrix} A_X & A_Y & 1 \\ B_X & B_Y & 1 \\ C_X & C_Y & 1 \end{bmatrix} \cdot \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \text{ qui se résout en donnant :}$$

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} = M \cdot \begin{bmatrix} N_{xA} & N_{yA} & N_{zA} \\ N_{xB} & N_{yB} & N_{zB} \\ N_{xC} & N_{yC} & N_{zC} \end{bmatrix} \text{ et } M \text{ étant la matrice définie au paragraphe 1•3•1•1.}$$

1•3•2•2 Interpolation lors d'une projection perspective.

• Si on aborde le problème de la perspective, en effectuant une projection perspective, l'interpolation s'effectue dans l'espace réel. Les composantes du vecteur normal issues de l'interpolation sont affectées au pixel projeté en calculant ses coordonnées : $\frac{D \times x}{D - Z}$ et $\frac{D \times y}{D - Z}$.

• Par contre, si on aborde le problème par le biais de la transformation perspective, l'interpolation doit être effectuée dans l'espace virtuel. L'interpolation doit donc concerner $\frac{D}{D-Z}\cdot\hat{N}$ et on obtient un vecteur normal multiplié par $\frac{D}{D-Z}$. Comme de toute façon, il faut normé le vecteur obtenu après l'interpolation, la transformation perspective n'augmente pas la complexité du calcul du vecteur $\hat{N}$. Cette remarque reste valable pour tout autre vecteur dont les composantes seraient issues d'une interpolation bilinéaire.

1•3•2•3 Remarque.

• Quel que soit le nombre de sources lumineuses, une seule interpolation du vecteur $\hat{N}$ est suffisante. Par contre, il est nécessaire d'effectuer une interpolation du vecteur $\hat{H}$ et du vecteur $\hat{L}$ pour chaque source puisque ces vecteurs dépendent de la position de la source.

1•3•3 Les techniques d'accélération de l'ombrage de Phong.

Outre l'accroissement de qualité d'image lors de l'utilisation de la méthode de Phong par rapport à celle de Gouraud, c'est la complexité des calculs qui elle aussi est fortement accrue. Différentes méthodes ont donc été proposées pour accélérer ou simplifier la méthode d'ombrage de Phong sans trop amoindrir la qualité des images obtenues.

1•3•3•1 Sans renormaliser les vecteurs.

• La première simplification suppose de ne pas normaliser les vecteurs issus de l'interpolation. La qualité est amoindrie mais les images fournies sont meilleures que par l'interpolation de Gouraud comme le montre Claussen dans [8].

1•3•3•2 L'interpolation du produit scalaire.

• Cette méthode, qui consiste à effectuer l'interpolation des produits scalaires $\hat{N}\cdot\hat{L}$ et $\hat{N}\cdot\hat{H}$, offre un résultat visuel meilleur. Elle permet également d'économiser la normalisation des vecteurs.

Il ne faut plus calculer l'interpolation du vecteur $\hat{N}$, mais cela requiert l'interpolation des deux produits scalaires pour chaque source.

1•3•3•3 L'interpolation des angles.

• Cette méthode consiste à interpoler le vecteur $\hat{N}$ en coordonnée polaire (ρ,θ,φ). En l'occurrence le vecteur restera normé si $\rho = 1$. Il est donc seulement nécessaire d'interpoler les angles. D'après les travaux publiés par Claussen, les résultats visuels de cette méthode, bien qu'ils soient meilleurs que ceux obtenus par l'interpolation du produit scalaire, sont cependant moins bons que ceux obtenus pour l'interpolation des composantes de la normale.

1•3•3•4 Méthode de Bishop et Weimer.

Bishop et Weimer[2] ont proposé en 1986 une méthode pour accélérer le calcul de l'ombrage de Phong. Les formules d'éclairement du diffus et du spéculaire sont les suivantes :

$$I_{diffus}(x,y) = \frac{(\hat{L}\cdot\hat{A})x + (\hat{L}\cdot\hat{B})y + \hat{L}\cdot\hat{C}}{\|\hat{L}\| \times \|(\hat{L}\cdot\hat{A})x + (\hat{L}\cdot\hat{B})y + \hat{L}\cdot\hat{C}\|}$$

Les vecteurs $\hat{A}$, $\hat{B}$ et $\hat{C}$ sont fixés par la géométrie de la facette.

$$I_{spéculaire}(x,y) = [DP(x,y)]^n \text{ avec } DP(x,y) = \frac{(\hat{A}'x+\hat{B}'y+\hat{C}')\cdot(\hat{D}'x+\hat{E}'y+\hat{F}')}{\|\hat{A}'x+\hat{B}'y+\hat{C}'\| \times \|\hat{D}'x+\hat{E}'y+\hat{F}'\|}$$

Les vecteurs $\hat{A}'$, $\hat{B}'$, $\hat{C}'$, $\hat{D}'$, $\hat{E}'$ et $\hat{F}'$ dépendent de la géométrie mais également de la position de l'observateur et de la source.

• La méthode proposée consiste à utiliser des développements de Taylor pour $I_{diffus}(x,y)$ et pour $DP(x,y)$. Plutôt que d'utiliser une forme bilinéaire pour calculer la couleur comme dans le cas de la méthode de Gouraud, on utilise ici une forme de degré supérieur pour améliorer le rendu du spéculaire.

• Pour différentes sources, il est possible de sommer les coefficients des développements de Taylor relatifs au calcul du diffus. Mais cela est impossible pour le spéculaire car il faut effectuer l'élévation à la puissance du terme $DP(x,y)$ indépendamment pour chaque source.

1•3•4 Comparaison critique des méthodes de Phong et de Gouraud.

• Outre le fait d'améliorer la qualité des images, la méthode de Phong présente d'autres avantages comparée à la méthode de Gouraud. Lors de la conversion des objets en pixels, avec la méthode de Gouraud, il est nécessaire d'effectuer les calculs du modèle d'éclairement pour chaque source et pour chaque facette dans le hôte afin de déterminer les coefficients de l'interpolation. Dans le cas de la méthode de Phong, il est possible de limiter les calculs du hôte. En effet, si l'on interpole seulement le vecteur $\hat{N}$, les calculs effectués par le hôte en vue de l'interpolation sont indépendants du nombre de sources.

Par contre, la complexité de l'unité de conversion des objets en pixels est légèrement supérieure. Dans le cas de la méthode de Gouraud, il faut effectuer trois interpolations pour les trois couleurs. Par contre, en ce qui concerne la méthode de Phong, il faut toujours effectuer trois interpolations mais pour obtenir les composantes du vecteur $\hat{N}$. Les couleurs sont codées sur 8 bits mais il faudra vraisemblablement utiliser des mots plus larges pour coder les composantes du vecteur $\hat{N}$. La largeur exacte fera l'objet d'une étude approfondie au cours des chapitres suivants.

Avec la méthode de Phong, les calculs d'éclairement sont repoussés après l'éliminations des parties cachées. L'éclairement ne sera donc calculé que pour les pixels visibles. La puissance requise par les unités d'éclairement est donc indépendante de la complexité de la scène.

L'inconvénient majeur de la méthode de Phong est le coût de chaque unité d'éclairement qui doit être utilisée pour chaque source après l'élimination des parties cachées.

1•4 L'ombrage de quadriques.

• Dans le cas des quadriques, il est possible de transformer ces objets en facettes mais il est aussi possible de les traiter grâce à leur propre équation. On peut calculer la profondeur d'une quadrique avec une racine carrée, une addition, une expression du premier degré et une autre du second degré [29].

$$Z = A\cdot x + B\cdot y + C \pm \sqrt{D\cdot x^2 + E\cdot xy + F\cdot y^2 + G\cdot x + H\cdot y + I}$$

On peut donc effectuer l'élimination des parties cachées sans avoir obligatoirement recours aux facettes pour représenter l'objet. Le contour de l'objet est obtenu directement à partir de l'évaluation de la profondeur. Un pixel est intérieur au contour si la profondeur est calculable c'est à dire si le terme sous la racine est positif. On obtient donc le véritable contour de l'objet et non plus une forme approchée par les arêtes des facettes. Le vecteur de la normale peut être obtenu de la même façon.

$$N_i = A_i \cdot x + B_i \cdot y + C_i \pm \sqrt{D_i \cdot x^2 + E_i \cdot xy + F_i \cdot y^2 + G_i \cdot x + H_i \cdot y + I_i}$$

• Dans le cas de la transformation perspective, il faut calculer non pas la normale à la surface de l'espace virtuel mais calculer la valeur de la normale à la surface dans l'espace réel correspondant au point de l'espace virtuel. Mais cela ne change rien quant à la forme de la fonction produisant les composantes du vecteur $\hat{N}$.

• Il ne s'agit donc plus d'effectuer une interpolation de Phong mais bien de calculer les vraies valeurs du vecteur normal. Ce qui peut être fait pour les quadriques peut aussi être fait pour des objets plus complexes comme les quartiques. Il faudra alors trouver des méthodes rapides permettant le calcul de la vraie profondeur et de la vraie normale. Cela sera suffisant pour être en mesure de les utiliser sans modifier l'unité d'éclairement. (La conversion de l'objet en pixels n'est pas le sujet de ce mémoire.)

Puisqu'il s'agit de déterminer la vraie normale, le choix entre le calcul des composantes cartésiennes ou le calcul des coordonnées sphériques n'affectera pas la qualité de l'image. Le choix de la méthode aura essentiellement une influence sur la complexité d'une éventuelle architecture.

1•5 Les textures.

L'utilisation des différents modèles d'éclairement permet d'augmenter la qualité des images produites, mais les objets sont représentés avec un aspect surfacique uniforme. Or, un grand nombre de matières présentent des irrégularités de leur surface comme le bois ou la peau d'orange par exemple. Un certain nombre d'algorithmes ont été développés pour simuler l'aspect texturé des objets.

1•5•1 Le "mappage" de textures 2D.

• Les premières techniques efficaces de mappage de textures ont été proposées par Catmull [6] puis reprises par Blinn et Newell [3] en 1975 et 1976. Le principe consiste à appliquer un motif plan sur un objet. Pour cela, la technique appelée "transformation inverse"[1] est utilisée. Elle transforme les coordonnées (x, y, z) de l'élément de surface de l'objet en coordonnées (u, v) de l'élément de la texture plane correspondante. Les fonctions $f_1(x, y, z) = u$ et $f_2(x, y, z) = v$ sont déterminées suivant le relief de l'objet.

Quand les objets sont découpés en facettes, les valeurs de u et v sont calculées aux sommets des facettes et les valeurs intérieures aux facettes sont obtenues par interpolation bilinéaire comme pour l'intensité, dans la méthode d'ombrage de Gouraud, ou les normales, pour la méthode d'ombrage de Phong.

1. Cette technique est appelée "inverse mapping" en anglais.

Dans le cas d'une scène visualisée en perspective, le problème de l'interpolation qui a été exposé au paragraphe 1•3•1•2 reste d'actualité. Il est nécessaire de calculer une division. Cependant, il est possible d'évaluer de façon approximative la fonction rationnelle qui devrait être utilisée par une fonction quadratique plus facile à calculer. Dans le cas présent, cette approximation fournit un résultat visuel très satisfaisant.

1•5•2 La perturbation de normales.[1]

Pour donner un effet 3D aux textures, on peut ajouter au vecteur normal une perturbation.

$$N_x \rightarrow N_x + p_x(x,y)$$
$$N_y \rightarrow N_y + p_y(x,y)$$
$$N_z \rightarrow N_z + p_z(x,y)$$

On calcule ensuite l'éclairement en utilisant le vecteur perturbé qu'il convient de normaliser. La nature de la texture obtenue est déterminée par les trois fonctions de perturbation $P_x(x,y)$, $P_y(x,y)$ et $P_z(x,y)$.

1•5•3 Les textures 3D.

La méthode de perturbation de la normale n'est pas suffisante pour donner un véritable effet 3D aux textures. Par exemple, pour donner l'idée qu'un objet est en marbre, il est nécessaire de calculer la texture en découpant l'objet dans un bloc de marbre. Il faut donc une véritable représentation 3D de chaque matière. L'ajout de la texture sur l'objet est obtenu en recherchant la couleur (ou les paramètres lumineux) de la surface dans la matrice 3D des paramètres de la matière. Il est possible de limiter la taille de la matrice en considérant que la matière est constituée de motifs périodiques.

1•6 Les architectures dédiées.

Les architectures dédiées à la synthèse d'images requièrent une grande puissance de calcul. Cette puissance est obtenue en répartissant l'opération de conversion des objets en pixels sur un ensemble de processeurs spécialisés [19]. Suivant le parallélisme utilisé, on peut classer les architecture selon deux types.

D'une part, on trouve les systèmes à parallélisme pixels qui distribuent les pixels de l'écran sur le réseau de processeurs. Chacun d'eux reçoit tous les objets de la scène, calcule l'appartenance du pixel à l'objet, la profondeur puis effectue l'élimination des parties cachées.

D'autre part, on trouve les systèmes à parallélisme objets qui distribuent les objets de la scène sur le réseau de processeurs. Chacun d'eux reçoit un objet et effectue les calculs pour tous les pixels de l'écran. (Une classification de ces architectures peut être obtenue dans [7].)

1•6•1 Les architectures à parallélisme pixels.

Plusieurs projets de recherche, ont étudié des machines utilisant le modèle du parallélisme pixels. On dénombre, entre autres, les projets Pixel-Planes 4, Pixel-Power et Pixel-Planes 5.

---

1. Cette technique est appelée "bump mapping" en anglais.

1•6•1•1 Pixel-Planes 4.

Le projet Pixel-Planes 4 a été développé par une équipe de recherche de l'université de Caroline du Nord dès 1981 [15]. L'architecture générale de la machine est présentée dans la figure ci-dessous.

• Le Système mémoire 'Smart' est un réseau de processeurs élémentaires pouvant calculer pour tous pixels le résultat d'une expression linéaire en x et y. C'est à dire $A \times x + B \times y + C$. Les coefficients A, B et C proviennent du pré-processeur. Chaque processeur pixel dont la présentation synoptique est donnée dans la figure ci-contre dispose, en plus, d'un registre de profondeur et d'un comparateur, intégrés dans l'ALU 1 bit, pour exécuter l'algorithme du Z-buffer. D'autres registres permettent la mémorisation de la couleur.

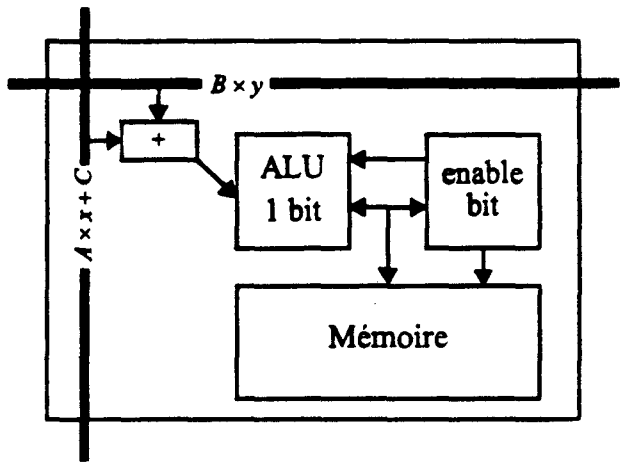


Figure 1•15 Vue synoptique d'un processeur.

• La couleur, comme la profondeur peut être obtenue pour chaque pixel comme le résultat d'une expression linéaire [16]. Cela permet de calculer l'ombrage des facettes par la méthode de Gouraud. La machine peut utiliser également la méthode d'ombrage de Phong. Dans ce cas les trois composantes du vecteur normal $\hat{N}$ sont calculées à la place des trois composantes de la couleur. Puis, lorsque la phase de conversion des objets en pixels et d'élimination des parties cachées est terminée, chaque processeur pixel normalise le vecteur $\hat{N}$ qu'il contient et calcule par programme la formule d'éclairement pour chaque source. Le temps d'exécution de l'algorithme dépend du nombre de sources et de la nature de chaque source.

1•6•1•2 Pixel-Power.

• Le projet Pixel-Power est la suite du projet Pixel-Planes 4. Afin d'améliorer les possibilités du rendu de la machine, les expressions calculées en chaque pixel sont désormais du second degré [22]. Cela permet d'améliorer la qualité des primitives, plus complexes que la facette, telles les sphères et les cylindres. L'architecture générale de la machine reste la même.

L'obtention d'une expression du second degré pour calculer les composantes de la couleur trouve aussi son intérêt pour l'utilisation de la méthode d'accélération du calcul de l'éclairement de Phong proposée par Bishop et Weimer. (Voir paragraphe 1•3•3•4.)

1•6•1•3 Pixel-Planes 5.

• Une étude de la rentabilité de l'architecture des projets Pixel-Planes 4 et Pixel-Power a montré que le réseau de processeurs pixels était sous-exploité [17]. Les facettes étant de petites tailles, seulement une faible partie du réseau travaille encore lorsque le contour de la facette est complètement calculé. Pour remédier à ce problème, l'unité "Smart" doit seulement s'occuper d'une petite partie de l'écran afin d'augmenter la rentabilité. L'architecture générale du projet, qui a donc été remaniée [18], est présentée dans la figure ci-après.

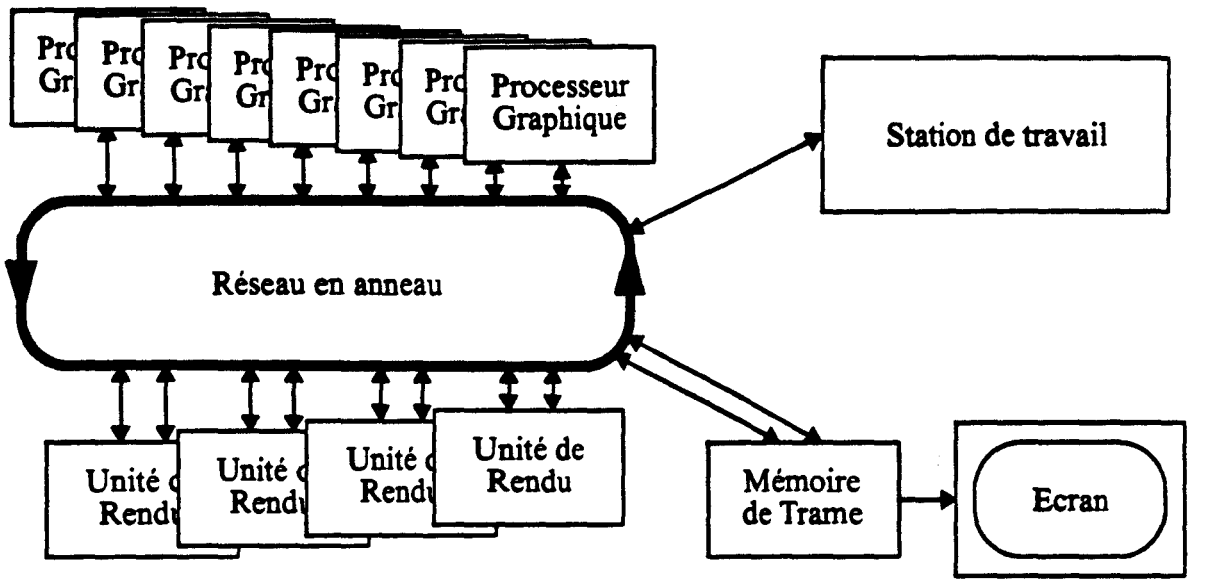


Figure 1•16 Architecture générale de la machine Pixel-Planes 5.

• Chaque unité de rendu, dont l'architecture est présentée ci-contre, dispose d'un réseau de 128 sur 128 processeurs. Comme dans le projet Pixel-Power, chaque processeur reçoit pour son pixel les résultats d'expressions du second degré et les utilise pour effectuer la conversion des objets en pixels, l'élimination des parties cachées et le calcul de l'éclairement.

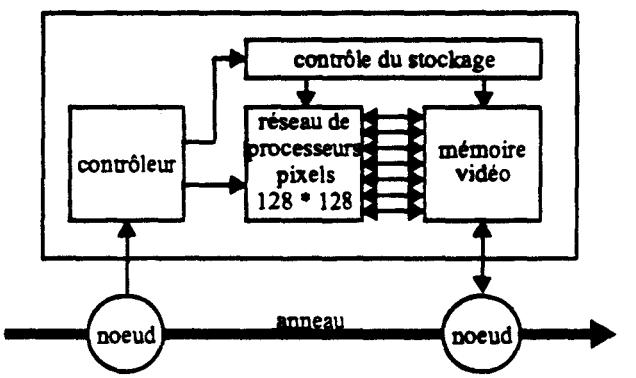


Figure 1•17 L'unité de Rendu.

• L'ensemble des processeurs graphiques constitue le hôte du système graphique. Par rapport aux projets Pixel-Planes 4 et Pixel-Power, sa puissance a été accrue pour suivre l'évolution des performances de l'unité graphique. Il a en charge tous les calculs des coefficients A, B, C, D, E, et F des expressions du second degré :

$$A \times x^2 + B \times xy + C \times y^2 + D \times x + E \times y + F.$$

• Bien que l'on puisse utiliser la méthode de Bishop et Weimer pour calculer l'éclairement de Phong, la puissance des processeurs pixels permet le calcul direct par programmation. Cela allège ainsi la tâche des processeurs graphiques. Avec la méthode de Bishop et Weimer, les processeurs graphiques devaient calculer, pour chaque source, les coefficients des deux développements limités de second ordre des produits scalaires. En appliquant directement la méthode de Phong, ils doivent seulement calculer les coefficients des trois expressions linéaires pour les coordonnées du vecteur normal. On déporte ainsi le calcul de l'éclairement sur le réseau pixel après l'élimination des parties cachées.

1•6•2 Les systèmes à parallélisme objet.

D'autres projets de recherche, ont étudié des machines utilisant le modèle du parallélisme objet. On dénombre, entre autres, les projets G.S.P.-N.V.S., PROOF et I.M.O.G.E.N.E.

1•6•2•1 G.S.P.-N.V.S.[1]

• Le projet G.S.P.-N.V.S. a été développé par l'équipe de "Schlumberger Palo Alto Research" [13]. L'architecture générale de la machine est un pipeline. Il débute avec la station de travail servant d'hôte au système graphique et se termine avec une mémoire de trame. Celle ci suivie du moniteur vidéo, mémorise les trois composantes de la couleur et la profondeur.

• L'unité DLR[2] extrait la géométrie et les commandes d'affichage depuis la RAM qui sert de tampon entre le système graphique G.S.P.-N.V.S. et la station de travail hôte. Puis les transformations et le fenêtrage (ou clipping en anglais) sont réalisés par l'unité XTC. Le buffer Y effectue le tri des triangles suivant leur ordre d'apparition lors du balayage vertical de l'écran. 1024 listes sont ainsi constituées correspondant au 1024 lignes de l'écran.

Chaque liste contient les informations relatives à chaque facette apparaissant pour la première fois sur la ligne courante. Le pipeline de processeurs triangles reçoit les informations issues du buffer Y et calcule le contour et la profondeur pour effectuer l'élimination des parties cachées.

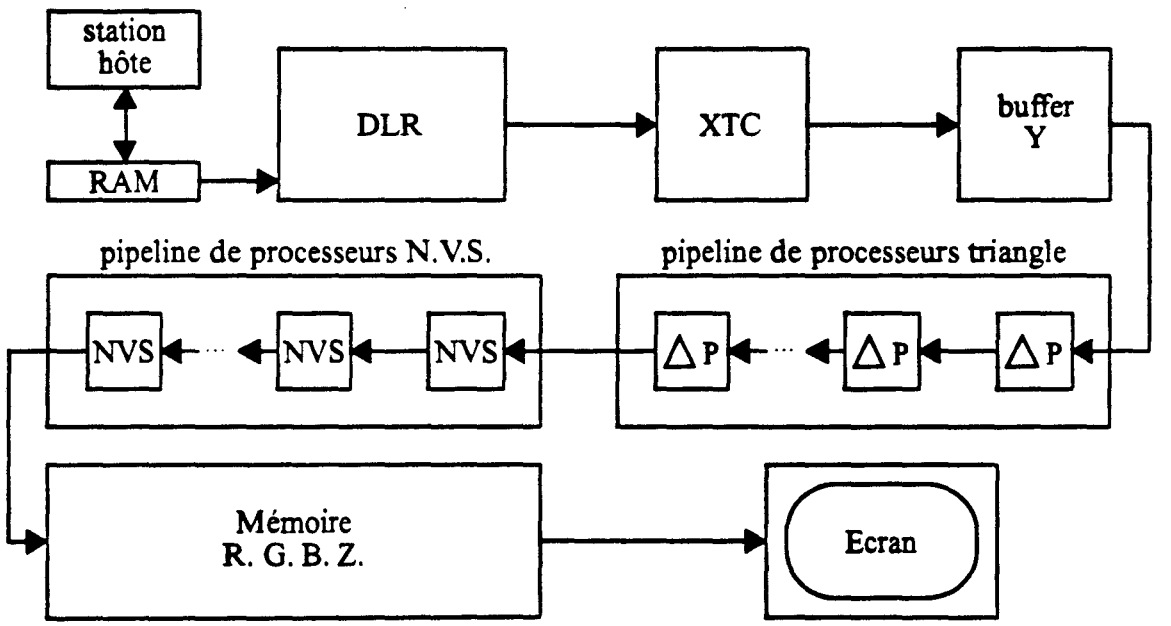


Figure 1•18 Architecture générale de la machine G.S.P.-N.V.S.

• Pour effectuer le calcul de l'éclairement, les processeurs triangles fournissent pour chaque pixel, un numéro de matière codé sur 9 bits et un vecteur normal dont chacune des trois composantes est codée sur 18 bits. Le système graphique peut donc utiliser simultanément 512 matières.

1. G.S.P. signifie en anglais "Graphical Signal Processing" et N.V.S. signifie "Normal Vector Shader".
2. Display List Runner.

• Les caractéristiques de la matière, telles que la couleur, son coefficient diffus, son coefficient spéculaire et son exposant spéculaire, seront retrouvées par les unités d'éclairement à partir du numéro. Ces informations sont stockées, dans les processeurs N.V.S., dans une SRAM réservée à cet effet.

• L'ensemble du pipeline des 16 processeurs N.V.S. calcule l'éclairement pour 5 sources directionnelles. (C'est à dire situées à l'infini.) Selon les concepteurs, ce pipeline aurait une puissance de 2 milliards de multiplications par seconde. Le modèle d'éclairement choisi est le modèle de Phong utilisant le vecteur réfléchi. (Voir paragraphe 1•2•3•1.) L'observateur est à distance finie.

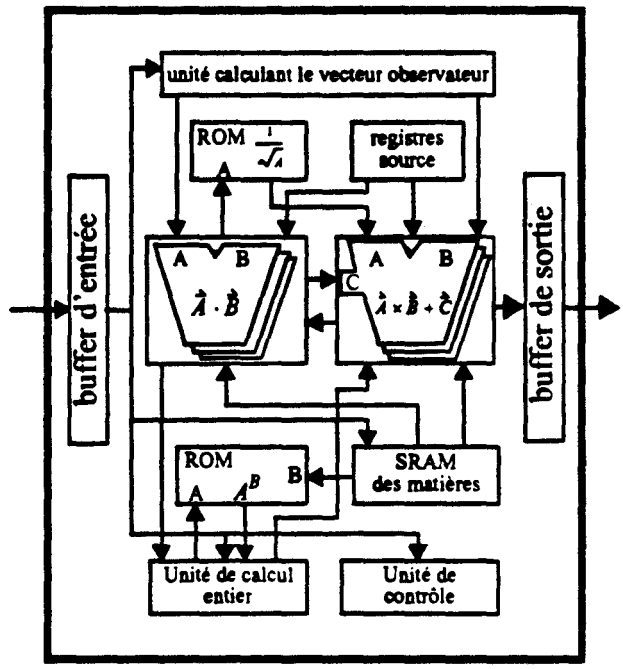


Figure 1•19 Diagramme d'un N.V.S.

Le processeur N.V.S. est constitué d'une unité d'extraction de la racine carrée et d'une unité d'élévation à la puissance, que les concepteurs du projet proposaient d'intégrer sous forme de ROM. D'autre part, une unité servant pour le calcul incrémental du vecteur observateur, une unité de calcul d'un produit scalaire et trois unités parallèles de multiplication-addition, sont également intégrées dans chaque processeur. Le projet n'a malheureusement pas été mené à terme.

1•6•2•2 P.R.O.O.F.[1]

• Le projet P.R.O.O.F. a été développé par une équipe de recherche de l'université de Tübingen en Allemagne. Son architecture, telle qu'elle a été proposée par B.O. Schneider[49], est présentée dans la figure suivante.

• L'architecture proposée ressemble par bien des points à celle du projet G.S.P.-N.V.S. mais elle intègre en plus un mécanisme pour prendre en compte l'anti-aliasing[2] par sur-échantillonnage. Des processeurs de filtrage sont donc ajoutés à la fin du pipeline.

• Les tables LUT permettent la réduction de la largeur des liaisons entre les différents blocs de l'unité de rendu. Les processeurs objets effectuent la conversion des objets en pixels, l'élimination des parties cachées et fournissent enfin la profondeur et une interpolation de la direction de la normale.

U. Claussen a étudié, pour ce projet, les diverses méthodes et techniques d'accélération de l'ombrage de Phong déjà présentées au paragraphe 1•3•3. La meilleure qualité est obtenue quand les coordonnées du vecteur normal $\hat{N}$ interpolées dans les processeurs objets sont communiquées au pipeline des processeurs d'ombrage.

1. Pipeline for Rendering in an Object Oriented Framework.
2. L'aliasing regroupe tous les défauts visuels introduits, sur une image, par un échantillonnage qui ne respecte pas le théorème de Shannon. L'anti-aliasing est la technique qui tente de supprimer l'aliasing.

Comme la normalisation d'un vecteur s'avère une opération très coûteuse, les concepteurs du projet ont décidé d'utiliser la méthode d'éclairement de Blinn avec la source et l'observateur à l'infini. Ainsi qu'on va le montrer au chapitre suivant, grâce à cette restriction, seul le vecteur normal doit être normé. D'autre part, ils ont estimé inutile de normaliser le vecteur $\hat{N}$ en chaque pixel de la facette. (Voir paragraphe 1•3•3•1.)

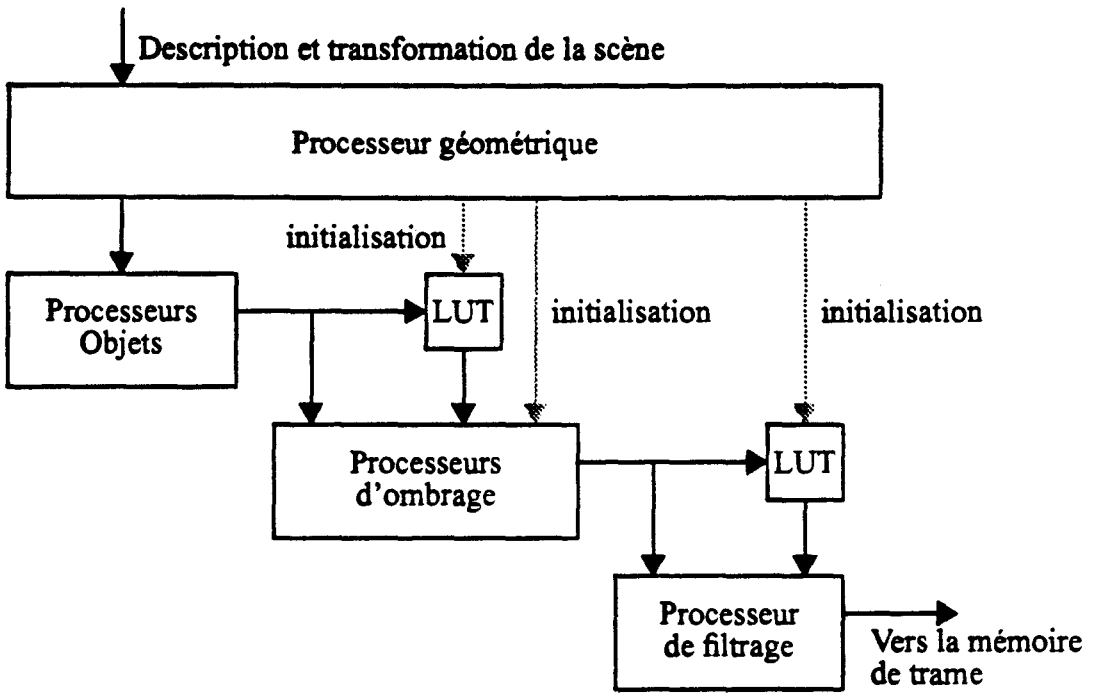


Figure 1•20 Architecture générale de la machine P.R.O.O.F.

Dans le pipeline des processeurs d'ombrage, on associe pour chaque source deux processeurs. Le premier calcule l'éclairement diffus et le second l'éclairement spéculaire.[10]

1•6•2•3 I.M.O.G.E.N.E.[7]

• Ce projet est développé par le Laboratoire d'Informatique Fondamentale de Lille. Il s'apparente aux deux projets précédemment présentés car il s'agit là encore d'un système de conversion objet. Comparée à ses deux homologues, cette machine présente surtout l'originalité de ne pas posséder de mémoire de trame. Il convient donc d'effectuer la conversion des objets en pixels et de calculer l'éclairement au rythme du balayage écran.

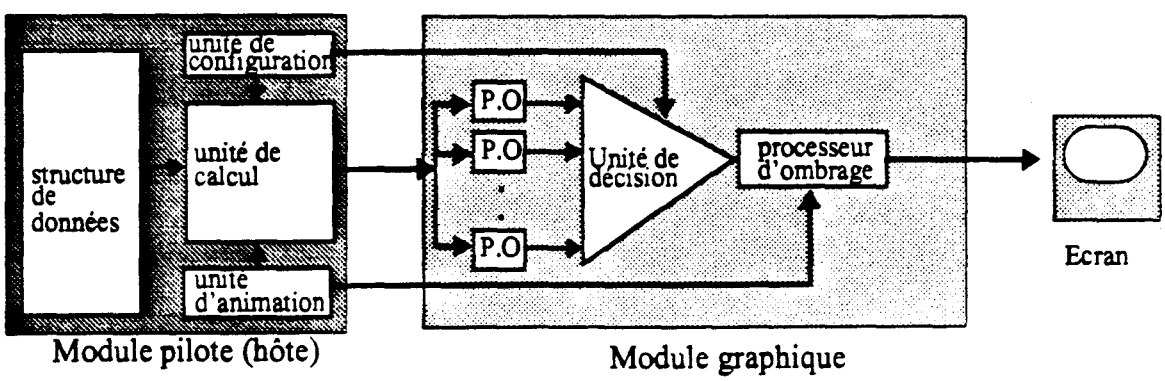


Figure 1•21 Architecture générale de la machine I.M.O.G.E.N.E.

Chaque processeur reçoit un morceau d'objet constitué à l'aide d'une primitive graphique qu'il traite. Les primitives de base de la machine sont les facettes comme dans tous systèmes graphiques évolués.

• Quand il s'agit de construire une sphère ou un cylindre en utilisant la primitive facette, il faut décomposer l'objet en un grand nombre d'éléments. Si le nombre de facettes est insuffisant la qualité des contours de l'objet et de son éclairement est fortement amoindrie, comme le montre la Figure 1•11 pour une sphère découpée en 128 facettes. Il est possible d'améliorer, à tout point de vue, la qualité de la représentation de l'objet en utilisant sa forme mathématique. [40]

Des travaux ont montré que les méthodes d'approximation par des fonctions du second degré n'étaient pas satisfaisantes. Pour obtenir un résultat correct, il faut calculer l'expression présentée au paragraphe 1•4. On doit donc calculer une racine carrée dans le processeur objet traitant les primitives sphère et cylindre. [25] [32]

Le nombre d'objets utilisés pour représenter la scène peut alors être réduit. Dans le cas de notre sphère, ce nombre est réduit d'un facteur[1] supérieur à 128 compensant largement l'accroissement de la complexité de la primitive. [29] [42] Un processeur objet quadrique est à l'étude. [41] Pour l'instant, la quadrique n'est pas une primitive couramment utilisée en modélisation. Pour pallier les besoins des programmes de modélisation, il faudrait prévoir des fonctions de Bézier. M. Froumentin propose donc des solutions permettant d'utiliser une primitive patch quadrique analogue à la primitive quadrique pour la modélisation. [20]

• Les pixels sont produits de façon synchrone et sont fournis à l'unité de décision pour l'élimination des parties cachées. L'unité de décision peut revêtir plusieurs configurations.

Premièrement, les processeurs objets peuvent être les feuilles d'un arbre équilibré. Tous les processeurs objets travaillent en même temps sur le même pixel et fournissent la profondeur et les autres paramètres relatifs à l'objet pour le pixel courant. L'unité de décision est alors formée par un arbre binaire dont chaque noeud est constitué d'un comparateur et d'un multiplexeur.

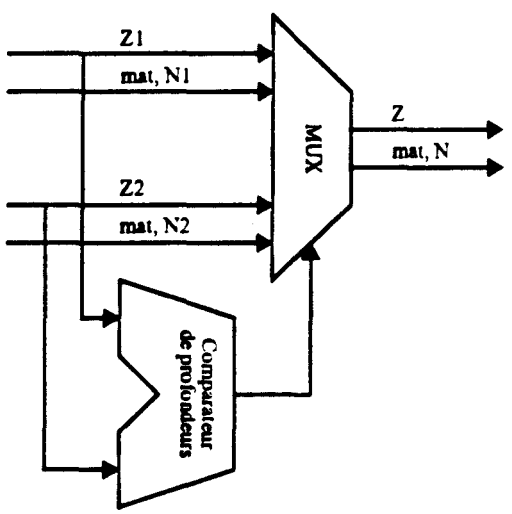


Figure 1•22 Vue synoptique d'un noeud de l'arbre de décision.

Deuxièmement, les processeurs objets peuvent être placés en pipeline. Chaque processeur dispose alors d'un morceau de l'unité de décision lui permettant de déterminer si l'objet qu'il construit, est plus proche de l'observateur que la profondeur qui lui est fournie par ces prédécesseurs dans le pipeline. S'il est plus proche, il passe ses paramètres. Dans le cas contraire, il passe ceux qui lui on été transmis.

1. La sphère de notre exemple est découpée en 128 facettes sans obtenir pour autant une qualité comparable à celle de la sphère que l'on obtient par les équations.

Dans la première solution, il est nécessaire de concevoir des VLSI disposant de beaucoup de broches et d'une partie opérative très restreinte. Economiquement, cette solution ne présente donc pas d'intérêt. Il serait trop coûteux de fabriquer un VLSI très volumineux pour une si petite unité opérative. Par contre la disposition des processeurs objets en pipeline n'induit pas cet inconvénient. Il suffit d'intégrer le comparateur et le multiplexeur avec la partie opérative du processeur objet. Il est nécessaire d'augmenter le nombre de broches du boîtier mais la taille conséquente du chips le permet. [28]

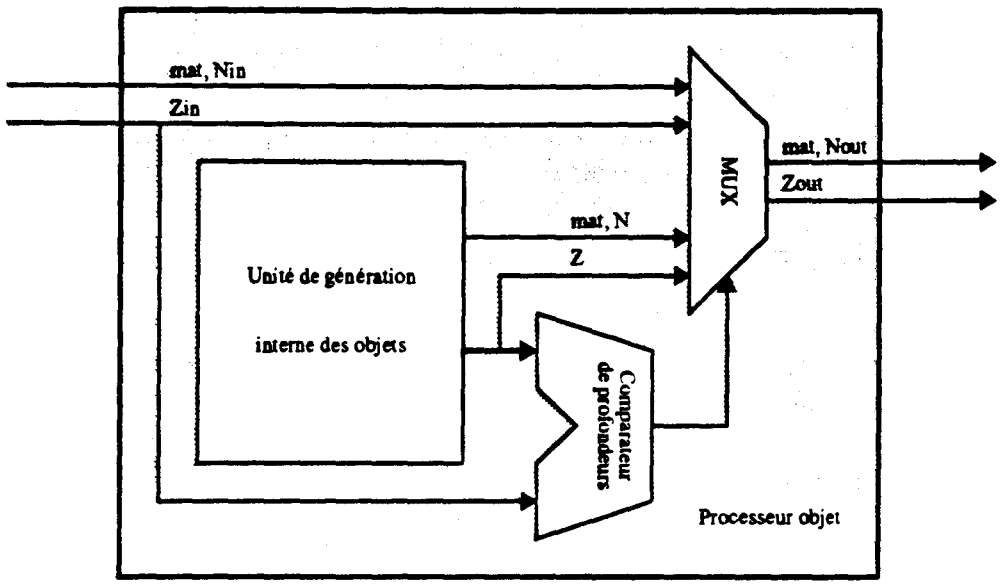


Figure 1•23 Disposition des processeurs objets en pipeline.

• Le système d'ombrage de la machine est constitué d'un ensemble de processeurs d'éclairement effectuant en parallèle les calculs pour chaque source. Il reçoit les données depuis l'unité de décision et les paramètres des sources depuis l'ordinateur hôte de la machine. Le système d'ombrage et son élaboration sont en grande partie l'objet de cette thèse. Les spécifications sont résumées dans le paragraphe suivant.

1•7 Cahier des charges.

1•7•1 Poser le problème.

Nous cherchons à définir, pour des machines de synthèse d'images spécialisées dans le temps réel, l'architecture d'une unité d'éclairement. La machine choisie pour cible de notre architecture est le projet I.M.O.G.E.N.E. que nous venons de présenter au paragraphe 1•6•2•3. Par rapport aux stations de travail existantes qui disposent déjà de ce genre d'unité, nous souhaitons apporter quelques caractéristiques supplémentaires.

Il faut que l'unité d'éclairement puisse fonctionner indépendamment des primitives d'affichage choisies. Les facettes ne doivent pas être les seules primitives que le système d'éclairement doivent pouvoir traiter.

Les images obtenues doivent être d'une qualité appréciable notamment vis à vis de l'éclairement spéculaire, sans qu'on soit obligé de diminuer la taille des facettes de la base de données.

1•7•2 L'intérêt des méthodes de Phong.

Sur la Figure 1•13, nous observons que l'ombrage de Gouraud peut donner des effets indésirables sur le rendu de l'éclairement spéculaire. Ce qui n'est pas le cas de la méthode d'ombrage de Phong. Pour répondre aux exigences du schéma de cette dernière méthode, il faut calculer le vecteur normal dans l'unité de conversion des objets en pixels. L'unité d'éclairement associée reçoit donc le vecteur directeur de la normale pour effectuer les calculs. Il est ainsi possible d'utiliser d'autres primitives d'affichage, comme la quadrique, dès lors que l'unité de conversion calcule à la fois la profondeur et le vecteur directeur de la normale. La méthode d'ombrage de Phong correspond donc parfaitement aux exigences que nous avons imposées.

1•7•3 Fixer les contraintes.

L'unité devant s'intégrer au sein de l'architecture du projet I.M.O.G.E.N.E. un certain nombre de contraintes sont imposées.

- L'unité d'éclairement peut utiliser les différentes données qui lui sont fournies par l'unité de conversion des objets en pixels. A savoir : la profondeur, la normale, les caractéristiques de la matière (de l'objet). Elle dispose aussi des paramètres de chaque source, qui lui sont fournis pour chaque image par l'ordinateur hôte de la machine.
- L'éclairement des pixels doit être calculé dans l'ordre du balayage écran. Le temps de calcul de l'éclairement d'un pixel doit être constant quelle que soit la nature de la source et quels que soient les paramètres concernant la matière de l'objet. Cette contrainte nous est imposée parce qu'I.M.O.G.E.N.E. ne dispose pas de mémoire de trame. Il faut donc que chaque pixel soit traité à la vitesse du balayage écran.
- La vitesse minimum de fonctionnement de l'unité d'éclairement doit permettre l'utilisation d'une résolution de 512 pixels fois 512 lignes en 25 images par seconde. Le temps maximal réservé pour le calcul d'un pixel est donc de 32 ns.
- On dispose entre deux images successives d'un laps de temps qui correspond au retour de trame. Il pourra être utilisé par l'unité d'éclairement pour effectuer toutes sortes d'initialisation. Le temps, plus restreint, du retour ligne pourra également servir, si c'est nécessaire. Toutes les techniques sont envisageables pour la réalisation dès lors qu'elles permettent d'atténuer le coût du post-processeur.

1•7•4 Les points devant être examinés.

Pour résoudre notre problème, nous allons d'abord examiner la complexité du calcul de l'éclairement suivant la position de la source et la position de l'observateur. Nous comparerons les formules d'éclairement de Phong et de Blinn et choisirons dans chacun des cas précédents la solution la plus avantageuse.

Grâce à cette première étude, nous connaîtrons la puissance requise par le calcul de l'éclairement et nous serons amenés à restreindre notre champ d'étude au type de sources les moins complexes, en vue d'une réalisation matérielle. Cet examen nous permettra, aussi de cerner les opérations requises les plus critiques.

Nous poursuivrons par une première proposition d'implémentation matérielle. Diverses propositions pour l'architecture de l'unité de normalisation des vecteurs seront présentées. Nous traiterons ensuite la composante spéculaire suivant plusieurs approches. Nous commencerons en présentant une solution simple mais aux possibilités réduites. Nous étudierons ensuite le moyen d'étendre les caractéristiques sans accroître la complexité de l'architecture dans des proportions significatives.

Enfin nous proposerons des méthodes d'approximation pour le calcul de l'élévation à la puissance et nous indiquerons comment obtenir un compromis avantageux entre les différentes approches. Nous terminerons notre première proposition d'implémentation en détaillant le reste des modules dont la difficulté est moindre.

Au cours de cette présentation, certains points ne pourront être réglés et des choix devront rester en suspens. Pour y remédier, nous effectuerons ensuite une étude mathématique de la transmission de l'erreur au cours des calculs. Cette étude aura pour objet principal l'unité de normalisation et elle montrera l'utilité d'introduire une unité de pré-normalisation. De plus, on pourra ainsi calculer la taille nécessaire pour chaque chemin de données. Le coût des unités opératives dépendant de la taille des données qu'elles manipulent, nous pourrons ainsi connaître le coût effectif de chaque solution et décider quelle est la meilleure parmi toutes celles présentées pour l'architecture de l'unité de normalisation.

Nous terminerons cette thèse, en complétant notre première proposition. Nous commencerons par présenter l'architecture de l'unité de pré-normalisation, puis nous examinerons l'unité de normalisation et nous chiffrerons la complexité de l'architecture utilisant la notation binaire naturelle et l'architecture utilisant la notation logarithmique signée. Enfin nous présenterons les solutions technologiques visant la réalisation matérielle de VLSIs pour l'unité d'éclairement.

CHAPITRE II: La complexité de l'éclairement de Phong et son optimisation.

Pour éclairer les objets d'une scène en synthèse d'images, il est possible d'utiliser différentes méthodes. Sur les facettes, nous avons présenté, outre l'ombrage constant, la méthode de Gouraud puis la méthode de Phong. En partant de cette dernière méthode, nous avons montré l'intérêt de déporter le calcul de l'éclairement après la conversion des objets en pixels.

Premièrement, il est nécessaire de calculer le vecteur de la normale en même temps que la profondeur dans l'unité de conversion des objets en pixels. Dans le cas des facettes le calcul de ce vecteur s'effectue de la même façon que le calcul des intensités R.V.B. pour l'ombrage de Gouraud. Dans le cas des quadriques, le calcul de chaque composante du vecteur est semblable au calcul de la profondeur. Les moyens permettant d'obtenir le vecteur de la normale sont connus dans le cas des facettes ou font l'objet d'autres travaux de recherche, dans le cas des quadriques [40]. Certaines stations de travail commerciales disposent d'unités spécialisées pour calculer l'ombrage de Gouraud. Le calcul du vecteur de la normale ne fait pas partie de cette thèse. Seules sa normalisation et la taille de ses composantes nous intéresseront.

Deuxièmement, le vecteur de la normale, la profondeur, la position de l'observateur, la position de la source et les caractéristiques de la matière sont utilisés pour obtenir l'éclairement.

Notre but est de construire une unité pour calculer l'éclairement après la conversion des objets en pixels. Pour choisir des solutions technologiques envisageables, la première difficulté à laquelle nous nous heurtons est l'estimation du coût en nombre d'opérations du calcul de l'éclairement.

Le coût dépend de la position de la source. Dans le cas où la source est rejetée à l'infini, les calculs sont moindres par rapport au cas où la source est située à distance finie de la scène. De même le coût dépend de la position de l'observateur. La projection orthogonale est moins coûteuse que la projection perspective. Au cours de ce chapitre, nous passerons en revue les différents cas, suivant les positions de la source et de l'observateur.

Pour calculer l'éclairement spéculaire, nous pouvons utiliser au choix la formule empirique de Phong ou celle de Blinn. Le rendu est sensiblement différent suivant la formule utilisée mais aucune des deux n'est jugée meilleure que l'autre. Le choix de la formule s'appuiera donc sur le coût qu'elle induit sur le calcul de l'éclairement. Ce choix sera remis en cause pour chacun des cas que nous étudierons.

Enfin, après avoir effectué l'estimation pour une source, nous étudierons à chaque fois les opérations qui peuvent être effectuées indépendamment du nombre de sources. Nous en déduirons une optimisation des calculs dans le cas de l'utilisation de plusieurs sources semblables.

2•1 Scène en projection orthogonale.

Dans le chapitre 1, nous avons mis en évidence deux situations dépendant de la position de l'observateur. On obtient la première situation quand l'observateur est rejeté à l'infini. Il faut alors procéder à une projection orthogonale de la scène. Lors de la seconde situation, on place l'observateur à une distance déterminée du plan de projection. Il faut alors procéder à une projection perspective. Outre les différences notables survenant lors de la transformation des objets en pixels, expliquées au chapitre précédent, cela influence la complexité du calcul de l'éclairement. Il convient donc d'étudier tour à tour les deux situations.

2•1•1 Rappel sur la projection orthogonale.

• Lors de la visualisation de la scène en projection orthogonale, on procède à l'élimination des parties cachées en utilisant une méthode produisant le même résultat que le "Z-buffer". (Cf §1•1•1•2). Le vecteur observateur est constant, quels que soient les paramètres de la source et quelle que soit l'image. Remarque : on oriente l'axe Z vers l'observateur.

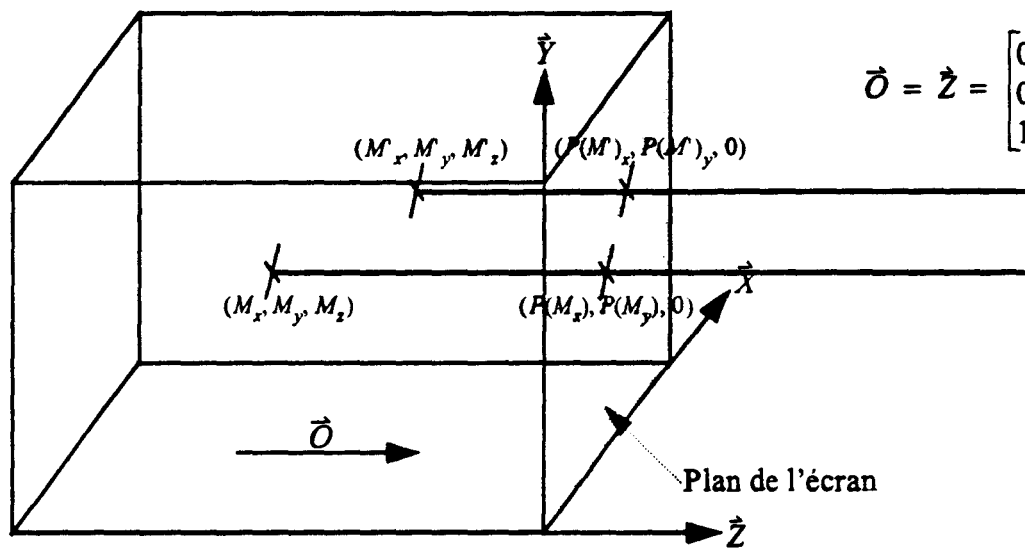


Figure 2•1 La projection orthogonale.

2•1•2 Avec sources de lumière à l'infinie.

2•1•2•1 L'algorithmique.

• Le vecteur normé allant du point éclairé vers la source lumineuse ne dépend pas de la position de l'objet mais seulement de la direction de la source, pour une même image. On ne doit recalculer ce vecteur que lorsque la source change de place. Dans le cas le moins favorable, il sera nécessaire de calculer à nouveaux ce vecteur à chaque image, donc 25 fois par seconde dans le cadre des hypothèses que nous nous sommes fixées. (Cf §1•7).

• Pour calculer l'éclairement diffus, on utilise la formule : $I_D = I \times K_D(\hat{N} \cdot \hat{L})$. Le vecteur $\hat{L}$ et l'intensité de la source I ne varient pas. Pour calculer l'éclairement spéculaire, on peut utiliser soit le vecteur réfléchi $\hat{R}$, soit le vecteur surbrillance $\hat{H}$.

• Le vecteur réfléchi $\hat{R}$ doit être dans le même plan que les vecteurs $\hat{L}$ et $\hat{N}$, et l'angle qu'il forme avec le vecteur $\hat{N}$ doit être le même que l'angle formé par les vecteurs $\hat{L}$ et $\hat{N}$. On en déduit donc les relations suivantes :

$$\left.\begin{array}{l}\vec{N}\wedge\vec{L}=\vec{R}\wedge\vec{N}\\ \vec{N}\cdot\vec{L}=\vec{N}\cdot\vec{R}\end{array}\right\}\Rightarrow\begin{bmatrix}0 & -N_x & N_y\\ N_z & 0 & -N_x\\ -N_y & N_x & 0\\ N_x & N_y & N_z\end{bmatrix}\cdot\begin{bmatrix}R_x\\ R_y\\ R_z\end{bmatrix}=\begin{bmatrix}(N_z\cdot L_y)-(N_y\cdot L_z)\\ (N_x\cdot L_z)-(N_z\cdot L_x)\\ (N_y\cdot L_x)-(N_x\cdot L_y)\\ (N_x\cdot L_x)+(N_y\cdot L_y)+(N_z\cdot L_z)\end{bmatrix}$$

Après simplification, on obtient la formule :

$\vec{R} = 2\vec{N}\cdot(\vec{N}\cdot\vec{L}) - (\vec{L})$; d'autre part $\|\vec{R}\| = \|\vec{L}\|$ si $\|\vec{N}\| = 1$

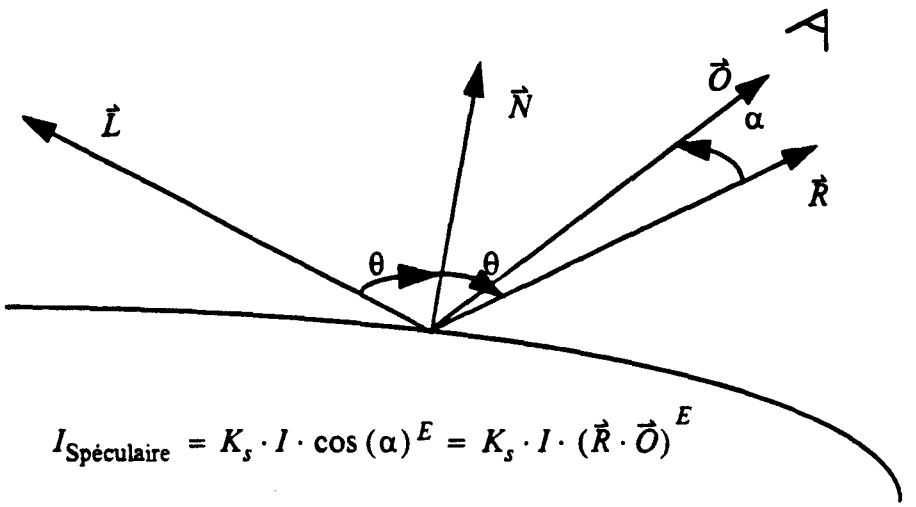


Figure 2•2 Illustration de l'utilisation du vecteur réfléchi.

• Le vecteur surbrillance $\vec{H}$ est le vecteur directeur de la bissectrice entre la direction de l'observateur et la direction de la source.

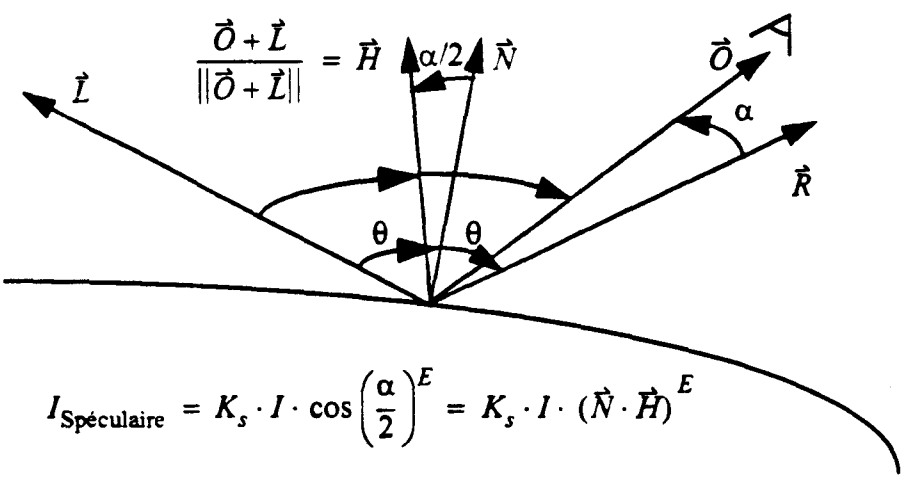


Figure 2•3 Illustration de l'utilisation du vecteur surbrillance.

Le vecteur $\vec{H}$ s'obtient en effectuant la somme des vecteurs $\vec{L}$ et $\vec{O}$ puis en normalisant le résultat. Les vecteurs $\vec{L}$ et $\vec{O}$ gardent la même valeur pour toute l'image, il en est donc de même pour $\vec{H}$ qui ne doit être calculé qu'une fois par image, soit dans le cas le moins favorable, 25 fois par seconde.

• De ces deux méthodes, la seconde est ici la plus avantageuse. Le calcul à effectuer entre deux images est minime, et on économise pour chaque pixel : un produit scalaire, une multiplication et une soustraction.

Pour la réalisation d'un processeur d'éclairement travaillant en projection orthogonale avec une source à l'infini, notre choix se portera donc sur l'utilisation du vecteur surbrillance $\vec{H}$.

2•1•2•2 Coût et optimisation pour une source.

• Essayons d'optimiser le calcul dans l'optique d'un seul processeur d'éclairement. On examine quelques transformations de la formule d'éclairement en vue de réduire le nombre d'opérations et leur complexité. Les images sont synthétisées en utilisant le modèle de couleur R-V-B, il faut donc effectuer un calcul pour chacune des composantes tri-chromatiques Rouge, Verte et Bleue. Les formules sont les suivantes :

$$I_{\text{Rouge}} = (I_{\text{Rouge}} \times K_{D\,\text{Rouge}}) \times \left[\frac{\vec{L}}{\|\vec{L}\|} \cdot \frac{\vec{N}}{\|\vec{N}\|}\right] + (I_{\text{Rouge}} \times K_{S\,\text{Rouge}}) \times \left[\frac{\vec{H}}{\|\vec{H}\|} \cdot \frac{\vec{N}}{\|\vec{N}\|}\right]^{E}$$

$$I_{\text{Vert}} = (I_{\text{Vert}} \times K_{D\,\text{Vert}}) \times \left[\frac{\vec{L}}{\|\vec{L}\|} \cdot \frac{\vec{N}}{\|\vec{N}\|}\right] + (I_{\text{Vert}} \times K_{S\,\text{Vert}}) \times \left[\frac{\vec{H}}{\|\vec{H}\|} \cdot \frac{\vec{N}}{\|\vec{N}\|}\right]^{E}$$

$$I_{\text{Bleu}} = (I_{\text{Bleu}} \times K_{D\,\text{Bleu}}) \times \left[\frac{\vec{L}}{\|\vec{L}\|} \cdot \frac{\vec{N}}{\|\vec{N}\|}\right] + (I_{\text{Bleu}} \times K_{S\,\text{Bleu}}) \times \left[\frac{\vec{H}}{\|\vec{H}\|} \cdot \frac{\vec{N}}{\|\vec{N}\|}\right]^{E}$$

• Pour optimiser le calcul, on va essayer de regrouper les opérations de façon à diminuer leur nombre.

Comme il n'y a qu'un seul processeur d'éclairement, il est avantageux de ne pas normaliser le vecteur $\vec{N}$ mais de seulement calculer la norme. Au lieu de diviser chaque composante du vecteur $\vec{N}$ par la norme, on effectue alors la division dans le processeur d'éclairement sur le résultat des deux produits scalaires. Ainsi, on économise une division.
On effectue le calcul en le décomposant de la manière suivante :

$$\left[\frac{\vec{L}}{\|\vec{L}\|} \cdot \frac{\vec{N}}{\|\vec{N}\|}\right] = \frac{\left(\frac{\vec{L}}{\|\vec{L}\|}\right) \cdot \vec{N}}{\|\vec{N}\|} \qquad \left[\frac{\vec{H}}{\|\vec{H}\|} \cdot \frac{\vec{N}}{\|\vec{N}\|}\right] = \frac{\left(\frac{\vec{H}}{\|\vec{H}\|}\right) \cdot \vec{N}}{\|\vec{N}\|}.$$

Pour le détail de l'algorithme voir paragraphe A•2•1 en annexe. (La légende et les explications concernant la présentation de cette annexe sont décrites au paragraphe A•1.) La complexité est résumée dans la Figure 2•4.

2•1•2•3 Optimisation pour plusieurs sources.

• Dans le cas de plusieurs sources et donc de plusieurs processeurs travaillant en parallèle ou en pipeline, l'optimisation proposée pour une source unique n'est plus valable. La méthode optimale devient celle de la première décomposition proposée.

L'unité de normalisation du vecteur $\hat{N}$ est indépendante du processeur d'éclairement. Elle est placée à la fin du pipeline du rendu après génération de la normale et fournit la valeur normée du vecteur $\hat{N}$ à tous les processeurs. Pour le détail de l'algorithme voir paragraphe A•2•2 en annexe. La complexité est résumée dans la figure ci-dessous.

Opérations	Une source unique			Plusieurs sources			
	Par image		Par pixel	Globales par pixel	Par source		
		Par matière			Par image		Par pixel
						Par matière	
Addition	5		9	2	5		10
Soustraction							
Multiplication		6	12			6	12
Elévation au carré	6		3	3	6		
Division / Inversion / Multiplication	6 2 6		2 1 2	3 1 3	6 2 6		
Racine carrée	2		1	1	2		
Elévation à la puissance			1				1

Figure 2•4 Détail des opérations utilisées dans les formules optimisées, pour le calcul de l'éclairement, avec projection orthogonale et sources à l'infini.

2•1•3 Avec sources de lumière à distance finie.

2•1•3•1 L'algorithmique.

• Dès lors que la source n'est plus placée à l'infini, le vecteur $\hat{L}$ n'est plus constant et doit donc être recalculé pour tous les pixels de l'écran. Soit un pixel pour lequel l'éclairement doit être calculé, soient (x,y) les coordonnées de ce pixel et z sa profondeur, c'est-à-dire la distance moyenne à l'écran de l'élément de surface visible qui se projette sur le pixel.

Remarque : la profondeur est une valeur négative. Soient (X_S, Y_S, Z_S) les coordonnées de la source lumineuse dans l'espace.

Le vecteur $\hat{L}$ s'obtient par la formule suivante :

$$\hat{L} = \frac{1}{\sqrt{(X_S-x)^2+(Y_S-y)^2+(Z_S-z)^2}} \times \begin{bmatrix}(X_S-x)\\(Y_S-y)\\(Z_S-z)\end{bmatrix} = \begin{bmatrix}X_L\\Y_L\\Z_L\end{bmatrix}$$

• Puisque le vecteur réfléchi $\hat{R}$ dépend de la valeur du vecteur $\hat{L}$, comme on l'a déjà indiqué dans le paragraphe 2•1•2•1, il faudra calculer sa valeur pour chaque pixel.

Rappelons la formule pour $\hat{R}$:

$$\vec{R} = 2\vec{N}\cdot(\vec{N}\cdot\vec{L}) - (\vec{L}) \qquad et \qquad ||\vec{R}|| = ||\vec{L}|| \qquad si \qquad ||\vec{N}|| = 1$$

• Le vecteur surbrillance $\hat{H}$, contrairement au cas de la source rejetée à l'infinie, varie lui aussi pour chaque pixel et doit être calculé à chaque fois avec la formule :

$$\vec{H} = \frac{\vec{L}+\vec{O}}{\|\vec{L}+\vec{O}\|} = \frac{1}{\sqrt{(X_L)^2+(Y_L)^2+(Z_L+1)^2}} \times \begin{bmatrix} (X_L) \\ (Y_L) \\ (Z_L+1) \end{bmatrix}$$

Pour calculer $\vec{H}$ il faut donc calculer le vecteur $(\vec{L}+\vec{O})$ puis le normaliser. Bien que $\vec{L}$ et $\vec{O}$ soient normés leur somme ne l'est pas, elle peut varier entre 0 et 2. En utilisant le fait que $\vec{L}$ est normé, on va heureusement pouvoir simplifier le calcul. En développant le calcul de $\|\vec{L}+\vec{O}\|^2$ puis en regroupant, on aboutit à l'expression suivante :

$$\|\vec{L}+\vec{O}\| = \sqrt{(X_L)^2+(Y_L)^2+(Z_L+1)^2} = \sqrt{2+2Z_L}$$

Cela permet d'économiser trois élévations au carré dans la normalisation de $(\vec{L}+\vec{O})$ en échange d'une multiplication par deux qui est simple à programmer ou à câbler.

2•1•3•2 Optimisation pour une source.

• Essayons d'optimiser le calcul dans l'optique d'un seul processeur d'éclairement. On propose quelques transformations des formules utilisées en vue de réduire le nombre d'opérations et la complexité de celles-ci.

Premièrement, on adopte la transformation qui a déjà fait ses preuves au paragraphe 2•1•2•2 pour réduire le nombre de divisions. Pour le calcul de $\vec{R}$, il est normalement nécessaire que $\vec{N}$ soit normé, mais il est toutefois possible de modifier la formule pour ne pas devoir normaliser $\vec{N}$ et économiser ainsi une division. Le calcul de $\vec{R}$ ne nécessite pas que $\vec{L}$ soit normé. On peut donc se contenter de calculer la norme de $\vec{L}$ puis on divise le résultat des deux produits scalaires par la norme de $\vec{L}$. La décomposition des calculs est la suivante :

$$\left[\frac{\vec{N}}{\|\vec{N}\|} \cdot \frac{\vec{L}}{\|\vec{L}\|}\right] = \frac{(\vec{N}\cdot\vec{L})}{\|\vec{N}\| \times \|\vec{L}\|} \qquad \left[\left(2\frac{\vec{N}}{\|\vec{N}\|} \times \left(\frac{\vec{N}}{\|\vec{N}\|} \cdot \frac{\vec{L}}{\|\vec{L}\|}\right) - \frac{\vec{L}}{\|\vec{L}\|}\right) \cdot \vec{O}\right] = \frac{2N_Z \times \frac{(\vec{N}\cdot\vec{L})}{\|\vec{N}\|^2} - L_Z}{\|\vec{L}\|}$$

Pour le détail de l'algorithme voir paragraphe A•2•3•1 en annexe.

• Quand on utilise le vecteur surbrillance $\vec{H}$, au lieu de normaliser $\vec{N}$ et $\vec{H}$, on calcule seulement les normes, puis on effectue les divisions après le calcul des produits scalaires. Pour le détail de l'algorithme voir paragraphe A•2•3•2 en annexe.

• Si le calcul de la racine carrée s'avère trop complexe, on peut en éliminer une en ne calculant plus la norme de $\vec{H}$ mais en calculant son carré et en modifiant la constante de l'élévation à la puissance. On substitue les formules suivantes dans la décomposition précédente du calcul :

$$NormeN2 = \|\vec{N}\|^2 \qquad NormeN = \sqrt{NormeN2} \qquad NormeH2 = 2+2Z_L$$

$$PS2 = \vec{N}\cdot\vec{H} \qquad Q = (PS2)^2 \qquad R = (NormeN2 \times NormeH2) \qquad p = \left(\frac{Q}{R}\right)^{E/2}$$

• Par cette méthode, on remplace l'extraction de la racine carrée par le calcul d'un carré. Les avantages et les inconvénients de cette transformation dépendent fortement des circonstances et plus précisément de la technologie utilisée. Si le calcul de l'éclairement devait être effectué par un ensemble de CPUs type DSP ou de microprocesseurs RISC, il est évident que le calcul du carré serait plus facile à programmer que l'extraction de la racine carrée. Dans un tel contexte la solution proposée serait donc attrayante.

Si par contre on désire synthétiser le calcul de la racine carrée et le calcul de l'élévation au carré en utilisant des ROMs, la complexité des deux opérateurs serait la même. La taille des ROMs ne dépend pas du type d'opération mais seulement de la taille des valeurs d'entrée et de sortie.

S'il s'agit de créer un VLSI, il existe une architecture, calculant une racine carrée, dont la complexité est de l'ordre de la moitié de celle d'un multiplieur.

Il est de toute façon nécessaire de concevoir une unité d'extraction de racine carrée pour calculer la norme de $\vec{N}$. Aussi, pour la réalisation d'un VLSI, cette méthode ne présente même pas l'intérêt d'économiser l'étude et la conception de l'unité d'extraction de racine carrée. Malgré tout, cette méthode pourrait s'avérer utile dès lors que l'architecture et la technologie mises en oeuvre valorisent l'élévation au carré plutôt que l'extraction de la racine carrée.

En comparant les opérations requises, pour la méthode utilisant le vecteur $\vec{R}$ et celle utilisant le vecteur $\vec{H}$, on se rend compte que cette dernière est de toute façon un peu plus coûteuse que l'autre (cf Figure 2•6).

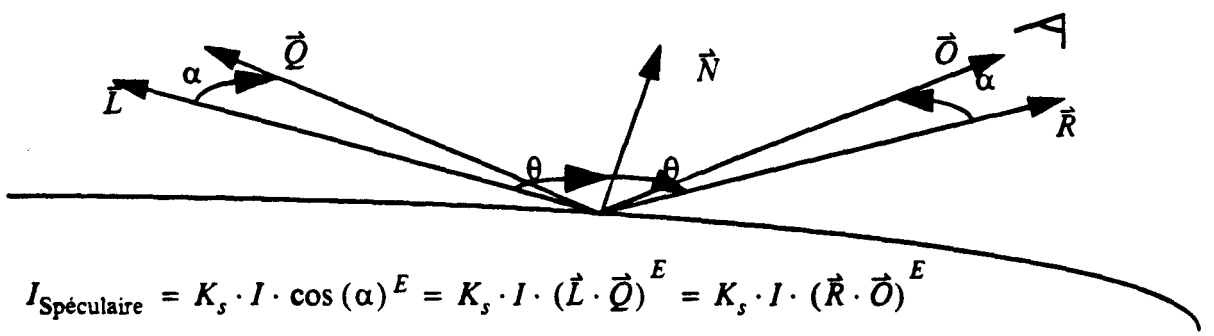
2•1•3•3 Optimisation pour plusieurs sources.

• Essayons d'optimiser le calcul dans l'optique de plusieurs processeurs d'éclairement travaillant en pipeline ou en parallèle. Dans ce cas, comme dans le cas de plusieurs processeurs d'éclairement avec source à l'infini, on ne doit normaliser $\vec{N}$ qu'une seule fois en utilisant une unité de normalisation extérieure aux processeurs d'éclairement.

Les vecteurs $\vec{R}$ et $\vec{H}$ dépendent de la position de la source c'est à dire du vecteur $\vec{L}$. On ne peut donc pas extraire du processeur d'éclairement une unité unique calculant l'un ou l'autre des deux vecteurs de manière unique pour toutes les sources. Dès lors que le nombre de sources est élevé, il est très intéressant de supprimer un module répété autant de fois qu'il y a de sources par un module même plus complexe mais qui ne soit présent qu'en un seul exemplaire. Pour le détail des deux algorithmes voir paragraphe A•2•4•1 et paragraphe A•2•4•2 en annexe

• On définit le vecteur $\vec{Q}$ tel que $\vec{Q} = 2 \times (\vec{N} \cdot \vec{O}) \times \vec{N} - \vec{O}$ qui ne dépend pas de la source mais dépend seulement de $\vec{N}$ et $\vec{O}$. Ce vecteur $\vec{Q}$ est le symétrique du vecteur $\vec{O}$ par rapport à $\vec{N}$, de la même façon que $\vec{R}$ est le symétrique du vecteur $\vec{L}$ par rapport à $\vec{N}$.

La figure suivante illustre comment utiliser ce vecteur $\vec{Q}$, nommé "vecteur réfléchi inverse", pour calculer l'éclairement spéculaire. On l'appelle ainsi puisqu'il indique le sens du rayon réfléchi pour un rayon incident fictif provenant de l'observateur.



$$I_{\text{Spéculaire}} = K_s \cdot I \cdot \cos(\alpha)^E = K_s \cdot I \cdot (\vec{L} \cdot \vec{Q})^E = K_s \cdot I \cdot (\vec{R} \cdot \vec{O})^E$$

Figure 2•5 Illustration de l'utilisation du vecteur réfléchi inverse.

L'égalité des deux produits scalaires est évidente dans le plan (cf Figure 2•5), mais pas lorsque l'on travaille dans l'espace; en voici la démonstration pour le cas général.

$$\vec{R} \cdot \vec{O} = [2 \times (\vec{N} \cdot \vec{L}) \times \vec{N} - \vec{L}] \cdot \vec{O} = 2 \times (\vec{N} \cdot \vec{L}) \times (\vec{N} \cdot \vec{O}) - (\vec{L} \cdot \vec{O}) = \vec{L} \cdot [2 \times (\vec{N} \cdot \vec{O}) \times \vec{N} - \vec{O}] = \vec{L} \cdot \vec{Q}$$

• Outre le fait d'utiliser un vecteur à la place d'un autre, on ne change pratiquement rien aux calculs qu'il faut effectuer. Pour le détail de l'algorithme voir paragraphe A•2•4•3 en annexe. Il faut cependant noter que, pour le cas qui nous intéresse (celui de la projection orthogonale), le vecteur $\vec{\partial}$ se réduit à une composante unitaire en Z, ce qui permet de simplifier le calcul du produit scalaire $\vec{R}\cdot\vec{\partial}$. Il n'est pas possible d'effectuer une telle simplification pour le produit scalaire $\vec{L}\cdot\vec{Q}$. En comparant les coûts des trois méthodes proposées, on voit que celle qui utilise le vecteur $\vec{R}$ reste la plus intéressante.

Le tableau suivant récapitule le coût des différentes solutions.

Opérations par pixel	Un processeur		Plusieurs processeurs					
			Globale			Par source		
	Avec $\vec{R}$	Avec $\vec{H}$	Avec $\vec{R}$	Avec $\vec{H}$	Avec $\vec{Q}$	Avec $\vec{R}$	Avec $\vec{H}$	Avec $\vec{Q}$
Addition	9	12	2	2	2	10	13	12
Soustraction	4	3			1	4	3	3
Multiplication	11	11			2	10	10	12
Elévation au carré	6	6	3	3	4	3	3	3
Division / Inversion / Multiplication	3 / 3 / 3	5 / 3 / 5	3 / 1 / 3	3 / 1 / 3	3 / 1 / 3	2 / 1 / 2	3 / 2 / 3	2 / 1 / 2
Racine carrée	2	3	1	1	1	1	2	1
Elévation à la puissance	1	1				1	1	1
Décalage fixe	1	1			3	1	1	

Figure 2•6 Récapitulatif des coûts des différentes solutions dans le cas des sources à distance finie en projection orthogonale.

2•2 Scène en projection perspective.

Les coûts relatifs aux différents types de sources lumineuses ayant été examinés dans la situation de la projection orthogonale, il faut maintenant considérer les méthodes en prenant en compte la projection perspective.

2•2•1 Rappel sur la projection perspective.

• Lors de la visualisation de la scène en projection perspective, on procède à l'élimination des parties cachées en utilisant un système produisant le même résultat que le "Z'-buffer". Voir paragraphe 1•1•1•3.

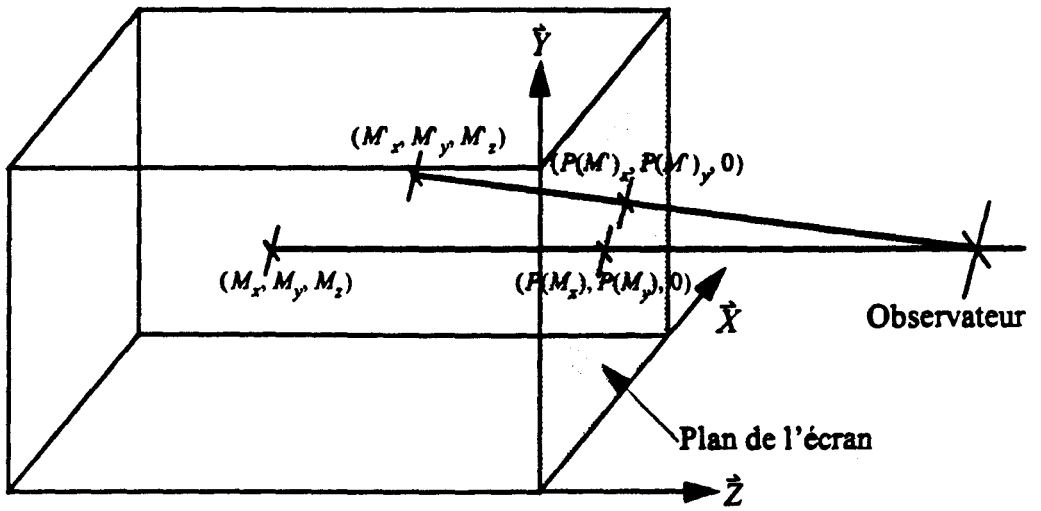


Figure 2•7 Illustration de la projection perspective.

• Contrairement à la situation de la projection orthogonale, le vecteur observateur n'est plus constant. Il varie en fonction de la distance entre l'observateur et l'écran de projection ainsi qu'en fonction de la position du pixel. Par contre, il est toujours indépendant de la scène pour un pixel donné de l'écran.

Comme le vecteur $\vec{O}$ n'intervient pas dans le calcul du diffus, la complexité sera la même que celle déjà calculée pour le cas de l'observateur rejeté à l'infini. Par contre, peu importe que l'on utilise le vecteur réfléchi $\vec{R}$ ou le vecteur surbrillance $\vec{H}$, le vecteur $\vec{O}$ intervient toujours dans le calcul de l'éclairement spéculaire.

Les composantes du vecteur $\vec{O}$ s'obtiennent par la formule suivante :

$$\vec{O} = \frac{1}{\sqrt{(X-X_o)^2 + (Y-Y_o)^2 + (Z_o)^2}} \times \begin{bmatrix} (X-X_o) \\ (Y-Y_o) \\ Z_o \end{bmatrix}$$

Pour normaliser le vecteur $\vec{O}$, il faut calculer l'expression $(X-X_o)^2 + (Y-Y_o)^2 + (Z_o)^2$ puis en extraire la racine carrée et enfin diviser successivement $X-X_o$, $Y-Y_o$ et Z par le résultat obtenu.

Cette expression est du second degré en X et Y et peut donc être calculée de façon incrémentale par un DDA du second degré. Cela permet d'économiser trois élévations au carré en les remplaçant par quelques additions.

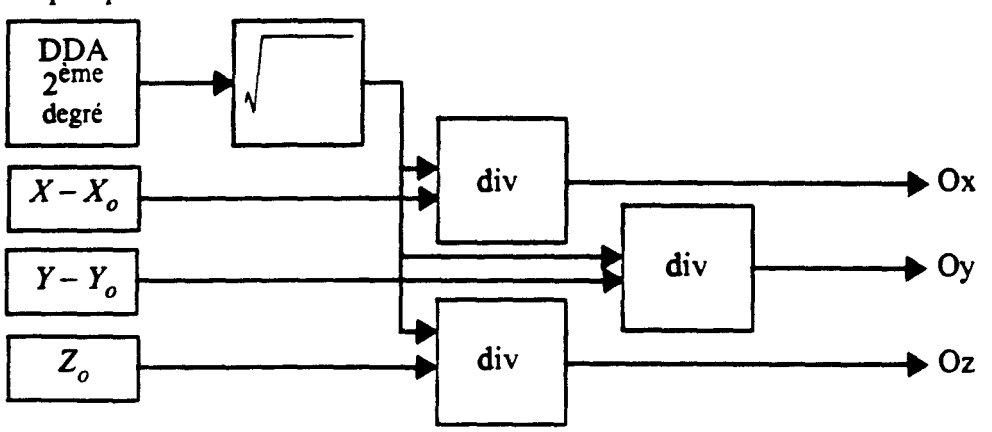


Figure 2•8 Schéma bloc pour l'obtention du vecteur observateur normé.

2•2•1•1 Programmation d'un DDA du second degré.

• Pour établir le programme du DDA, on va calculer les accroissements en X et Y de la fonction $f(X,Y) = \|\vec{O}\|^2 = (X-x_0)^2 + (Y-y_0)^2 + z_0^2$.

Les relations suivantes nous donnent les accroissements en X :

$\delta_X(X) = f(X+1, Y) - f(X, Y) = (X+1)^2 + x_0^2 - 2 \times x_0 \times (X+1) - X^2 - x_0^2 + 2 \times x_0 \times X$

$\delta_X(X) = 2 \times X + 1 - 2 \times x_0$

$\delta_{XX}(X) = \delta_X(X+1) - \delta_X(X) = 2 \times (X+1) + 1 - 2 \times x_0 - 2 \times X + 2 \times x_0 - 1$

$\delta_{XX}(X) = 2$

Les variables X et Y ayant des rôles symétriques, on en déduit les accroissements en Y :

$\delta_Y(Y) = 2 \times Y + 1 - 2 \times y_0$ et $\delta_{YY}(Y) = 2$.

• A partir de ces relations, on écrit le programme suivant :

```
Début
    X ← 0
    Y ← 0
    Fy ← x0²+y0²+z0²
    δX ← 1 - 2x0
    δY ← 1 - 2y0
    Pour chaque ligne Faire
            Fx ← Fy
        Pour chaque pixel de la ligne Faire
            Fx ← Fx + δX
            δX ← δX + 2
            Utilisation de Fx
        FinPour
            Fy ← Fy + δY
            δY ← δY + 2
    FinPour
Fin
```

• Le calcul de $\|\vec{O}\|^2$ ne réclame que 2 additions par pixel et 2 additions supplémentaires pour chaque ligne de l'écran.

2•2•2 Avec sources de lumière à l'infini.

2•2•2•1 L'algorithmique.

• Pour calculer l'intensité de la lumière diffuse, on n'utilise pas le vecteur $\vec{O}$. Les résultats obtenus précédemment dans le cas de la projection orthogonale restent donc valables pour la composante diffuse de l'éclairement. La seule question consistera à déterminer s'il faut normaliser le vecteur $\vec{N}$ ou bien s'il faut diviser le produit scalaire $\vec{N} \cdot \vec{L}$ par $\|\vec{N}\|$.

• Quelle que soit la méthode choisie pour calculer l'intensité spéculaire, il faut toujours que le vecteur $\vec{O}$ soit calculé (cf Figure 2•9). D'autre part, les vecteurs $\vec{R}$, $\vec{H}$ ou $\vec{Q}$ dépendent tous trois d'au moins un vecteur qui n'est pas constant pour la totalité de l'image. On ne peut donc plus choisir une méthode parce qu'elle réduit la complexité en utilisant un vecteur constant (Cf §2•1•2•1).

Méthodes		$\vec{O}$	$\vec{L}$	$\vec{N}$	Equations
$\vec{R}$	Dépend de		X	X	$\vec{R} = 2(\vec{N}\cdot\vec{L})\times\vec{N}-\vec{L}$
	Produit scalaire avec	X			$(\vec{R}\cdot\vec{O})$
$\vec{H}$	Dépend de	X	X		$\vec{H} = (\vec{O}+\vec{L})/\|\vec{O}+\vec{L}\|$
	Produit scalaire avec			X	$(\vec{H}\cdot\vec{N})$
$\vec{Q}$	Dépend de	X		X	$\vec{Q} = 2(\vec{N}\cdot\vec{O})\times\vec{N}-\vec{O}$
	Produit scalaire avec		X		$(\vec{Q}\cdot\vec{L})$

Figure 2•9 Tableau de dépendance en fonction de la méthode choisie.

2•2•2•2 Le coût de la mise en oeuvre.

• Dans le but d'optimiser les calculs avant de choisir la méthode la plus avantageuse, on va transformer les formules de chaque méthode.

Dans la méthode utilisant le vecteur $\vec{R}$, on peut optimiser le calcul et supprimer 2 divisions en calculant seulement la norme de $\vec{O}$ sans normaliser le vecteur mais en divisant le résultat du produit scalaire $(\vec{R}\cdot\vec{O})$ par cette norme. Pour le détail de cet algorithme, référencé (1) dans la suite, voir paragraphe A•3•1•1 en annexe.

Dans la méthode utilisant le vecteur surbrillance, pour optimiser les calculs, on peut transformer les formules : $\vec{H} = \left(\frac{\vec{L}}{\|\vec{L}\|}\right) + \frac{\vec{O}}{\|\vec{O}\|}$ $PS = \frac{(\vec{H}\cdot\vec{N})}{\|\vec{H}\|\times\|\vec{N}\|}$ $I_S = (I\times K_S)(PS)^E$

Pour le détail de l'algorithme référencé (2) dans la suite, qui utilise l'expression classique de la norme d'un vecteur pour calculer $\vec{H}$, voir paragraphe A•3•1•2 en annexe.

Pour le détail de l'algorithme référencé (3) dans la suite, qui utilise la formule : $\|\vec{H}\| = 2\left[\|\vec{O}\| + \vec{O}\cdot\frac{\vec{L}}{\|\vec{L}\|}\right]$, voir paragraphe A•3•1•3 en annexe.

En procédant ainsi, il n'est plus nécessaire de normaliser $\vec{O}$. Il faut seulement calculer la norme de $\vec{O}$. L'introduction de cette méthode devrait permettre de gagner quelques opérations.

• Remarque : De la même façon que $\|\vec{O}\|^2$ peut s'obtenir par un calcul incrémental, $(\vec{O}\cdot\vec{L})$ peut aussi être calculé d'une manière incrémentale grâce à un DDA. Cela permet de supprimer les multiplications du produit scalaire.

• Pour établir le programme du DDA, on va calculer les accroissements en X et Y de la fonction $f(X,Y) = (\vec{O}\cdot\vec{L}) = (X-x_0)\times x_L + (Y-y_0)\times y_L + z_0\times z_L$.

Les relations suivantes nous donnent les accroissements en X et en Y :

$\delta_X = f(X+1, Y) - f(X, Y) = x_l$ et $\delta_Y = f(X, Y+1) - f(X, Y) = y_l$.

En connaissant la largeur de l'écran, c'est à dire la longueur d'une ligne, on peut calculer l'accroissement en Y qui prend en compte le mouvement en X du retour ligne. L'observateur étant situé au milieu de l'écran, la longueur d'une ligne est $2x_O$.

On obtient donc : $\delta_{\text{retour ligne}} = y_l + 2x_O \times x_l$. qui a une valeur constante.

• Le calcul de $[\vec{O} \cdot (\vec{L}/\|\vec{L}\|)]$ ne réclame qu'une addition par pixel, une addition supplémentaire pour chaque ligne de l'écran et enfin pour chaque image, une multiplication et une addition.

• Dans la méthode utilisant le vecteur $\vec{Q}$ on peut optimiser le calcul et supprimer 2 divisions en calculant seulement la norme de $\vec{O}$ sans normaliser le vecteur mais en divisant le résultat du produit scalaire $(\vec{Q} \cdot \vec{L})$ par cette norme.

Pour le détail de l'algorithme référencé (4) dans la suite, voir paragraphe A•3•1•4 en annexe.

• Il n'y a pas de grande différence entre la méthode utilisant le vecteur réfléchi $\vec{R}$ et celle utilisant le vecteur réfléchi inverse $\vec{Q}$. La complexité est sensiblement la même (il y a juste un produit scalaire en plus), mais en considérant le parallélisme, la méthode avec $\vec{R}$ devrait permettre d'effectuer le calcul de $\vec{R}$ en même temps que le calcul de la norme de $\vec{O}$. Pour choisir l'une ou l'autre des méthodes il faut comparer leurs coûts.

Opérations par pixel	(1)	(2)	(3)	(4)
Addition	11	16	16	13
Soustraction	5	2	2	5
Multiplication	15	16	16	18
Elévation au carré	3	6	3	3
Division / Inversion / Multiplication	3 / 3 / 3	2 / 2 / 2	2 / 2 / 2	3 / 3 / 3
Racine carrée	2	3	2	2
Elévation à la puissance	1	1	1	1
Décalage fixe	1			1

Figure 2•10 Comparaison des coûts suivant les méthodes, quand l'observateur est à distance finie et la source à distance infinie.

2•2•2•3 Optimisation pour une source.

• Pour le cas d'une source unique, en lisant le tableau ci-dessus (cf Figure 2•10), on se rend compte que la méthode (4) est plus compliquée que la (1). De même la (3) est plus compliquée que la (2). Enfin, le choix de la méthode (3) semble plus judicieux que celui de la méthode (1). Elle permet d'économiser une division même si elle requiert globalement une opération de type addition soustraction et une multiplication. Une division coûte plus chère qu'une soustraction et une multiplication réunies.

2•2•2•4 Optimisation pour plusieurs sources.

• Essayons d'optimiser le calcul dans l'optique de plusieurs processeurs d'éclairement travaillant en pipeline ou en parallèle. Comme dans tous les cas où il y a plusieurs processeurs d'éclairement, on ne doit normaliser $\hat{N}$ qu'une seule fois en utilisant une unité de normalisation extérieure aux processeurs d'éclairement.

• Dès lors que le nombre de sources est élevé, il est très intéressant de remplacer un module, répété autant de fois qu'il y a de sources, par un module, même plus complexe, mais qui ne soit présent qu'en un seul exemplaire.

Comme nous l'avons déjà évoqué au paragraphe 2•1•3•3, le vecteur $\vec{O}$ est indépendant du vecteur $\hat{L}$ et donc de la source. Le calcul du vecteur $\vec{O}$ et sa norme seront donc calculés par un module unique extérieur aux processeurs d'éclairement comme cela a déjà été fait pour normaliser $\hat{N}$.

• De nouveau, il s'impose de choisir entre les quatre méthodes présentées. Pour optimiser chacun de ces algorithmes pour l'utilisation de plusieurs sources, il convient d'extraire les instructions communes à chaque source. Pour la première, la seconde et la troisième méthode, seuls les calculs de $\vec{O}$, $\|\vec{O}\|$ et $\hat{N}/\|\hat{N}\|$ peuvent être factorisés. Tous les autres doivent être effectués par chaque processeur. Dans le cas de la méthode (4), le vecteur $\vec{Q}$ peut être calculé et normalisé une seule fois pour tous les processeurs d'éclairement. En conséquence, cela va permettre de réduire le coût de cette méthode.

Compte tenu des modifications, le détail de cet algorithme, référencé désormais (1') dans la suite, peut être visé au paragraphe A•3•2•1 en annexe. Pour l'algorithme référencé (2'), voir paragraphe A•3•2•2 en annexe. Suite aux améliorations, l'algorithme référencé (3') utilise la formule suivante lors du calcul de $\|\hat{H}\|$: (voir paragraphe A•3•2•2 en annexe.)

$$\left[\frac{\hat{H}}{\|\hat{H}\|}\cdot\frac{\hat{N}}{\|\hat{N}\|}\right] = \frac{\hat{H}\cdot\dfrac{\hat{N}}{\|\hat{N}\|}}{2\left[1+\left(\dfrac{\vec{O}}{\|\vec{O}\|}\cdot\dfrac{\hat{L}}{\|\hat{L}\|}\right)\right]}$$

Pour l'algorithme référencé (4'), voir le descriptif au paragraphe A•3•2•4 en annexe.

• Pour choisir l'une ou l'autre des méthodes, il faut comparer leurs coûts opératoires. A la lumière de la Figure 2•10, on voit que la méthode (4') est la plus avantageuse.

Opérations par pixel et par source	(1')	(2')	(3')	(4')
Addition	10	15	16	10
Soustraction	3			
Multiplication	15	16	15	12
Elévation au carré		3		
Division / Inversion / Multiplication		1 1 1	1 1 1	
Racine carrée		1		
Elévation à la puissance	1	1	1	1
Décalage fixe	1		1	

Figure 2•11 Comparaison des coûts suivant les méthodes, quand l'observateur est à distance finie et les sources à distance infinie.

2•2•2•5 Récapitulatif des coûts.

• Le tableau ci-dessous récapitule la complexité du calcul de l'éclairement dans le cas d'une projection perspective avec une ou plusieurs sources à l'infini.

Opérations	Une source unique			Plusieurs sources			
	Par image		Par pixel	Globales par pixel	Par source		
		Par matière			Par image	Par matière	Par pixel
Addition	2		14	6+(2/lg)	2		10
Soustraction			2	5			
Multiplication		6	16	6		6	12
Elévation au carré	3		3	3	3		
Division / Inversion / Multiplication	3 1 3		2 2 2	6 2 6	3 1 3		
Racine carrée	1		2	2	1		
Elévation à la puissance			1				1
Décalage fixe			1	1			

Figure 2•12 Détail des opérations utilisées dans les formules optimisées, pour le calcul de l'éclairement, avec projection perspective et sources à l'infini.

2•2•3 Avec sources de lumière à distance finie.

2•2•3•1 L'algorithmique.

• Dès lors que la source n'est plus placée à l'infini, le vecteur $\vec{L}$ n'est plus constant et doit donc être recalculé pour tous les pixels de l'écran. On reprend donc la formule présentée dans le paragraphe 2•1•3•1. Soit un pixel, de coordonnées (x, y), pour lequel l'éclairement doit être calculé, soit le point M, de coordonnées (X_M, Y_M, Z_M), le centre de l'élément de surface visible se projetant sur le pixel choisi. Soient (X_S, Y_S, Z_S) les coordonnées de la source lumineuse dans l'espace.

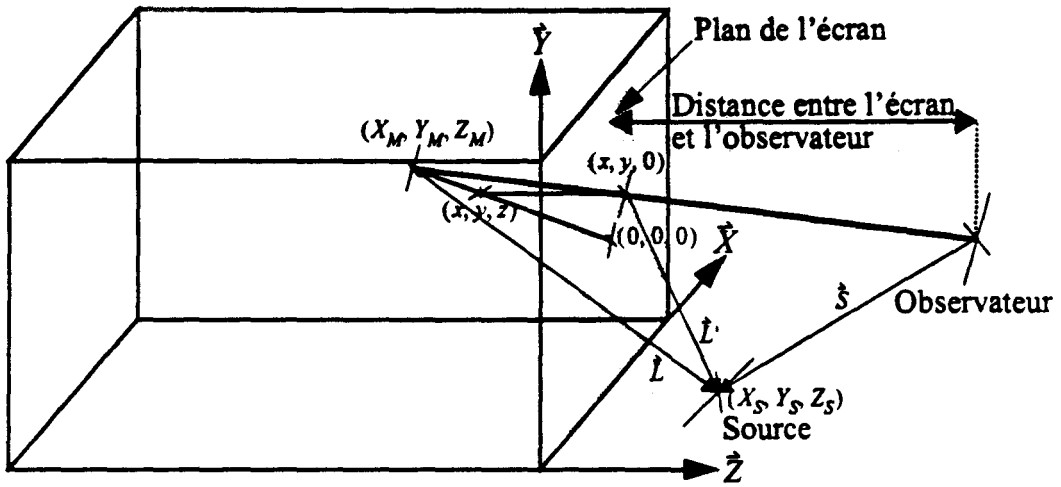


Figure 2•13 Illustration de la projection perspective avec source à distance finie.

Le vecteur $\vec{L}$ s'obtient par la formule suivante :

$$\vec{L}_{\text{normé}} = \frac{1}{\sqrt{(X_S - X_M)^2 + (Y_S - Y_M)^2 + (Z_S - Z_M)^2}} \times \begin{bmatrix} (X_S - X_M) \\ (Y_S - Y_M) \\ (Z_S - Z_M) \end{bmatrix} = \frac{1}{\|\vec{L}\|} \times \begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} = \frac{\vec{L}}{\|\vec{L}\|}$$

• On ne dispose pas en chaque pixel de l'écran des coordonnées (X_M, Y_M, Z_M).

Le rendu des objets en perspective est effectué par la technique de la transformation perspective. Cette transformation associe à chaque point M de coordonnées (X_M, Y_M, Z_M), un point M' de coordonnées (x, y, z). Cette transformation plaçant l'observateur à l'infini dans l'espace virtuel, l'élimination des parties cachées peut être effectuée par un système de "Z-buffer" comme dans le cas de la projection orthogonale.

Soit D, la distance entre l'observateur et l'écran. Les formules suivantes donnent les correspondances entre les coordonnées dans l'espace transformé et les coordonnées dans l'espace réel : $\left\{ X_M = \frac{D \times x}{D - z} \qquad Y_M = \frac{D \times y}{D - z} \qquad Z_M = \frac{D \times z}{D - z} \right\}$

Le vecteur $\vec{L}$ s'obtient par la formule suivante :

$$\vec{L}_{\text{normé}} = \frac{1}{\sqrt{(X_S - X_M)^2 + (Y_S - Y_M)^2 + (Z_S - Z_M)^2}} \times \begin{bmatrix} (X_S - X_M) \\ (Y_S - Y_M) \\ (Z_S - Z_M) \end{bmatrix} = \frac{1}{\|\vec{L}\|} \times \begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} = \frac{\vec{L}}{\|\vec{L}\|}$$

$$\vec{L}_{\text{normé}} = \frac{1}{\sqrt{\left(X_S - \frac{D \times x}{D - z}\right)^2 + \left(Y_S - \frac{D \times y}{D - z}\right)^2 + \left(Z_S - \frac{D \times z}{D - z}\right)^2}} \times \begin{bmatrix} \left(X_S - \frac{D \times x}{D - z}\right) \\ \left(Y_S - \frac{D \times y}{D - z}\right) \\ \left(Z_S - \frac{D \times z}{D - z}\right) \end{bmatrix}.$$

Cette manière de calculer le vecteur $\vec{L}_{\text{normé}}$ requiert en chaque pixel, outre une normalisation, quatre soustractions, trois multiplications et trois divisions. Cela s'avère fort coûteux. En manipulant les formules ci-dessus, on peut établir la relation vectorielle suivante : $(D-z)\cdot\vec{L} = z\cdot\vec{S} + D\cdot\vec{L}'$.

On a donc $\vec{L}_{\text{normé}} = \frac{\vec{L}}{\|\vec{L}\|} = \frac{(D-z)\cdot\vec{L}}{\|(D-z)\cdot\vec{L}\|} = \frac{(z\cdot\vec{S} + D\cdot\vec{L}')}{\|z\cdot\vec{S} + D\cdot\vec{L}'\|}$. En utilisant cette relation, le calcul du vecteur $\vec{L}_{\text{normé}}$ nécessite en chaque pixel une normalisation, trois additions, deux soustractions, six multiplications. Cette façon de procéder offre déjà l'avantage de remplacer les trois divisions par trois multiplications. Mais, on va pouvoir faire encore mieux, car le terme $D\cdot\vec{L}'$ peut être calculé de façon incrémentale.

On a $D\cdot\vec{L}' = \begin{bmatrix} D\times(X_S - x) \\ D\times(Y_S - y) \\ D\times Z_S \end{bmatrix} = \begin{bmatrix} (D\times X_S) - (D\times x) \\ (D\times Y_S) - (D\times y) \\ D\times Z_S \end{bmatrix}$

Pour simplifier les notations, appelons désormais $\vec{L}$ le vecteur $z\cdot\vec{S} + D\cdot\vec{L}'$.

- A partir de là, on écrit facilement le programme calculant les composantes du vecteur $\vec{L}$:

```
Début
    X ← 0
    Y ← 0
    Fx ← D × Y_S
    Fy ← D × Y_S
    Fz ← D × Z_S
    Pour chaque ligne Faire
        Pour chaque pixel de la ligne Faire
            Lx ← Fx + [z × (X_S − X_o)]
            Ly ← Fx + [z × (Y_S − Y_o)]
            Lz ← Fz + [z × (Z_S − D)]
            Utilisation de L
            Fx ← Fx - D
        FinPour
            Fx ← Fx + (2D × X_o)
            Fy ← Fy - D
    FinPour
Fin
```

- Le calcul de $\vec{L}$ réclame, par pixel, trois additions, une soustraction et trois multiplications et pour chaque ligne de l'écran une addition et une soustraction supplémentaires.

2•2•3•2 Le coût de la mise en oeuvre.

• Dans le but d'optimiser les calculs avant de choisir la méthode la plus avantageuse, on va une fois encore transformer les formules de chaque méthode.

Pour la méthode utilisant le vecteur $\hat{R}$, on pourra viser l'algorithme référencé (1), au paragraphe A•3•3•1, en annexe

En ce qui concerne la méthode utilisant le vecteur surbrillance, afin d'optimiser les calculs, deux solutions sont envisageables suivant la manière employée pour calculer $\|\hat{H}\|$. On pourra regarder l'algorithme référencé (2), au paragraphe A•3•3•2, et l'algorithme référencé (3) au paragraphe A•3•3•3 en annexe.

Pour la méthode utilisant le vecteur $\hat{Q}$, l'algorithme référencé (4) pourra être vu au paragraphe A•3•3•4 en annexe.

Afin d'opter pour choisir l'une ou l'autre des méthodes, il faut comparer leurs coûts.

Opérations	(1)	(2)	(3)	(4)
Addition	16	21	21	15
Soustraction	6	3	6	5
Multiplication	20	18	16	20
Elévation au carré	6	9	6	6
Division / Inversion / Multiplication	3 / 3 / 3	5 / 3 / 5	5 / 3 / 5	3 / 3 / 3
Racine carrée	3	4	3	3
Elévation à la puissance	1	1	1	1
Décalage fixe.	1			1

Figure 2•14 Comparaison des coûts suivant les méthodes, quand l'observateur et la source sont à distance finie.

2•2•3•3 Optimisation pour une source.

• Pour le cas d'une source unique, le tableau ci-dessus (cf Figure 2•10), montre que la méthode (4) est la plus économique.

2•2•3•4 Optimisation pour plusieurs sources.

• Essayons d'optimiser le calcul dans l'optique de plusieurs processeurs d'éclairement travaillant en pipeline ou en parallèle. Comme dans tous les cas où il y a plusieurs processeurs d'éclairement, on ne doit normaliser $\hat{N}$ qu'une seule fois en utilisant une unité de normalisation extérieure aux processeurs d'éclairement.

• Dès lors que le nombre de sources est élevé, il est très intéressant de remplacer un module, répété autant de fois qu'il y a de sources, par un module unique , même plus complexe.

Comme nous l'avons déjà dit, le vecteur $\hat{O}$ est indépendant de la source. Le calcul du vecteur $\hat{O}$ et sa norme seront donc calculés par un module unique extérieur aux processeurs d'éclairement comme cela a déjà été fait pour normaliser $\hat{N}$.

• De nouveau, il s'impose de faire un choix entre les méthodes (1), (2), (3) et (4). En utilisant la méthode (1), mis à part $\vec{\partial}$, $\|\vec{\partial}\|$ et $\hat{N}/\|\hat{N}\|$, chaque processeur d'éclairement doit tout calculer car le vecteur $\hat{R}$ dépend de la source. Les conclusions sont les mêmes pour les méthodes (2) et (3), mis à part $\vec{\partial}$, $\|\vec{\partial}\|$ et $(\hat{N}/\|\hat{N}\|)$, tous les calculs doivent être effectués par chaque processeur. Dans le cas de la méthode (4), le vecteur $\vec{Q}$ peut être calculé et normalisé une seule fois pour tous les processeurs d'éclairement. Cela va permettre de réduire le coût de cette méthode.

• Compte tenu des modifications, les formules utilisées pour chacune des méthodes, sont détaillées ci-dessous.

Avec la méthode utilisant le vecteur $\hat{R}$, on pourra viser l'algorithme référencé (1'), au paragraphe A•3•4•1, en annexe.

Avec celle utilisant le vecteur surbrillance, pour optimiser les calculs, deux solutions sont envisageables suivant la manière utilisée pour calculer $\|\hat{H}\|$. On pourra viser l'algorithme référencé (2'), au paragraphe A•3•4•2, et l'algorithme référencé (3') au paragraphe A•3•4•3 en annexe.

En ce qui concerne la méthode utilisant le vecteur $\vec{Q}$, l'algorithme référencé (4') pourra être visé au paragraphe A•3•4•4 en annexe.

Afin de choisir l'une ou l'autre des méthodes, comparons leurs coûts opératoires.

Opérations	(1')	(2')	(3')	(4')
Addition	15	20	21	15
Soustraction	4	1	1	1
Multiplication	18	15	18	12
Elévation au carré	3	6	3	3
Division / Inversion / Multiplication	2 / 1 / 2	4 / 2 / 4	4 / 2 / 4	2 / 2 / 2
Racine carrée	1	2	1	1
Elévation à la puissance	1	1	1	1
Décalage fixe.	1		1	

Figure 2•15 Comparaison des coûts suivant les méthodes, quand l'observateur et les sources sont à distance finie.

A la lumière du tableau ci-dessus, on voit que la méthode (4') est la plus avantageuse. Par rapport à la méthode (1'), elle permet l'économie de 3 soustractions, de 6 multiplications et d'un décalage fixe. Les méthodes (2') et (3') réclament principalement des coûts accrus soit au niveau des racines carrées soit au niveau des divisions.

• Le tableau ci-dessous récapitule la complexité du calcul de l'éclairement dans le cas d'une projection perspective avec une ou plusieurs sources à distance finie.

Opérations	Une source unique			Plusieurs sources			
	Par image		Par pixel	Globales	Par source		
		Par matière			Par image		Par pixel
						Par matière	
Addition			13	6+(2/lg)			9
Soustraction			8	5			3
Multiplication		6	16	6		6	12
Elévation au carré			6	3			3
Division / Inversion / Multiplication			5 / 3 / 5	6 / 2 / 6			2 / 2 / 2
Racine carrée			3	2			1
Elévation à la puissance			1				1
Décalage fixe			1	1			

Figure 2•16 Détail des opérations utilisées dans les formules optimisées, pour le calcul de l'éclairement, avec projection perspective et sources à distance finie.

2•3 Synthèse et conclusion.

2•3•1 La puissance requise.

• Comme le montre chaque tableau figurant dans les paragraphes 2•1 et 2•2, le calcul de l'éclairement réclame une puissance de calcul assez importante. Pour réduire les calculs plusieurs méthodes ont été tour à tour étudiées. Aucune ne s'est avérée la plus intéressante quelles que soient les positions de la source et de l'observateur.

Nombre d'opérations par pixel										
Observateur à distance :	Source à distance :	Méthode	Addition	Soustraction	Multiplication	Elév. au carré	Inversion	Div. ou Mult.	Racine carrée	Elév. à la puis.	Décalage fixe
infinie	infinie	$\vec{H}$	9	0	12	3	1	2	1	1	0
infinie	finie	$\vec{R}$	9	4	11	6	3	3	2	1	1
finie	infinie	$\vec{Q}$	16	2	16	3	2	2	2	1	0
finie	finie	$\vec{Q}$	15	5	20	6	3	3	3	1	1

Figure 2•17 Récapitulatif des coûts pour une source.

• Le tableau précédant récapitule les coûts de l'éclairement suivant la position de la source et la position de l'observateur dans le cas d'une source unique. A chaque fois la méthode qui a été choisie comme étant la plus intéressante, est indiquée.

Le tableau suivant nous fournit les informations quand on utilise plusieurs sources (de même type).

Nombre d'opérations par pixel

Observ. à distance :	Sources	Méthode		Addition	Soustraction	Multiplication	Elév. au carré	Inversion	Div. ou Mult.	Racine carrée	Elév. à la puis.	Décalage fixe
infinie	infinie	$\hat{H}$	globale	2	0	0	3	1	3	1	0	0
			par source	10	0	12	0	0	0	0	1	0
infinie	finie	$\hat{R}$	globale	2	0	0	3	1	3	1	0	0
			par source	10	4	10	3	1	2	1	1	1
finie	infinie	$\vec{Q}$	globale	6	5	6	3	2	6	2	0	0
			par source	9	4	11	6	0	0	1	1	0
finie	finie	$\vec{Q}$	globale	6	5	6	3	2	6	2	0	1
			par source	9	3	12	3	2	2	1	1	0

Figure 2•18 Récapitulatif des coûts pour plusieurs sources.

2•3•2 Les opérations et fonctions souhaitées.

Pour calculer l'éclairement, il est nécessaire de disposer d'opérations qui sont relativement simples comme les additions, les soustractions et les décalages d'un nombre fixé de bits. D'autres opérations telles que les élévations au carré et les multiplications vont s'avérer coûteuses. Aussi, il sera important de déterminer de façon rigoureuse la taille des données que celles ci devront manipuler car cela aura une influence notable sur leur complexité. L'utilisation de la division pourra être contournée en décomposant le calcul en une multiplication et une inversion. L'extraction de la racine carrée et l'inversion vont poser quelques problèmes bien que ce soient des opérateurs unaires. La nécessité de limiter la taille des données utilisées par ces deux opérateurs va être encore plus grande qu'elle ne l'était pour la multiplication. Mais, le problème le plus délicat à résoudre sera celui de l'élévation à la puissance.

CHAPITRE III: Proposition d'implémentation matérielle.

La méthode d'ombrage de Phong permet, par rapport à celle de Gouraud , d'obtenir des images de qualité accrue. Elle permet aussi de déporter le calcul de l'éclairement après la transformation des objets en pixels et l'élimination des parties cachées. Nous avons montré au chapitre précédent que le calcul de l'éclairement est très gourmand en nombre d'opérations et requiert donc une puissance de calcul importante. Cette puissance dépend de différents facteurs et il résulte, de l'étude menée, un coût plus abordable quand l'observateur et les sources sont rejetés à l'infini. L'utilisation du vecteur surbrillance est alors optimum pour calculer l'intensité de lumière spéculaire. C'est dans le cadre de cette limitation des caractéristiques des sources que nous proposerons une implémentation matérielle.

D'abord, nous introduirons brièvement une architecture générale de l'unité de post-éclairement permettant l'obtention de la puissance requise pour un nombre de sources que l'on pourra moduler. Ensuite nous étudierons tour à tour les modules qui composent l'unité d'éclairement en portant un intérêt plus important sur les unités les plus stratégiques.

Parmi celles-ci, l'unité de normalisation attirera en premier lieu notre attention. Nous vérifierons donc la nécessité d'introduire une telle unité puis nous présenterons une première architecture de cette dernière. Nous essayerons ensuite de tirer parti des avantages de la "notation logarithmique signée" pour définir une seconde architecture. Pour choisir entre ces deux propositions, il faut prendre en compte des développements mathématiques, que nous examinerons au chapitre suivant. Nous déciderons donc ultérieurement.

Puis, nous nous intéresserons au problème central du calcul de l'éclairement spéculaire : l'élévation à la puissance. Nous passerons d'abord en revue les méthodes théoriques permettant de calculer une élévation à la puissance. Nous étudierons les techniques et les restrictions acceptables des performances permettant d'obtenir l'intensité spéculaire pour un faible coût. Enfin, nous en déduirons une architecture pour calculer l'intensité spéculaire.

Afin de ne pas rester prisonnier des limites de notre solution, nous examinerons les possibilités d'accroître, de façon significative, les performances pour un coût dérisoire. Parmi ces limitations, la précision des calculs lors de l'élévation à la puissance n'est pas suffisante pour permettre l'utilisation d'exposant spéculaire supérieur à 50. Il résultera de cette étude la présentation d'un module optionnel permettant l'utilisation par notre architecture d'exposants spéculaires plus élevés. Nous orienterons ensuite l'examen du calcul de l'éclairement spéculaire en étudiant des méthodes d'approximation quadratique puis rationnelle pour l'élévation à la puissance. Nous terminerons notre étude de l'éclairement spéculaire en proposant des solutions permettant de tirer avantage des deux approches du problème.

Ce chapitre se terminera en regroupant les unités précédemment exposées et en y adjoignant les unités pour les calculs d'intensité de lumière diffuse et des produits scalaires. Certains choix nécessitant une étude mathématique resteront momentanément sans réponse. Cela concerne notamment tous les problèmes de largeur des chemins de données qui font l'objet du chapitre suivant.

3•1 Rappel de l'architecture générale.

Le schéma général de l'architecture pour réaliser un post-éclairement de Phong est, rappelons-le, le suivant : Le module de conversion des objets en pixels fournit un vecteur de la normale, une profondeur et les caractéristiques de la matière de l'objet concerné. La première opération du post-éclairement consiste à normaliser le vecteur qui sera ensuite distribué à chaque unité d'éclairement. Il faudra en compter une par source. Au sein de chaque unité d'éclairement, on trouvera un bloc servant au calcul du diffus et un autre concernant le calcul du spéculaire. Dans chacun des blocs, on trouvera une unité de calcul du produit scalaire.

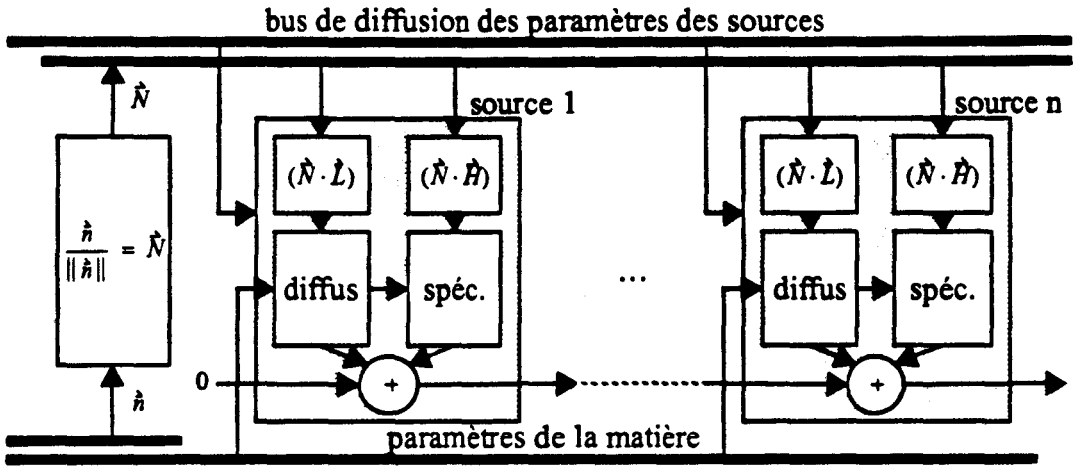


Figure 3•1 Architecture générale.

Afin de faciliter la réalisation, nous nous sommes limités à l'étude d'une unité d'éclairement fonctionnant pour des sources à l'infini. Les vecteurs $\hat{L}$ et $\hat{H}$ intervenant dans les produits scalaires sont donc constants. Aucune unité spécifique servant au calcul de ces vecteurs pour chaque pixel ne doit donc être ajoutée à la description précédente.

3•2 L'Unité de Normalisation.

L'étude de cette architecture commence par la réalisation de l'unité de normalisation. Celle-ci est la première rencontrée dans le pipeline du post-éclairement et c'est aussi l'une des plus coûteuses.

3•2•1 Nécessité de l'opération de Normalisation.

On peut discuter de la nécessité de la normalisation dans le cas de l'utilisation de primitives facettes et dans le cas de primitives plus complexes comme la quadrique.

3•2•1•1 La primitive facette.

• Pour une facette triangulaire avec interpolation de Phong du vecteur $\hat{N}$, on suit la méthode telle que nous l'avons présentée dans le paragraphe 1•3•2. Si la facette est petite la normalisation du vecteur n'est pas nécessaire. En l'omettant on accélère l'exécution, (ou dans notre cas, on diminue la complexité) mais cela diminue la qualité visuelle de l'image ainsi que nous en avons déjà fait état au paragraphe 1•3•3•1.

3•2•1•2 La primitive quadrique.

• Si nous prenons le cas des sphères, la norme de $\vec{N}(x,y)$ est constante pour un objet mais dépend du rayon. On peut donc fixer la valeur de la norme en ajustant les coefficients utilisés dans les fonctions $f_1(x,y)$ et $f_2(x,y)$. Par contre, si nous prenons le cas d'un ellipsoïde (voir Figure 3•2.), la norme de $\vec{N}(x,y)$ peut prendre ses valeurs dans un très grand intervalle au cours de la génération de l'objet. En l'occurrence, sur l'exemple, la norme est comprise entre 2^{12} et 2^{17}. En utilisant ainsi la forme simplifiée pour calculer les composantes du vecteur de la normale, lors de la génération de quadriques, la norme du vecteur $\vec{N}(x,y)$ peut donc varier dans de grandes proportions.

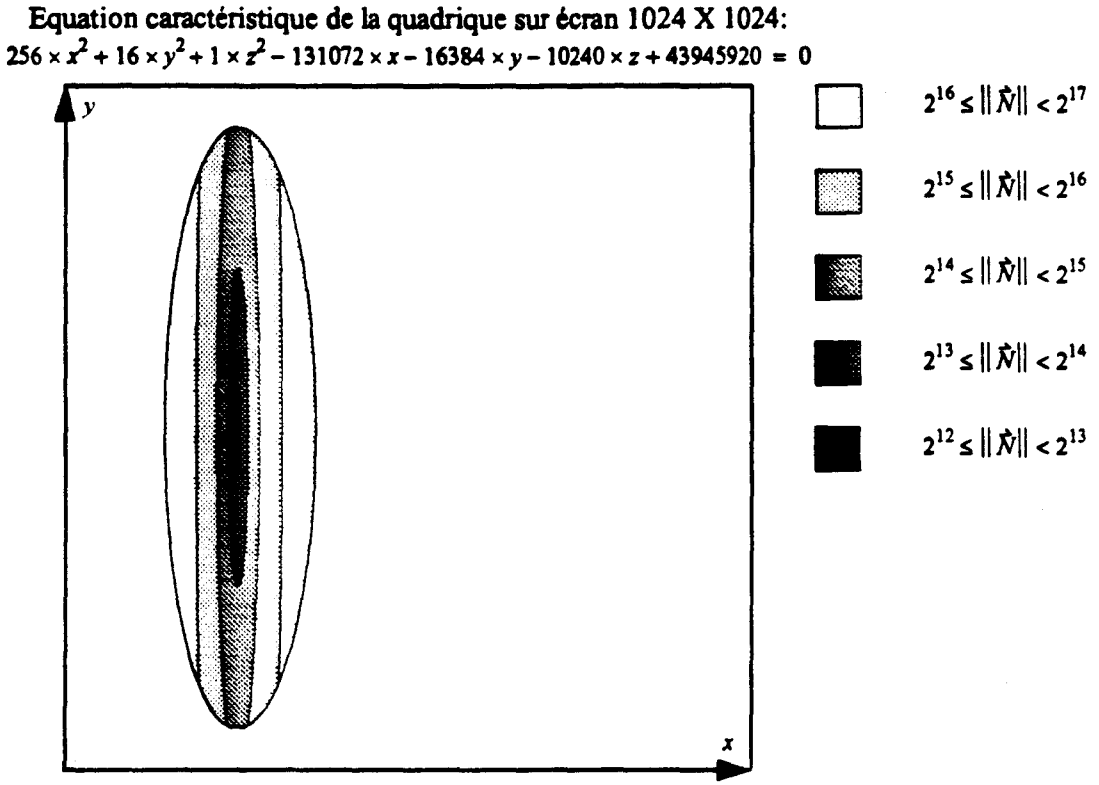


Figure 3•2 Exemple de variation de la norme du vecteur de la normale dans le cas d'un ellipsoïde.

• On démontre ainsi la nécessité d'avoir une unité de normalisation pour les vecteurs $\vec{N}(x,y)$ provenant d'un objet quadrique. Cette unité se place entre les unités de conversion des objets en pixels et les unités d'éclairement. (Voir Figure 3•1.)

3•2•2 Le coût de la Normalisation.

• Comme nous en avons déjà fait état au paragraphe 2•1•2•2 et au paragraphe 2•1•2•3, l'opération de normalisation du vecteur $\hat{N}$ nécessite : deux additions, trois élévations au carré, une extraction de racine carrée et trois divisions.

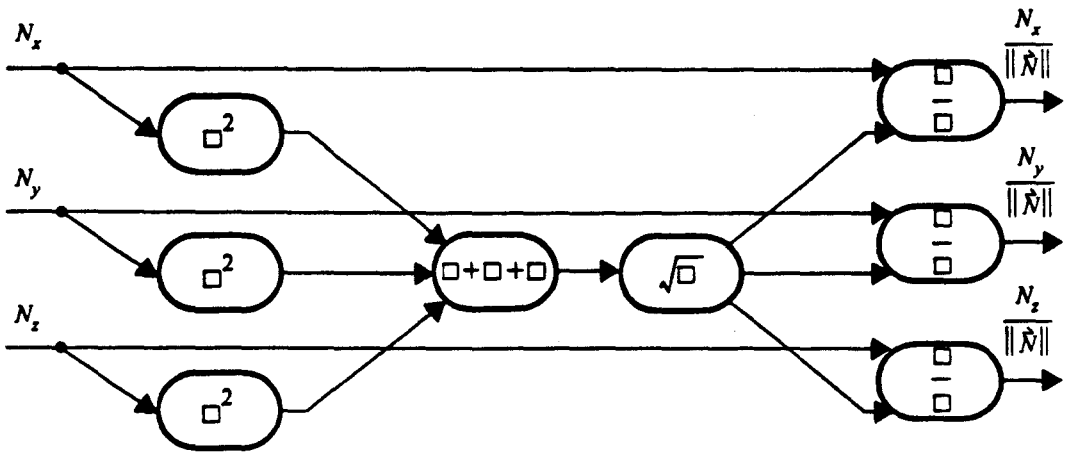


Figure 3•3 Diagramme de la normalisation.

• Lors de la réalisation matérielle, le coût de l'unité de normalisation va dépendre de la taille des données d'entrées et de sorties, c'est à dire du nombre de bits sur lesquels seront codées les valeurs de N_x, N_y, N_z, $(N_x/\|\hat{N}\|)$, $(N_y/\|\hat{N}\|)$ et $(N_z/\|\hat{N}\|)$.

Les opérations d'élévation au carré, d'extraction de la racine carrée et de division sont particulièrement coûteuses. Leur complexité dépend aussi de la taille des données internes qu'elles manipulent.

3•2•3 Architecture de l'unité de normalisation.

Pour concevoir l'architecture de l'unité de normalisation deux solutions s'offrent à nous. La première consiste simplement à réaliser une implémentation en suivant le diagramme tel qu'il est présenté au paragraphe précédent. L'autre solution consiste à modifier le diagramme afin d'optimiser la réalisation des unités servant au calcul des divisions. L'optimisation que l'on se propose d'étudier utilise les logarithmes et une autre forme de codification des nombres : la notation S./L.N.S..

3•2•3•1 Le diagramme de base.

• La figure suivante présente les différentes étapes de calcul intervenant dans la normalisation du vecteur $\hat{N}$. Chacune d'entre elles peut être attribuée à une unité. Mais il est aussi possible de regrouper le calcul de la racine carrée et l'inversion au sein d'une même unité; ou bien, on peut regrouper l'inversion et la multiplication en une seule unité de division.

• Le choix entre ces différentes solutions dépendra de la complexité et de la technologie utilisée pour la réalisation de l'unité de normalisation. Mais les répercussions de ce choix sur la taille des mots devront aussi entrer en ligne de compte.

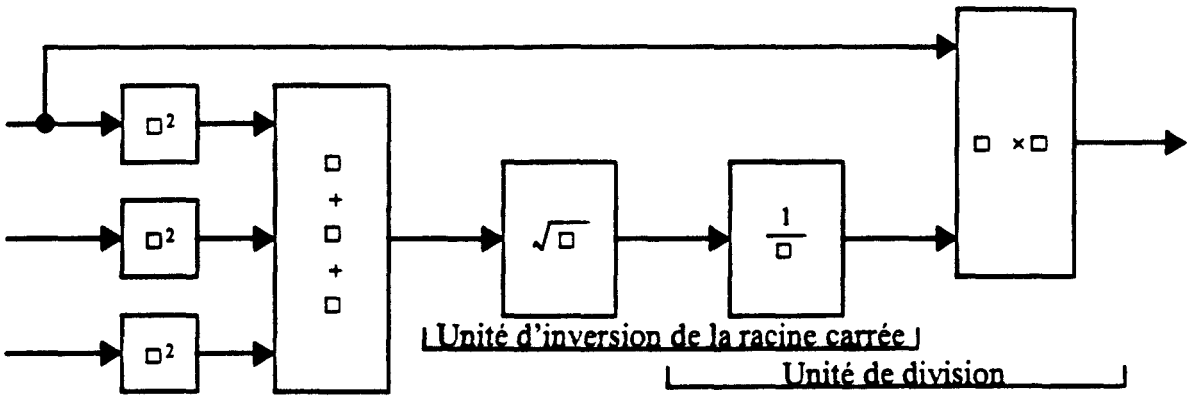


Figure 3•4 Diagramme de la normalisation d'une composante du vecteur de la normale.

3•2•3•2 L'utilisation de la notation logarithmique signée (S./L.N.S.). (Sign/Logarithm Number System [52])

• Différentes équipes de recherche se sont penchées sur l'utilisation de la S./L.N.S. pour faciliter les opérations de multiplications, de divisions et d'extractions de la racine carrée. Le principe consiste à séparer le signe de la valeur absolue, puis à transformer cette dernière en son logarithme. La multiplication des valeurs absolues s'obtient ainsi par l'addition des logarithmes. La soustraction remplace la division et le décalage à droite, l'extraction de la racine carrée. Cette notation ne permet plus de faire des additions et des soustractions. Il faut donc convertir les nombres codés en S./L.N.S., en nombres codés en binaire naturel avant toute opération d'addition ou de soustraction.

Les opérations de conversion d'un format dans un autre sont complexes. Pour les réaliser, il est nécessaire d'utiliser des ROMs servant de tables de conversion. L'utilisation de la S./L.N.S. pour calculer l'élévation au carré n'a donc aucun sens. On remplacerait une ROM par deux ROMs et un décaleur à gauche. La première ROM servirait pour la conversion vers le S./L.N.S. et la seconde restituerait le résultat en binaire naturel avant l'addition.

• Par contre, l'utilisation de la S./L.N.S., pourrait s'avérer avantageuse pour supprimer les unités de multiplication en les remplaçant par des additionneurs. Il faut que les valeurs présentées en entrées de l'unité, effectuant la multiplication des valeurs en binaire naturel, soient en S./L.N.S.

• Il faut donc d'abord convertir en S./L.N.S. le résultat de l'unité d'inversion de la racine carrée. Pour effectuer cette conversion, il suffit que le contenu de la ROM soit modifié de telle façon que la valeur fournie soit codée en S./L.N.S.

• Il faut ensuite que les composantes du vecteur de la normale soient transformées en S./L.N.S. Cela nécessitera trois ROMs pour effectuer ces conversions. Enfin, il n'est pas nécessaire de transformer le résultat en binaire naturel car le vecteur $\hat{N}$ intervient ensuite uniquement dans des produits scalaires au sein des processeurs d'éclairement. L'opération suivante est donc une multiplication que les processeurs d'éclairement pourront ici effectuer en profitant de la S./L.N.S.

• L'opération d'extraction de la racine carrée s'effectue par un simple décalage en codage S./L.N.S. L'inversion s'obtient en prenant le complément à deux du nombre L.N.S.

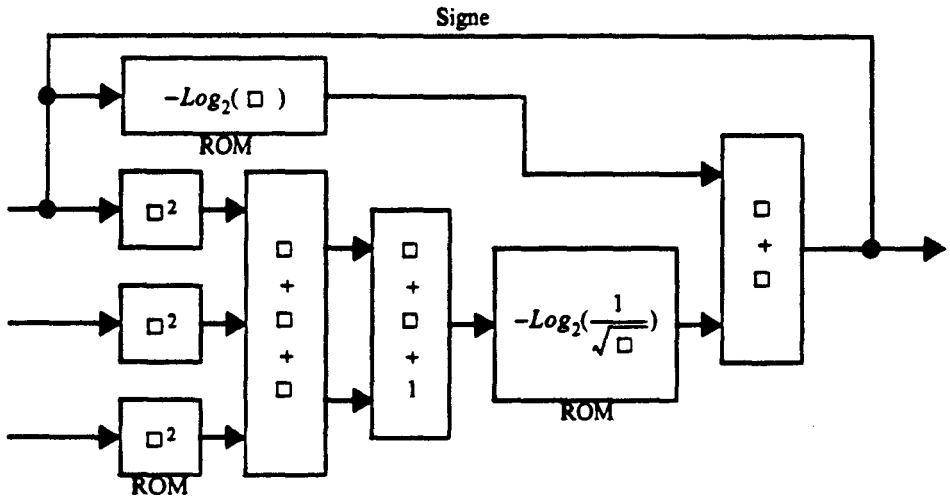


Figure 3•5 Diagramme de l'unité de normalisation avec S/L.N.S. (en version de base)

3•2•4 Conclusion

Suite à cette étude, on propose différentes solutions parmi lesquelles un choix sera effectué ultérieurement au chapitre 5 quand les formats des données utilisées auront été définis par l'étude théorique que l'on mènera au chapitre 4.

Voyons maintenant l'autre problème important de l'architecture : le calcul de l'éclairement spéculaire.

3•3 La composante spéculaire.

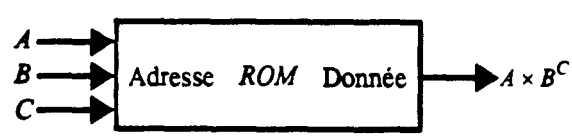
Des rayonnements diffus et spéculaire, c'est ce dernier le plus complexe. Outre un produit scalaire et quelques multiplications, qui figurent aussi dans le calcul du diffus, il nécessite une élévation à la puissance par le coefficient spéculaire de la matière. Ce n'est pas une opération simple à câbler ni même à calculer.

3•3•1 Les solutions de base.

3•3•1•1 Utilisation d'une ROM

• Outre le calcul préalable d'un produit scalaire, le calcul du spéculaire utilise un opérateur U défini par :

$$U : \begin{cases} \Re^3 \rightarrow \Re \\ A, B, C \rightarrow A \times B^C \end{cases}$$



Cet opérateur pourrait être implémenté sous forme d'une ROM fournissant un résultat pour chaque triplet d'entrée. Cette solution a l'avantage d'être très simple mais par contre sa conception matérielle n'est pas très réaliste. On réduit un peu la complexité en séparant la fonction exponentielle de la multiplication. La taille de la ROM est ainsi plus raisonnable.

- C'est ce modèle qu'U. Claussen propose d'utiliser pour la machine PROOF[9].

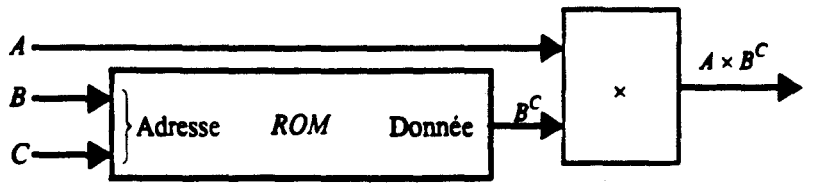


Figure 3•6 Diagramme du calcul d'une composante du rayonnement spéculaire

La taille d'une telle ROM est encore très exagérée et serait d'un coût prohibitif. Supposons que la valeur d'entrée soit inscrite sur 12 bits, l'exposant sur 8 bits, autorisant des exposants de 0 à 255, et le résultat sur 8 bits, il faut alors une ROM ou un ensemble de ROMs d'une capacité totale de 1 Méga-octet.
En composants discrets, les ROMs disponibles sur le marché ont des temps d'accès qui ne correspondent pas à nos besoins. De plus la capacité nominale des ROMs n'est pas très élevée comparée à celle de certaines RAM.

Les DRAMs offrent des capacités nominales importantes. Cependant, leur temps d'accès n'étant pas suffisamment rapide, un système qui permet d'entrelacer plusieurs bancs doit être mis en place. L'utilisation de SRAMs n'apporte guère de solutions au problème. Il faut donc trouver d'autres architectures plus intéressantes pour le calcul de l'élévation à la puissance.

3•3•1•2 Utilisation des opérateurs logarithme et exponentiel.

Chacun sait que l'utilisation des opérateurs logarithme et exponentiel permet de transformer l'élévation à la puissance en multiplication. Le prix de l'élévation à la puissance correspond au coût d'un opérateur logarithme, d'un opérateur exponentiel et enfin d'un multiplieur.

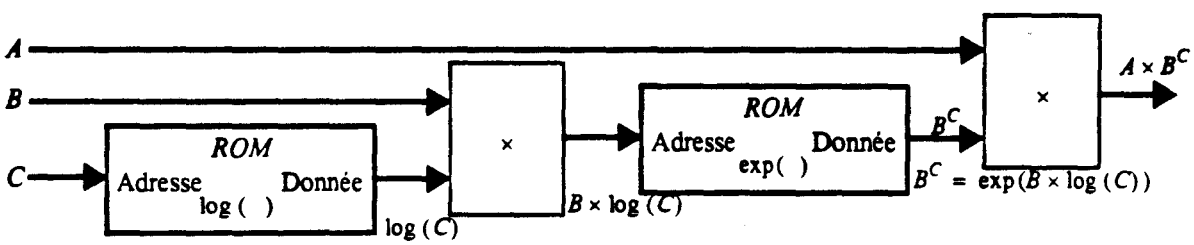


Figure 3•7 Calcul du spéculaire par les logarithmes.

3•3•1•3 Calcul itératif.

De la même façon qu'une multiplication peut se calculer par additions successives, l'élévation à la puissance peut être calculée par multiplications successives. Le nombre d'itérations, et donc le temps de calcul, est ainsi proportionnel à l'exposant utilisé.

- Pour calculer le résultat de la multiplication de A par B, on décompose B en bits. Puis pour chaque bit de B à 1, on ajoute la valeur de A décalée, de façon que le bit de poids faible de A soit à la position du bit de B considéré.

```
S ← 0
Pour i de 0 à n
   Si B[i] = 1 Alors
      S ← S + ( A << i )
   Fsi
Fpour
```

Figure 3•8 Calcul itératif de la multiplication. ($A \times B = S$)

• Pour calculer le résultat de l'élévation à la puissance C de B, on décompose C en bits. Puis pour chaque bit de C à 1, on multiplie le résultat intermédiaire par la valeur de P_i. On opère par élévations au carré successives, P_i étant le carré de P_{i-1} et B étant P_0.

```
M ← 1  P ← B
Pour i de 0 à n
   Si C[i] = 1 Alors
      M ← M * P
   Fsi
P ← P * P
Fpour
```

Figure 3•9 Calcul itératif de l'élévation à la puissance. ($B^C = M$)

3•3•2 Proposition d'une solution plus simple.

3•3•2•1 Limiter pour rendre faisable.

• Puisque la ROM utilisée dans PROOF est de taille trop importante, la première proposition va consister à réduire la taille de la ROM. Celle-ci est fixée par le nombre de bits (n_e) utilisés pour coder l'exposant spéculaire et par le nombre de bits (n_{Prs}) utilisés pour coder le résultat du produit scalaire (PrS). ($Taille = 2^{n_e + n_{Prs}}$)

La meilleure façon de réduire la complexité consiste à restreindre le codage de l'exposant et de "PrS". Si par exemple, on code "PrS" sur 10 bits en virgule fixe, la valeur de "PrS" étant comprise entre 0 et 1, on place la virgule après le $10^{ème}$ bit. (Cf chapitre 4.) On décode le nombre en divisant par 1024 le code binaire lu en valeur entière. Cela permet donc de coder 1024 valeurs (de 0/1024 à 1023/1024). On a, ainsi, une précision de 1/1024 mais la valeur 1 doit être codée "1.0000000000" soit (1024/1024).

En multipliant "PrS" par (1023/1024), la valeur 1 sera en fait codée "0.1111111111" et le zéro avant la virgule ne sera pas gardé dans le codage. On restreint ainsi les calculs à une précision de 1/1023. Ce changement d'échelle est supporté par l'unité de calcul du produit scalaire qui effectue la multiplication avant même de fournir le résultat.

Si, on limite à 16 le nombre d'exposants spéculaires, cela engendre naturellement des limitations quant aux performances du processeur d'éclairement qui serait synthétisé ainsi. Disposer seulement de 16 exposants spéculaires serait bien trop contraignant si cet exposant spéculaire devait obligatoirement avoir une valeur comprise entre 0 et 15 inclus.

• On introduit donc un numéro de matière. Ce numéro indique l'exposant spéculaire à utiliser mais aussi la constante spéculaire K_S. Une matière aura les mêmes propriétés lumineuses en tout point de l'espace. Ce modèle ne prend donc pas en compte les textures. Le nombre de matières est lui aussi limité à 16. La ROM réalise le calcul suivant : $(I \times K_s(Mat)) \cdot (PrS)^{exp(Mat)}$ intégrant ainsi la multiplication par le coefficient spéculaire.

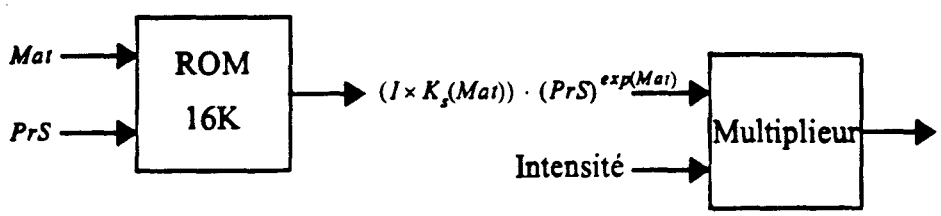


Figure 3•10 Diagramme de l'unité de base pour le calcul du spéculaire.

• En limitant à 10 bits, on a, bien entendu, réduit la taille de la ROM mais on a aussi diminué la précision des calculs. Dans le chapitre précédent, on a montré que la précision dans le calcul du spéculaire dépendait linéairement de l'exposant utilisé. Plus on utilisera un exposant spéculaire élevé plus l'erreur de calcul sera importante.

Sur les images synthétisées, des bandes de couleurs apparaissent et on perd le dégradé de couleur que l'on aurait dû obtenir si la précision n'était pas restreinte. On a conclu de façon empirique que l'utilisation des exposants spéculaires supérieurs à environ 45-50 donne des images qui présentent trop de bandes de couleurs induites par le manque de précision. On se limitera donc à l'utilisation d'exposants spéculaires inférieurs à 50, pour cette solution. Cela reste, bien sûr, subjectif et dépend de la qualité d'image que l'on désire obtenir.

3•3•2•2 Remplacement de la ROM par une L.U.T.

• La solution proposée utilise toujours un multiplieur qui permet de faire varier l'intensité de la source d'une image sur l'autre. En remplaçant la ROM par une RAM, on va pouvoir diminuer les contraintes pour l'utilisateur et tenter de supprimer le multiplieur. Avant chaque image, le contenu de la RAM peut être modifié.

Les données dans la RAM sont structurées sous forme d'un tableau à deux dimensions. On accède à l'information par la matière en première dimension et par le résultat du produit scalaire $\hat{N} \cdot \hat{H}$ en seconde dimension.

Cela va permettre, outre la variation de l'intensité, de disposer d'une plus vaste panoplie de matières. Il ne pourra toujours pas y avoir plus de 16 matières simultanément dans la scène mais les matières utilisées pourront être choisies par l'utilisateur, ce qui ne pouvait être le cas jusqu'à présent. (Les coefficients sont les produits de l'intensité I et de la valeur de K_S.)

Dans la figure ci-dessous, est rappelée, pour chaque case du tableau, l'expression arithmétique qui doit être évaluée.

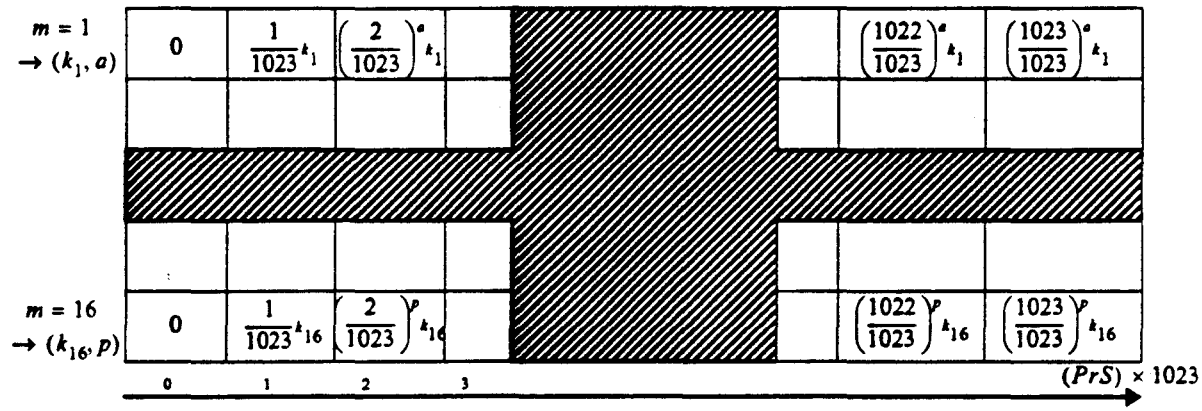


Figure 3•11 Le contenu de la RAM pour le calcul du spéculaire

• Il faut mettre en place un mécanisme permettant de modifier le contenu de la RAM. C'est à dire recalculer 1024 valeurs pour chacune des 16 matières. L'architecture générale de l'unité de calcul du spéculaire devient :

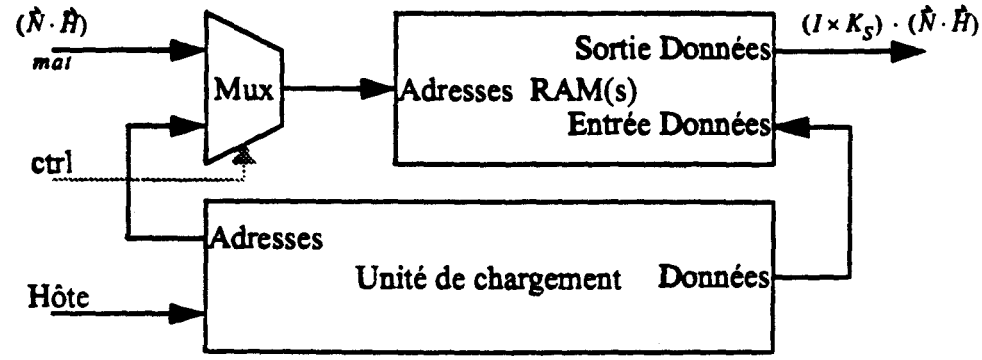


Figure 3•12 Architecture de l'unité de calcul spéculaire.

3•3•2•3 Algorithme de remplissage.

• Pour charger la RAM, les opérations requises sont les mêmes que pour calculer directement le spéculaire avec la formule de Phong [44]. Pour le moment le gain réside seulement dans le nombre de calculs à effectuer. Dans le cas d'une image 512 sur 512, qui a été pris comme exemple précédemment, il y a 16384 évaluations pour le chargement du tableau au lieu de 262144 pour le calcul direct. La puissance requise est divisée par 16.

Si on voulait transformer le modèle, pour avoir 256 matières simultanées au lieu de 16 ou accroître la précision en utilisant 14 bits du produit scalaire au lieu de 10, le gain serait nul. Certes un gain de 16, est déjà appréciable pour la réalisation, mais il n'est guère possible d'envisager une version future aux performances accrues. En fait, il serait intéressant de pouvoir encore diminuer le coût de cette solution.

• Quant à la ROM utilisée pour mettre en mémoire les fonctions élévation à la puissance, on se borne à y mettre sous forme de 0 et de 1 la "manière de tracer la fonction", c'est-à-dire la succession des mouvements horizontaux (0) et verticaux (1).

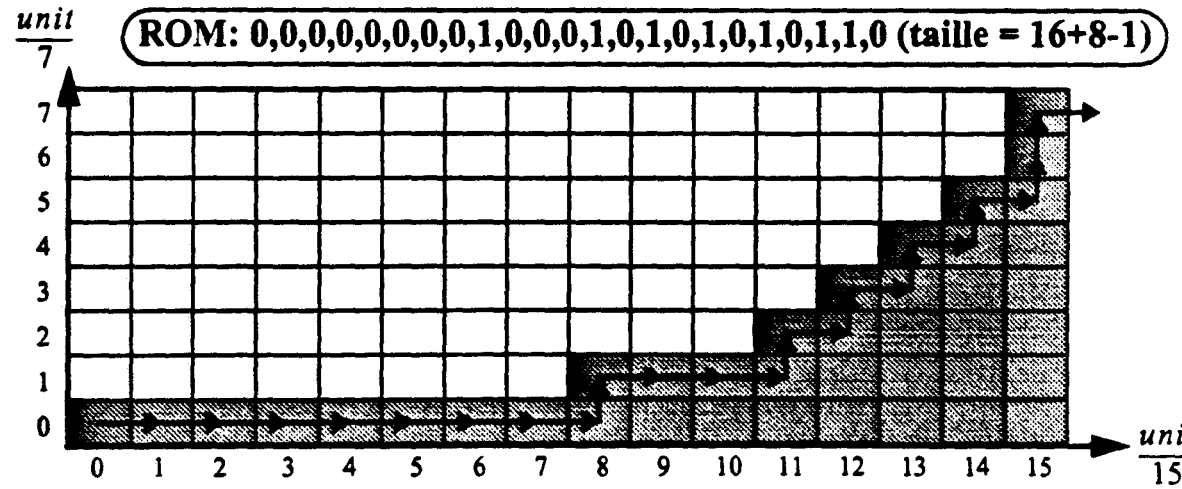


Figure 3•13 Exemple de codage de fonction en ROM.($f(x) = x^4$)

• Un dernier mouvement horizontal est ajouté à la fin pour terminer le remplissage de la table pour la matière courante et passer au traitement de la suivante. Maintenant, on sait calculer de manière incrémentale la fonction $K_S\cdot(x)^n$.

• Au départ, un cycle d'initialisation met, premièrement, le registre "Adresse RAM" à la position de début de la matière dans la RAM, puis, le registre "Valeur" qui sert à stocker dans la RAM et le registre "Erreur" à zéro.

La multiplication par K_S est effectuée par un additionneur qui ajoute K_S dans le registre "Erreur" à chaque déplacement vertical. Sur l'exemple (cf Figure 3•14) les cycles 9, 13, 15, 17, 19, 21 et 22 illustrent un déplacement vertical. Quand le registre "Erreur" déborde, c'est-à-dire dépasse la valeur P_y, le contenu du registre "Valeur" est augmenté de 1. Les cycles 13, 15, 17, 21 et 22 de l'exemple exécutent ce cas de figure.

A chaque déplacement horizontal, le contenu de "Valeur" est mis en mémoire dans la RAM à l'adresse spécifiée par le registre "Adresse RAM". Puis on passe à l'adresse suivante en ajoutant 1 dans ce registre.

Dans l'exemple, les valeurs sont mises en mémoire pendant les cycles 1 à 8, 10 à 12, 14, 16, 18, 20 et 22.

Cycle	In	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Adresse RAM av.	?	0	1	2	3	4	5	6	7	8	8	9	10	11	11	12	12	13	13	43	14	15	15	15
Erreur av. × 1/8	?	0	0	0	0	0	0	0	0	6	6	6	6	4	4	2	2	0	0	6	6	4	2	2
Valeur avant	?	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2	2	3	3	3	3	4	5	5
Code (ROM)		0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	1	0	1	1	0
Stockage RAM:																								
Adresse RAM ap.	0	1	2	3	4	5	6	7	8	8	9	10	11	11	12	12	13	13	14	14	15	15	15	0
Erreur ap. × 1/8	0	0	0	0	0	0	0	0	6	6	6	6	4	4	2	2	0	0	6	6	4	2	2	0
Valeur après	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2	2	3	3	3	3	4	5	5	6
Mouvement		D	D	D	D	D	D	D	D		D	D	D	H	D	H	D	H	D		D	H	H	D

Légende pour les mouvements: "D" vers la Droite ou "H" vers le Haut.

Figure 3•14 Exemple de chargement de la RAM : pour $f(x) = \frac{6}{8} \cdot x^4$ à partir du tracé de x^4.

Pour chaque cycle, on trouve dans l'exemple, le contenu des registres avant exécution, suivi du code de l'instruction puis une case indiquant le stockage dans la RAM quand elle est hachurée, enfin le contenu des registres après exécution suivi du mouvement (Droite, Haut).

L'algorithme, ainsi exécuté, charge dans la RAM la fonction $f(x) = (6/8) \cdot x^4$.

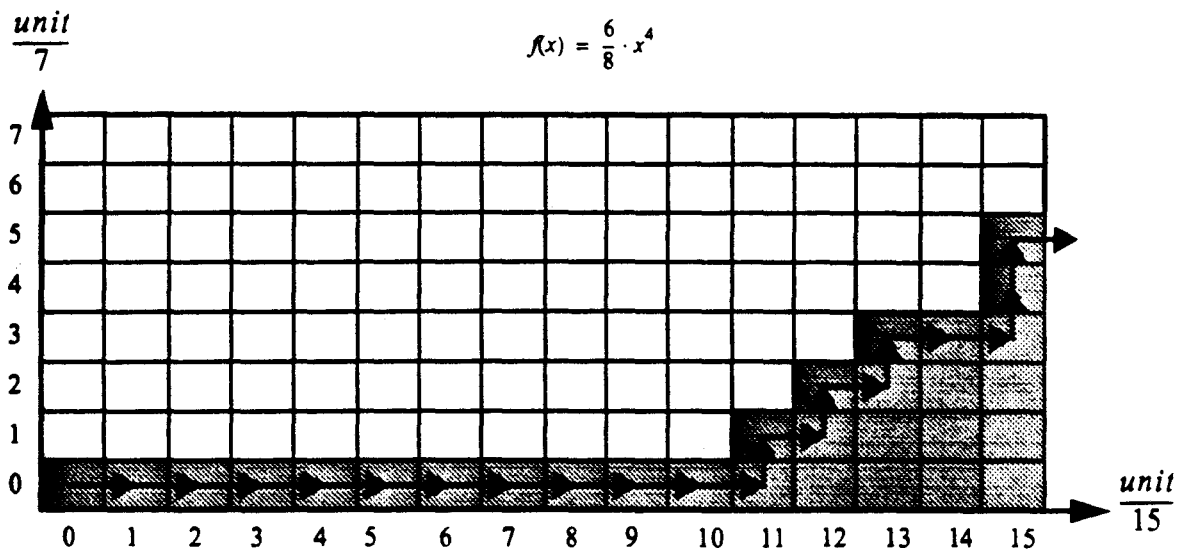


Figure 3•15 Exemple de résultat pour le tracé incrémental dans la RAM.

• Pour résumer, on utilise une ROM (cf Figure 3•16) dans laquelle se trouve la "manière" de tracer chaque fonction x^n. La "manière" est codée sous forme de 0 et de 1 correspondant aux déplacements horizontaux et verticaux intervenant dans le tracé de la courbe. Pour la fonction $K_s \times x^n$, on effectue une mise à l'échelle de la fonction x^n. L'opération est exécutée de façon incrémentale en jouant sur la hauteur d'un déplacement vertical.

Comme le calcul de $K_s \times x^n$ est effectué en deux étapes pour lesquelles la précision est limitée, l'erreur de calcul est accrue. Il est important de signaler qu'un algorithme non incrémental effectuant les calculs en deux étapes avec la même précision engendrerait la même erreur. On estime l'erreur maximale commise à $2 \times P_y$.

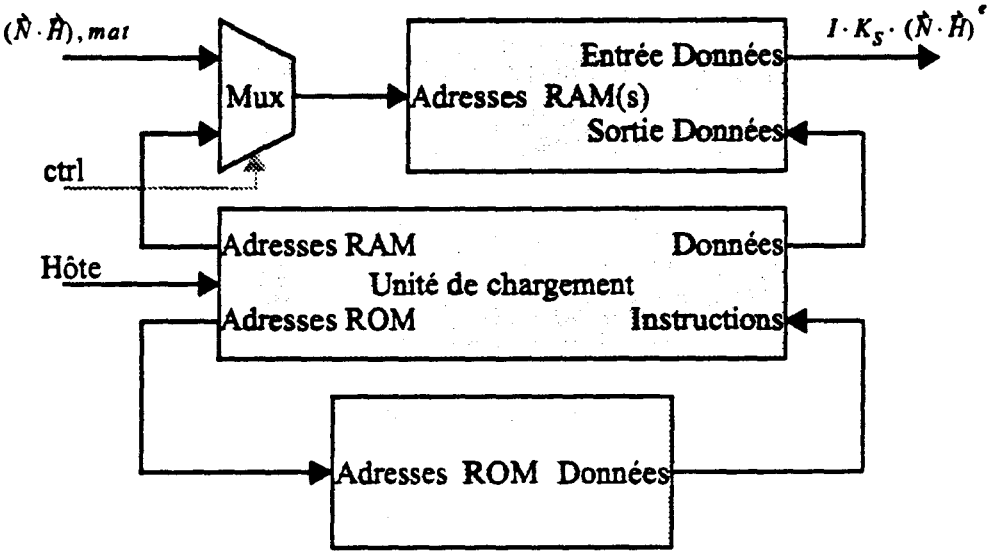


Figure 3•16 Architecture de l'unité de calcul du spéculaire.

• L'unité de chargement pourra être synthétisée de diverses façons suivant les choix technologiques. Le chapitre suivant évoquera en détail les diverses solutions d'implémentation fine, en fonction notamment du choix entre le chargement en retour trame et le chargement utilisant un "double-buffer". Mais, il faudrait d'abord, améliorer quelque peu la solution proposée. On retrouve sur la figure, les registres "Erreur", "Valeur" et "Adresse RAM" ainsi que la RAM qui sert pour mettre en mémoire les valeurs n et IK_S de chacune des 16 matières.

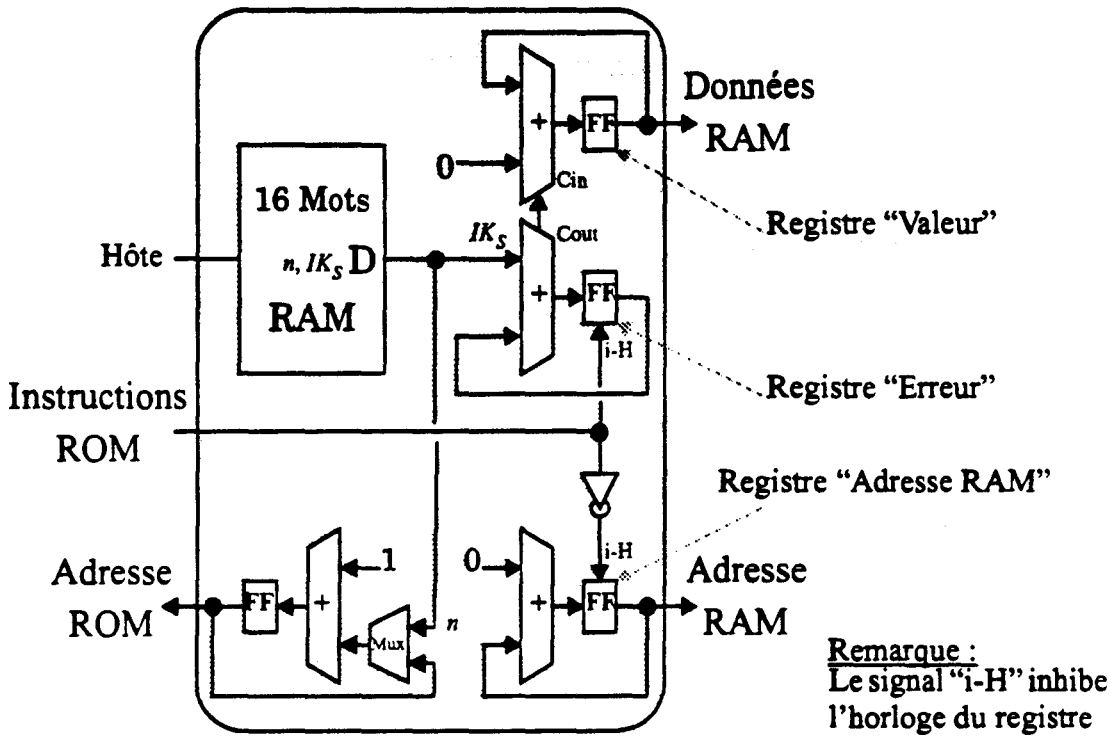


Figure 3•17 Architecture détaillée de l'unité de chargement pour le spéculaire

3•3•3 Reculer les limites.

3•3•3•1 Les limites.

• La limite principale de la solution présentée est de ne pas permettre facilement l'accroissement de ses performances. On s'est fixé un nombre réduit de matières (16 matières) et une précision limitée (10 bits du résultat du produit scalaire) afin d'utiliser une RAM de taille raisonnable. On peut aisément calculer la taille de la RAM et le temps de chargement en fonction des performances demandées.

Taille de la RAM

Matières / précision	16	32	64
10 bits	16 K	32 K	64 K
11 bits	32 K	64 K	128 K
12 bits	64 K	128 K	256 K

Nombre d'opérations pour une image (512/512) calculée au vol: 262144

Temps de chargement en cycles

Matières / précision	16	32	64
10 bits	20480	40960	81920
11 bits	36864	73728	147456
12 bits	69632	139264	278528
13 bits	135168	270336	
14 bits	266240		

Figure 3•18 Quelques exemples d'évolution du temps de chargement et de la complexité en fonction des performances exigées.

Comme le montrent les tableaux ci-dessus, non seulement la taille de la RAM doit croître avec les performances, mais aussi, le temps de chargement, ce qui est encore plus embarrassant. Avec 16 matières et 14 bits de précision sur le produit scalaire, il faut autant d'opérations pour charger la RAM que pour effectuer le calcul au vol. Certes, le chargement de la RAM requiert des opérations plus simples. L'architecture reste donc peu complexe, mais cela veut tout de même dire que le temps de chargement de la RAM sera de l'ordre du temps d'affichage d'une image écran.

• Cette lourdeur, face aux possibilités d'accroissement des performances, indique qu'il faudrait trouver une amélioration de l'architecture qui permette d'augmenter les performances sans accroître linéairement la complexité.

3•3•3•2 Le prix maximum admissible.

• Il faudrait que le coût de la solution soit modique. Augmenter le nombre de matières n'est pas la préoccupation principale pour accroître les performances. Augmenter le nombre de bits est plus important, car cela permet d'améliorer la qualité des images et d'utiliser des exposants spéculaires plus élevés.

• La nécessité de garder beaucoup de bits du produit scalaire pour le calcul de l'élévation à la puissance est induite par l'importante pente de la courbe au voisinage de 1. Autrement dit, la précision n'est requise que pour traiter les valeurs proches de 1. Par contre au voisinage de 0, la pente de l'élévation à la puissance est faible et le calcul ne requiert donc que peu de précision.
On va donc étudier la possibilité de faire varier la précision à la demande.

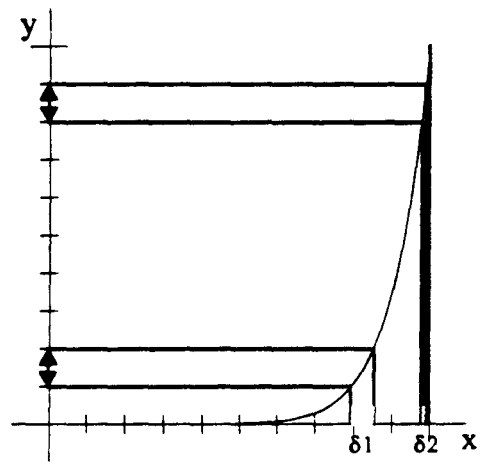


Figure 3•19 Variation de la précision.

3•3•3•3 Les objectifs de l'Unité "d'Arrangement des Bits" ou U.A.B.

• Sur la Figure 3•19, on observe facilement qu'une même variation en y, correspond à une variation qui dépend de l'abscisse x. $\delta 1$ est plus grand que $\delta 2$ car $\delta 1$ correspond à une abscisse x plus éloignée de la valeur 1. Il faut donc accroître la précision plus la valeur $(1-x)$ devient petite.

En s'inspirant du codage des nombres flottants, on peut trouver un système avantageux pour adapter la précision au besoin. Un nombre flottant x est codé sous forme d'un exposant e et d'une mantisse m de façon que $x = m \times 2^e$. Le codage des nombres flottants permet d'accroître la précision des petites valeurs c'est à dire au voisinage de 0. Un codage flottant de la valeur $(1-x)$ devrait donc résoudre le problème.

3•3•3•4 Définition formelle d'un opérateur pour l'U.A.B.

• Pour accroître la précision de x au voisinage de 1, il suffit d'augmenter la précision de $1-x$ au voisinage de 0. Afin d'utiliser une solution semblable à la transformation d'un nombre du codage virgule fixe vers le codage virgule flottante, on se définit l'opérateur *Uab* (le choix du nom sera expliqué ultérieurement) tel que :

$$Uab : \begin{cases} [0,1] \rightarrow [0,\frac{1}{2}] \times \mathbb{N} \\ x \rightarrow (m,e) \qquad \text{telque} \qquad (1-x) = (1-m) \times 2^{1-e} \end{cases}.$$

Comme x et m sont des nombres codés en complément à deux, on a les propriétés suivantes : $(1-x) = \bar{x}$, $(1-m) = \bar{m}$ et $(1-e) = \bar{e}$. Car on a $0 \le x < 1$ et $0 \le m < 1$. On peut ainsi simplifier la condition reliant x, m et e, pour obtenir finalement : $\bar{x} = \bar{m} \times 2^{\bar{e}}$.

• Pour synthétiser l'opérateur *Uab*, il faut avant tout établir la manière de procéder pour déterminer e à partir de x. Il s'agit, comme le montre la dernière relation qui vient d'être établie, de trouver l'exposant $\bar{e}$ du codage en virgule flottante.

Pour déterminer l'exposant utilisé dans le codage virgule flottante, on cherche la position du premier 1 en parcourant le codage binaire. Ce parcours est effectué en partant du bit de poids fort. Chercher la position du premier 1 dans $\bar{x}$, revient à chercher la position du premier 0 dans x.

3•3•3•5 Le mécanisme de l'U.A.B.

• La figure suivante illustre le principe de la transformation (cf Figure 3•20). Suivant la valeur trouvée pour e, tous les bits, depuis celui de poids fort jusqu'au premier 0 inclus, sont connus. Pour coder la mantisse m, seuls les bits qui ne peuvent être déterminés par e doivent être conservés. Grâce à ce système, on peut se fixer une dynamique donnée pour m et la précision sera adaptée au besoin. Plus x sera proche de 1, plus grande sera la précision qui sera gardée en mémoire dans le codage *Uab*(x).

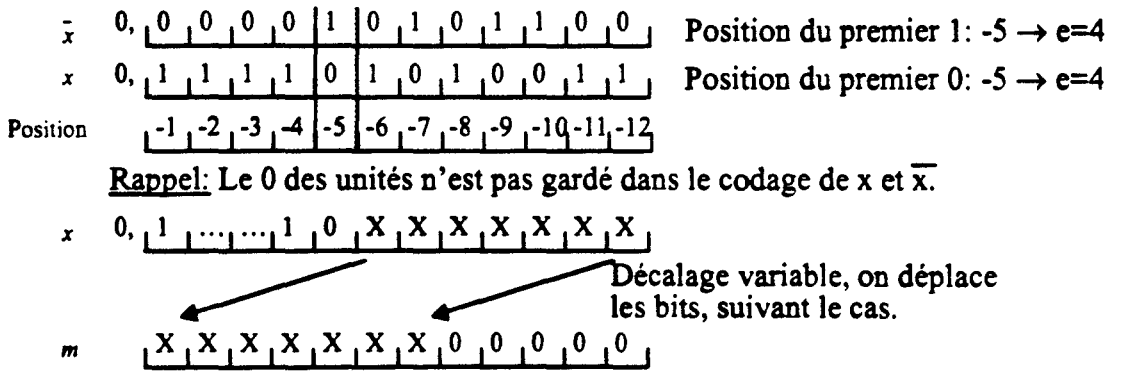


Figure 3•20 Exemple expliquant le fonctionnement de U.A.B.

• Il ne reste plus qu'à optimiser le fonctionnement du codage. Tant que le nombre des bits extraits de x est égal au nombre de bits de m, tout peut rester comme cela a été défini. Mais dès que le nombre de bits extraits devient inférieur, il n'est plus utile de continuer à décaler, car cela ne permet plus d'augmenter la précision.

En fonction du nombre de bits de x et de m, on va définir une valeur limite pour l'exposant e.

• Avec N_e bits pour coder l'exposant, e varie de 0 à $(2^{N_e}-2)$ pour coder le nombre qui s'écrit avec de 0 à $(2^{N_e}-2)$ bits à "1" suivis de la mantisse précédé d'un "0". Il peut aussi prendre la valeur $(2^{N_e}-1)$ pour coder le nombre qui s'écrit avec $(2^{N_e}-2)$ bits à "1" suivis de la mantisse. Soit N_m le nombre de bits pour coder la mantisse. Soit N_x le nombre maximum de bits utilisés par l'U.A.B. La précision est obtenue par la formule suivante : $N_x = 2^{N_e} + N_m - 1$.

exposant / mantisse	0 bit↓	1 bit↓	2 bits↓	3 bits↓	4 bits↓
10 bits→	10 bits	11 bits	13 bits		
9 bits→	9 bits	10 bits	12 bits	16 bits	
8 bits→	8 bits	9 bits	11 bits	15 bits	23 bits
7 bits→		8 bits	10 bits	14 bits	22 bits
6 bits→			9 bits	13 bits	21 bits

Exemple de codage 0.1 1 1 1 1 1 0 X X X X X X X e=6 codé sur 3 bits

pour le cas particulier: 0.1 1 1 1 1 1 1 X X X X X X X e=7 codé sur 3 bits

Le tableau précédent donne quelques exemples. Dans un premier temps, notre choix se portera sur 7 bits de mantisse et 3 bits d'exposant.

3•3•3•6 Les conséquences sur l'algorithme de chargement.

• Cette méthode permet de résoudre le problème de la croissance linéaire de la RAM. Cela permet, grâce à "l'unité d'arrangement des bits", une plus grande précision sans devoir accroître la taille de la RAM. Cette unité trouve sa place sur le parcours des données lors du calcul du spéculaire par la RAM. Le contenu de la RAM couplée à l'U.A.B est évidemment différent de celui de la RAM utilisée seule.

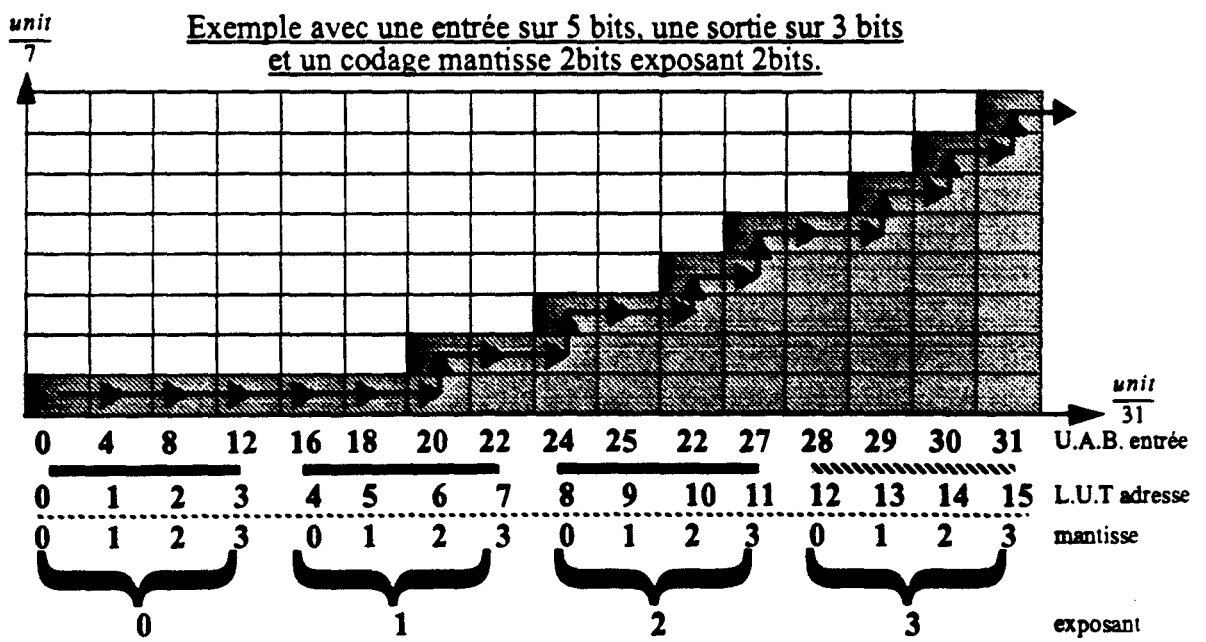


Figure 3•21 Exemple de codage de fonction en ROM avec l'U.A.B.

Mais, comme on peut le remarquer sur l'exemple, on doit toujours tracer une fonction strictement croissante. La fonction à tracer est la suivante : $f(a) = K_S \times (U_{ab}^{-1}(a))^n = y$. Pour charger de façon incrémentale cette fonction dans la RAM, on va toujours utiliser une ROM dans laquelle on stockera cette fois-ci la "manière" de tracer la fonction $(U_{ab}^{-1}(a))^n$ sous forme de mouvements horizontaux (codés 0) et de mouvements verticaux (codés 1).

Cela ne modifie donc en rien l'architecture de l'unité de chargement. Seul le contenu de la ROM est modifié.

• Dans l'exemple précédent, on remarque que l'on aurait pu utiliser des mouvements diagonaux au lieu des mouvements verticaux. On peut donc se demander si l'utilisation de l'U.A.B. ne présenterait pas l'avantage d'autoriser les mouvements diagonaux que la méthode précédente interdisait.

Rappelons que l'utilisation des mouvements verticaux a pour conséquence d'allonger le temps de chargement de la table de 255 cycles (le nombre d'états de sortie de l'unité de calcul, moins 1).

Soit Δ_x la largeur minimale d'un déplacement horizontal, soit Δ_y la hauteur d'un déplacement vertical, on a : $\Delta_x = 2^{[1 - N_m - 2^{N_e}]}$ $\Delta_y = 2^{-N_y}$ $\frac{\Delta_y}{\Delta_x} = 2^{[N_m + 2^{N_e} - N_y - 1]}$. En posant : $Max(n)$ la valeur maximale de n pour laquelle la pente de la courbe x^n reste inférieure à la pente permettant de passer d'un point au suivant par un mouvement diagonal. On obtient : $Max(n) = 2^{[N_m + 2^{N_e} - N_y - 1]}$.

• Pour $N_y = 8$, $N_m = 7$ et $N_e = 3$, la valeur maximale de n est 64. Cette valeur est déjà assez importante pour la plupart des images, mais par souci de ne pas limiter les possibilités, on continue à utiliser les mouvements horizontaux et verticaux et on s'autorise ainsi des exposants spéculaires supérieurs à 64.

3•3•3•7 Inconvénient.

Comme on vient de le voir, on peut à moindre coût accroître la précision et donc utiliser des coefficients spéculaires plus élevés. La conséquence directe en est l'élargissement de la gamme des coefficients spéculaires utilisables. La taille de la ROM, qui contient les descriptions algorithmiques pour chaque coefficient, devient donc importante.

3•3•4 Des méthodes d'approximation.

Dans le but de supprimer cette ROM, on propose ici des méthodes d'approximation ainsi que l'implémentation des algorithmes qui leur correspondent.

(Remarque : Dans toute cette partie, on oublie pour un temps "l'Unité d'Arrangement des Bits". La partie suivante expliquera comment allier la méthode d'approximation que l'on aura choisie et l' "U.A.B".)

Parmi les différentes méthodes d'approximation étudiées, on pourra retrouver en annexe l'approximation linéaire (Voir paragraphe B•1•1.), l'approximation quadratique (Voir paragraphe B•1•2.), l'approximation homographique (Voir paragraphe B•2•1.), l'approximation rationnelle de degré deux (Voir paragraphe B•2•2.) et enfin l'approximation rationnelle de degré deux par morceaux (Voir paragraphe B•3•2.).

• L'approximation linéaire ne présente pas d'intérêt. Par contre l'approximation quadratique, bien qu'elle ne fournisse pas un résultat visuel parfait, fournit une solution présentant un rapport coût qualité intéressant. Le paragraphe 3•3•4•1 reprend en détail les possibilités pour l'intégration matérielle de cette méthode.

L'approximation par une fonction homographique ne fournit pas le résultat que l'on pouvait escompter. L'étude de l'élaboration matérielle de cette méthode peut cependant être consultée en annexe (Voir paragraphe C•2 pour l'étude théorique et paragraphe C•4 pour l'application au cas de l'approximation désirée.).

L'approximation par une fonction rationnelle de degré deux, améliore, certes, le résultat mais n'est pas encore satisfaisante. Par conséquent cette méthode a été écartée au profit de la dernière méthode présentée. On peut, malgré tout, consulter en annexe l'étude de l'élaboration matérielle pour cette méthode (paragraphe C•3 pour la théorie et paragraphe C•5 pour l'application au cas de l'approximation désirée).

Enfin, l'approximation rationnelle de degré deux par morceaux donne les meilleurs résultats mais s'avérera plus coûteuse comme on va pouvoir le constater au paragraphe 3•3•4•2.

3•3•4•1 L'approximation quadratique.

• Il s'agit d'approximer la fonction x^n par un morceau de parabole. On peut déterminer les coefficients suivant un critère sur la pente ou sur la surface. L'étude théorique donnant leurs valeurs peut être consultée au paragraphe B•1•2.

• L'approximation étant établie, il reste à trouver un algorithme approprié, ainsi que l'architecture qui en découlera, pour charger la RAM pendant le retour trame. La première solution consiste tout naturellement à calculer de façon incrémentale la courbe au fur et à mesure du remplissage.

Le chargement du palier horizontal ne pose guère de problème : on parcourt la table de 0 à d et on efface la mémoire en y plaçant la valeur 0. Puis on calcule les valeurs suivantes obtenues par l'évaluation de la fonction $a \times x^2 + b \times x + c$. Les valeurs sont calculées par un système DDA.

$$\Delta_x(x) = Q(x+p_x) - Q(x) = 2a \times x \times p_x + a \times p_x^2 + b \times p_x \qquad \Delta_{xx} = \Delta_x(x+p_x) - \Delta_x(x) = 2a \times p_x^2$$

$$Q(d) = 0,\ \Delta_x(d) = \left(\frac{1}{1-d}\right)^2 \times p_x^2 \text{ et } \Delta_{xx} = 2\frac{d}{1-d} \times p_x^2.$$

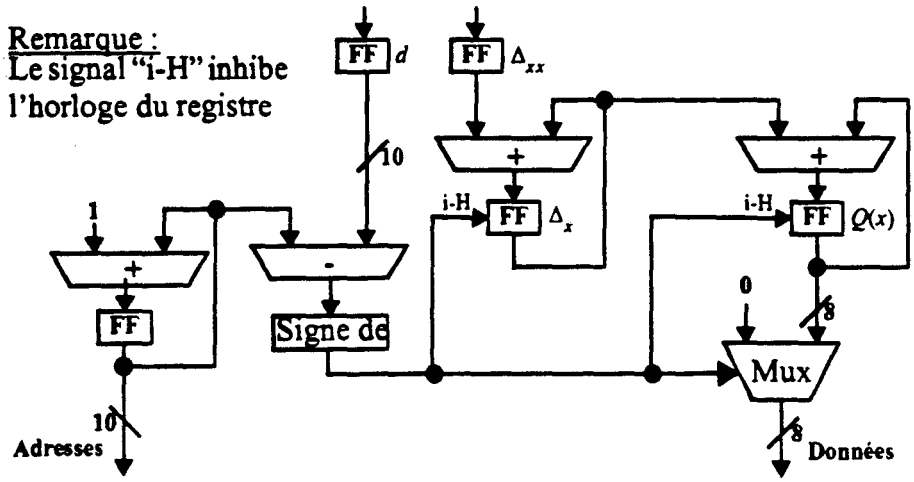


Figure 3•22 Schéma du DDA de chargement de l'approximation quadratique.

• Une autre architecture peut être conçue, s'inspirant de celle proposée pour le chargement avec la vraie fonction d'élévation à la puissance. Plutôt que de recalculer la fonction $\hat{Q}(x)$ avec un DDA, on peut l'obtenir par mise à l'échelle de la fonction x^2. On applique à la fonction une translation et une mise à l'échelle en X.

Plaçons dans la ROM des instructions (cf Figure 3•16) la "manière" de tracer la fonction x^2, pour que l'unité de chargement l'utilise en lui appliquant une mise à l'échelle en X. Le codage a été conçu pour permettre facilement la mise à l'échelle en X. Les mouvements utilisés sont seulement horizontaux et verticaux. Un étirement des abscisses est similaire à un étirement des ordonnées. En intervertissant les deux axes et les deux déplacements, on transpose un cas dans l'autre (cf Figure 3•23).

De manière à économiser le nombre de cycles utilisés pour le chargement, la mise à zéro de l'intervalle $[0..d]$ sera effectuée en parallèle, en profitant des cycles durant lesquels le processus de mise à l'échelle n'accède pas à la RAM parce qu'il n'effectue pas le déplacement horizontal. En procédant de la sorte, le temps de chargement d'une matière reste inchangé par rapport à celui de la méthode utilisant la ROM des fonctions élévation à la puissance sans approximation.

```
Début
    Erreur_A ← 0
    Erreur_Y ← 0
    Adresse_chargement ← d · Nx
    Adresse_mise_à_zéro ← 0
    Pour chaque Instruction Faire
        Suivant Instruction
        : → Faire
            Erreur_A ← Erreur_A + (1 - d) Nx
            Si ( Erreur_A > Nx )
            Alors
                Erreur_A ← Erreur_A - Nx
                [Adresse_chargement] ← Valeur
                Adresse_chargement ← Adresse_chargement + 1
            Sinon
                [Adresse_mise_à_zéro] ← 0
                Adresse_mise_à_zéro ← Adresse_mise_à_zéro + 1
            FinSi
        Fait ( → )
        : ↑ Faire
            Erreur_Y ← Erreur_Y + (I · Ks) · Ny
            Si ( Erreur_Y > Ny )
            Alors
            Erreur_A ← Erreur_A - Nx
            Valeur ← Valeur + 1
            FinSi
        Fait ( ↑ )
    FinPour
Fin
```

Figure 3•23 Algorithme de chargement incrémental pour l'approximation quadratique.

• A partir de l'algorithme ainsi établi, il est facile de constituer l'architecture de l'unité de chargement fonctionnant sur ce principe. On remarque que cette méthode donne une architecture globalement plus importante que celle obtenue par la méthode DDA.

Quand il va s'agir d'intégrer le mécanisme de l'U.A.B, il n'est pas certain non plus que ce soit l'une ou l'autre des deux méthodes qui offrira le plus d'avantages.

• Cependant la seconde offre l'avantage appréciable de permettre l'utilisation de toute méthode d'approximation consistant à effectuer la mise à l'échelle d'une fonction donnée, même très compliquée à câbler par un DDA.

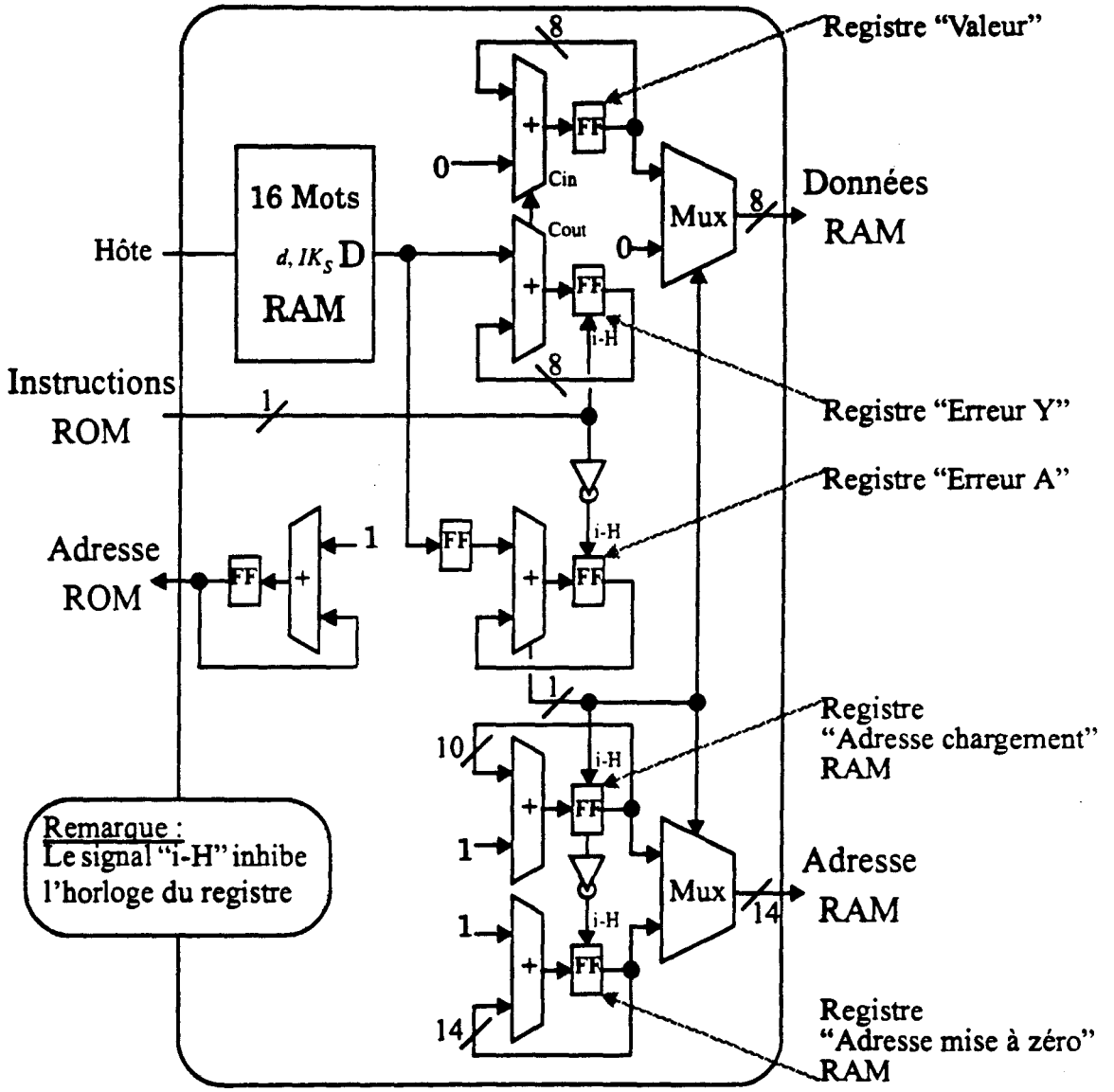


Figure 3•24 Architecture de l'unité de calcul de l'approximation quadratique.

On pourrait donc envisager de remplacer le morceau de parabole par une courbe de plus haut degré offrant une meilleure approximation. Cela permettrait, bien évidemment, d'obtenir une meilleure approximation pour les exposants spéculaires de haut degré, mais cela empêcherait l'utilisation des exposants spéculaires de valeur inférieure au degré de la courbe utilisée. En vue d'améliorer les performances, il semblerait donc intéressant d'étudier d'autres types d'approximation.

3•3•4•2 L'approximation rationnelle de degré deux par morceaux.

• Il s'agit d'établir une approximation de la fonction x^n avec un palier nul suivi d'un morceau de fonction rationnelle du second degré. On détermine les coefficients en utilisant à la fois un critère sur la pente et un critère sur la surface. L'étude théorique donnant leurs valeurs peut être consultée au paragraphe B•3•2.

• L'architecture proposée est un mélange entre l'automate de chargement défini en annexe au paragraphe C•5 pour calculer la fonction homographique, et l'un des deux automates présentés aux Figure 3•23 et Figure 3•24 pour effectuer la compression horizontale d'une fonction du second degré. La solution retenue consiste à utiliser l'architecture de la Figure 3•24. Le tracé de la fonction du second degré est mémorisé dans la ROM.

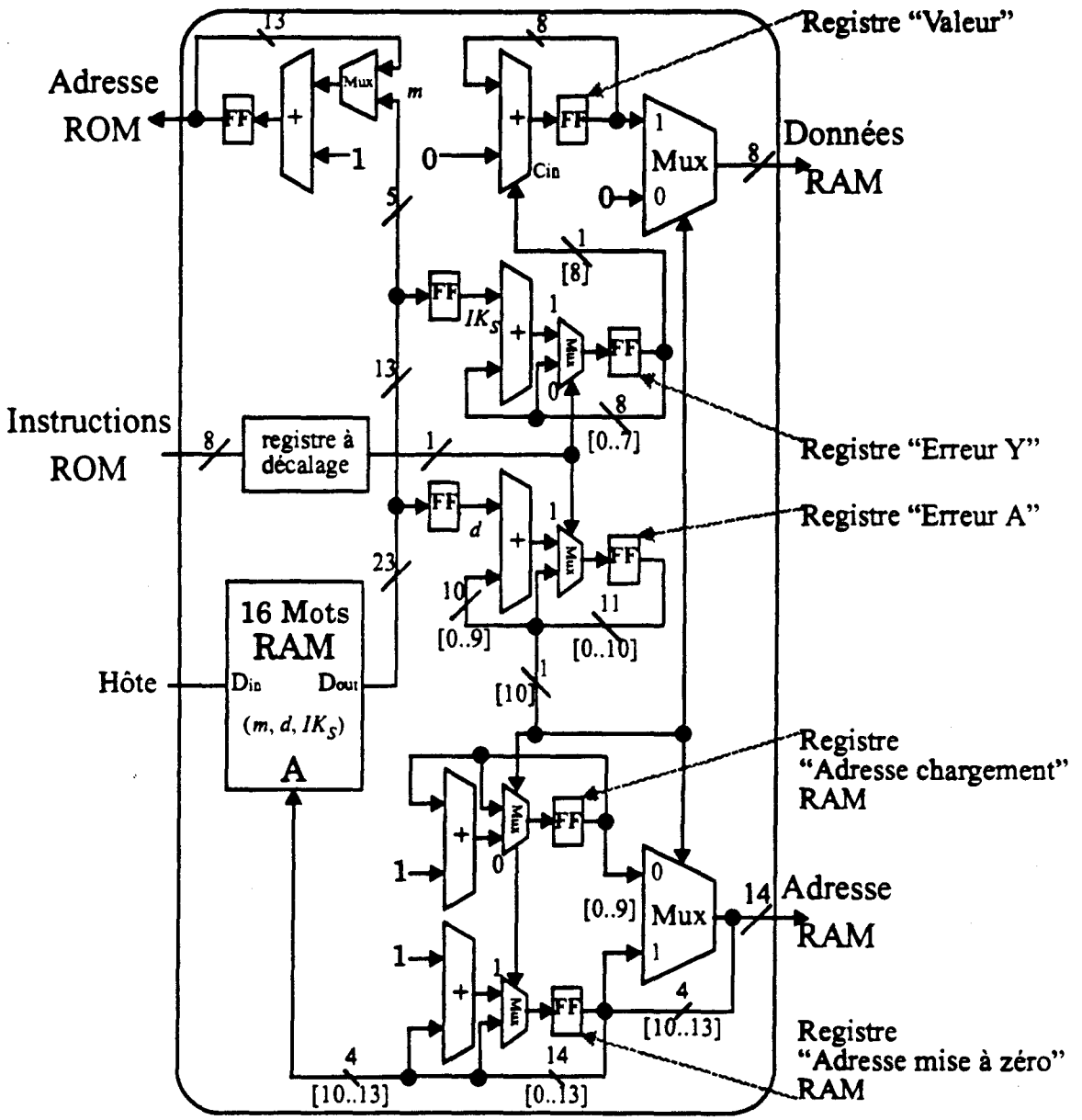


Figure 3•25 Architecture de l'automate de chargement avec mise à l'échelle horizontale.

On peut facilement modifier l'architecture pour qu'il y ait plusieurs fonctions dans la ROM. Chacune de ces fonctions correspond à une valeur de m. Comme on vient de le remarquer au paragraphe précédent, les valeurs de m sont regroupées dans un intervalle réduit. La fonction ne change presque pas quand m varie faiblement. Il n'y a donc pas beaucoup de fonctions à conserver dans cette ROM. Bien que l'on utilise sensiblement la même architecture que celle présentée Figure 3•17, on résout le problème que l'on s'était posé au paragraphe 3•3•3•7, à savoir la taille trop importante de la ROM.

3•3•5 Intégration des deux méthodes.

3•3•5•1 En précalculant dans des tables de transcodage.

• Dans toute la partie précédente traitant des méthodes d'approximation, nous avons abandonné l'Unité d'Arrangement des Bits. L'architecture finale de l'unité de chargement issues de notre réflexion sur les méthodes d'approximation diffère principalement de celle présentée au paragraphe 3•3•3•7 par le système de mise à l'échelle horizontale. Or l'U.A.B. procède, elle aussi à sa manière, à une compression horizontale. La difficulté va donc consister à faire cohabiter les deux systèmes.

La ROM permet d'effectuer le chargement incrémental de la table de transcodage (qui est réalisée sous forme d'une RAM) quand il n'y a pas de compression horizontale de la courbe. La méthode évidente de chargement de la RAM à partir de la ROM consisterait donc à parcourir celle-ci. On calcule à chaque étape l'adresse correspondante dans la RAM vu la compression utilisée et l'U.A.B. et on met en mémoire dans la RAM le résultat de la fonction. Le calcul suit le diagramme présenté ci-dessous :

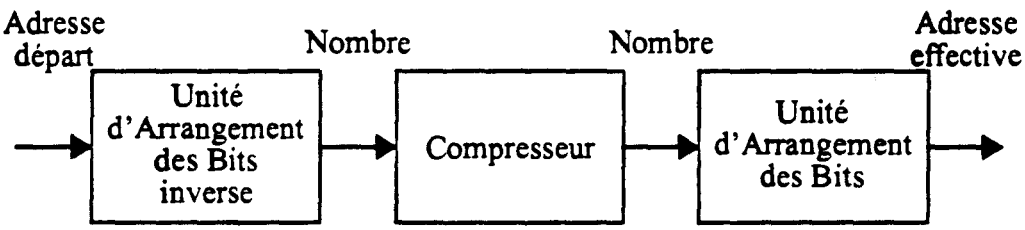


Figure 3•26 Transcodage adresse ROM vers adresse RAM.

• Grâce à ce transcodage pour chaque adresse de départ (ou adresse virtuelle qui peut être considérée comme étant l'adresse de stockage dans la RAM s'il n'y avait pas compression), on obtient une adresse effective. Pour pouvoir correctement charger la RAM, il faut que les adresses effectives parcourent l'intégralité des adresses de la RAM. Comme dans le cas précédent, un compteur peut se charger de la mise à zéro de la RAM pour les adresses comprises entre 0 et d. Il faudrait donc que toutes les adresses comprises entre d et 1 correspondent à une adresse effective issues du transcodage. Malheureusement cela n'est pas toujours vérifié comme le montre la figure suivante :

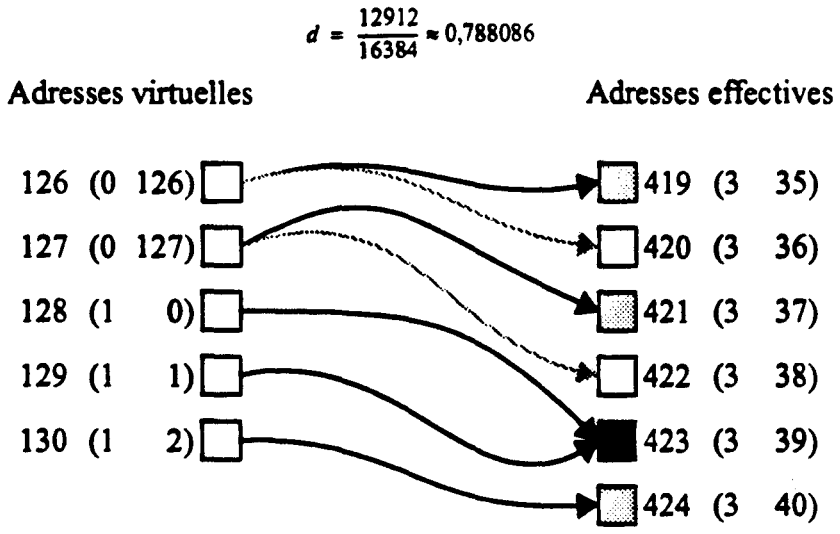


Figure 3•27 Exemple de l'effet du transcodage.

On voit sur l'exemple de la Figure 3•27 qu'il y a des adresses effectives qui ne sont pas l'image d'une adresse virtuelle par l'application de transcodage, et qu'il y en a d'autres qui sont l'image de plusieurs antécédents. Cela a deux significations :

- Premièrement, si on parcourt toutes les adresses de départ, on oubliera certaines adresses effectives et l'on ne remplira pas complètement la RAM. Il faudrait prévoir un mécanisme qui comble les trous. Pour ce faire, on peut soit les marquer pour s'occuper d'eux quand on accède une nouvelle fois à une même adresse effective, soit s'autoriser la possibilité d'effectuer deux écritures en même temps et ainsi combler les trous au fur et à mesure. (On peut facilement montrer qu'il n'y a jamais deux trous côte à côte.)

- Deuxièmement, si on parcourt toutes les adresses effectives de la RAM, cela permettra de combler les trous mais il sera parfois nécessaire d'avancer de deux positions dans les adresses virtuelles. Or celles-ci sont obtenues obligatoirement par le parcours séquentiel de la ROM. Pour pouvoir procéder de la sorte, il faudrait contrôler le balayage séquentiel des adresses effectives et bloquer ce balayage, le temps d'avancer de plusieurs adresses virtuelles. Dans ce cas, le temps de chargement dans la RAM d'une fonction spéculaire n'est plus constant (1280 cycles) mais dépendra de la valeur de d. On obtient l'algorithme incrémental de chargement présenté Figure 3•28.

- d correspond à la position dans la RAM où se termine le palier nul de la courbe. d peut donc prendre les valeurs entières comprises entre 0 et 1023 incluse. h est la valeur de l'incrément de l'erreur. Cette valeur est déterminée en fonction de la valeur de d. En exécutant cet algorithme pour les 1024 valeurs, on observe que le remplissage de la RAM s'effectue au maximum en 1412 cycles. Effectuer le remplissage de la RAM pour les 16 matières en moins de 600 µs requerrait une fréquence minimale de fonctionnement de 38 MHz. Pour 15 matières, elle devrait être de 36 MHz.

```
Procedure ( d, h, k)
Faire
   t = 32768
   cpt = a = i = 0
   e = val = 0
   TantQue (pas finie) Faire
      Si (e<t) Alors
         Si (ROM[i] == ↑) Alors
            val = val + k
         Sinon
            e = e + h
            Si (e > t) Alors
               RAM[ d ] = (val >> 8)
               e = e - t
               d = d + 1
               Si ((d & 128) == 0)
                  et ((d >> 7) != 7))
               Alors
                  e = e * 2
                  h = h * 2
               FinSi
            Sinon
               RAM[ cpt ] = 0
               cpt = cpt + 1
            FinSi
            a = a +1
            Si ((a & 128) == 0)
               et ((a >> 7) != 7))
            Alors
               h = h / 2
            FinSi
         i = i + 1
         FinSi
      Sinon
         RAM[ d ] = (val >> 8)
         e = e - t
         d = d + 1
         Si ((d & 128) == 0)
            et ((d >> 7) != 7))
         Alors
            e = e * 2
            h = h * 2
         FinSi
      FinSi
   FinPour
Fait
```

Figure 3•28 Algorithme incrémental.

3•3•5•2 En calculant au vol.

• Les architectures proposées jusqu'à présent dans ce chapitre, supposent toutes que l'éclairement est calculé sur tout l'écran et que l'on dispose donc d'un retour trame de 600 µs pendant lequel on peut charger des RAMs sans perdre de temps. Si le processeur d'éclairement devait fonctionner sur des morceaux d'image de taille réduite, le retour trame correspondrait au passage d'un morceau d'image à un autre. Il n'y aurait donc plus d'obligation technologique (retour du faisceau d'électrons à sa position d'origine) imposant sa durée. Le temps de chargement des RAMs doit donc être pris en compte et peut prendre une part importante du temps de calcul de l'éclairement.

• Dans ce cas, l'utilisation de RAM pourrait ne plus être judicieuse et il pourrait alors être préférable d'effectuer les calculs de l'éclairement au vol. L'élévation à la puissance ne reste pas moins le problème majeur et les méthodes d'approximation étudiées vont malgré tout s'avérer utiles. On se proposera d'utiliser une fonction rationnelle du second degré définie en deux parties car elle donne les meilleurs résultats. Le diagramme ci-dessous présente l'architecture de l'unité de calcul du spéculaire qui en découle. L'unité de calcul du produit scalaire n'est pas représentée.

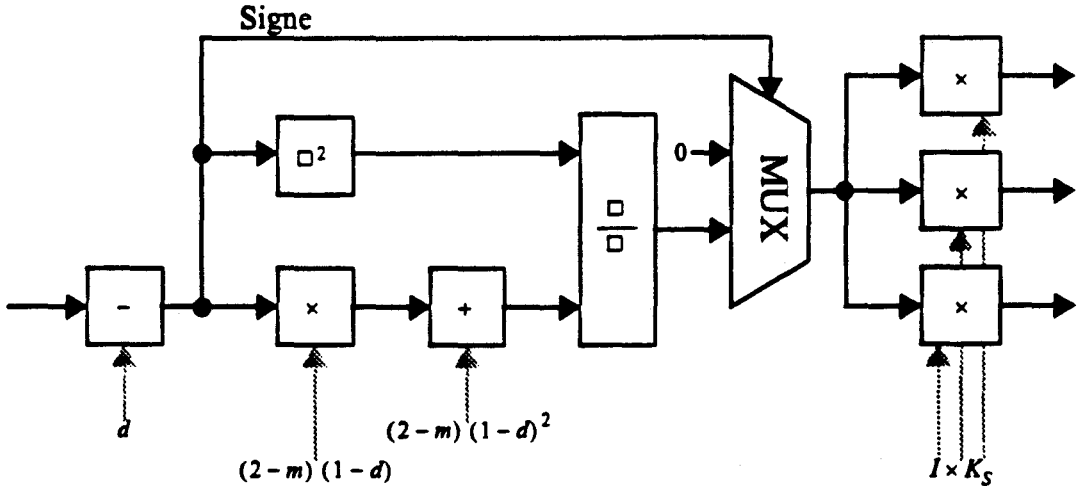


Figure 3•29 Diagramme du calcul au vol de l'éclairement spéculaire.

• L'unité d'arrangement des bits telle qu'elle a été définie ne trouve plus sa place au sein de cette architecture. Mais le souci d'économiser et d'ajuster la précision des calculs au besoin, restant le même, il serait bon d'étudier un système semblable adapté à l'architecture.

• Enfin le choix d'utiliser des fonctions rationnelles du second degré par rapport à celles du premier degré avait était motivé par le peu de différence au niveau des automates de chargement. Ici, l'utilisation d'une fonction rationnelle du premier degré permettrait d'économiser une élévation au carré. Il faudrait donc décider si l'amélioration de la qualité visuelle est justifiée au regard du prix de cette élévation au carré.

• Outre le rétrécissement des zones de l'écran, l'augmentation du nombre de matières, que l'on peut afficher simultanément à l'écran, peut motiver l'abandon des architectures utilisant des RAMs au profit de celles présentées ci-dessus.

3•3•5•3 Conclusion.

Diverses solutions ayant été proposées pour traiter le problème de l'élévation à la puissance dans le calcul de l'éclairement spéculaire, il faut maintenant intégrer ces solutions dans l'architecture d'éclairement constitué de l'unité de normalisation puis des processeurs d'éclairement proprement-dits.

3•4 La première version du processeur d'éclairement.

Pour cette première version du processeur d'éclairement, nous allons profiter au mieux de toutes les possibilités permettant la simplification de l'architecture et ainsi la diminution de son coût. Le processeur d'éclairement sera précédé de l'unité de normalisation de base telle qu'elle a été définie au paragraphe 3•2•3•1 et au paragraphe 3•2•3•2. (selon qu'on utilise, pour le calcul, des multiplications en binaire naturel ou la notation S./L.N.S.)

Comme on l'a montré au chapitre 3, c'est dans le cas où les sources et l'observateur sont à l'infini, que le processeur d'éclairement est le plus simple. C'est dans ce contexte que le premier prototype sera étudié. Le calcul de l'éclairement utilisera donc le vecteur $\hat{H}$. Rappelons que ce vecteur reste constant pour toute l'image. La valeur des composantes du vecteur $\hat{H}$ sera fournie par le microprocesseur hôte du système graphique.

Les restrictions proposées suite à l'étude de l'éclairement spéculaire seront utilisées. Le nombre de matières présentes simultanément à l'écran est réduit à 16.

3•4•1 L'architecture des unités de calcul des produits scalaires.

3•4•1•1 Calcul en binaire naturel.

• Dans un premier temps, étudions la solution la plus simple pour l'élaboration de l'unité de calcul du produit scalaire en supposant que le vecteur de la normale ait été normalisé par une unité utilisant la multiplication en binaire naturel. Que cela soit pour le calcul du diffus ou du spéculaire, c'est un produit scalaire par un vecteur constant (le vecteur $\hat{L}$ ou $\hat{H}$, suivant le cas) qui est calculé. Le calcul des deux produits scalaires se résume donc à l'évaluation de six multiplications par des nombres constants puis aux additions.

• Nous montrerons au chapitre suivant que les composantes du vecteur $\hat{N}$ peuvent être codées sur un nombre limité de bits, sans détériorer la qualité de l'image. On peut donc simplifier les unités de calcul des multiplications. En effet, les vecteurs $\hat{L}$ et $\hat{H}$ étant également constants, on peut profiter des circonstances pour remplacer les vraies unités de multiplication par de simples tables de multiplication. Ces tables contiendront toutes les valeurs résultant de toutes les multiplications possibles entre la valeur absolue d'une composante du vecteur et la valeur absolue de l'autre composante qui reste fixe pendant toute l'image.

La multiplication des signes est effectuée par un "ou exclusif". Le résultat extrait de la RAM est ensuite inversé ou non, suivant le signe résultant.

Les six tables peuvent être chargées en début d'image par additions successives. Le nombre de cycles requis pour charger par un automate câblé l'intégralité des tables dépendra de leurs tailles. Cette solution n'est donc viable qu'à la condition de pouvoir réduire la taille des composantes du vecteur $\hat{N}$. La partie opérative de l'automate utilisé ne requiert qu'un simple additionneur.

• Une unité d'addition décomposée en deux parties réalise l'addition des trois termes pour le produit scalaire. Quand le signe du produit scalaire est négatif, le calcul de l'éclairement impose d'utiliser la valeur 0 à la place du résultat du produit scalaire. L'unité "raz" composée principalement de fonctions logiques "et" effectue la mise à zéro pour les valeurs négatives du résultat. Afin de prendre en compte les éventuels dépassements de capacités induits par les petites erreurs de calcul de l'architecture, un bit supplémentaire est ajouté en sortie de l'additionneur. Ce bit est un bit de déplassement de capacité qui indique que le résultat réel doit prendre la valeur maximale admissible pour le produit scalaire. L'unité "max" composée principalement de fonctions logiques "ou" effectue cette transformation.

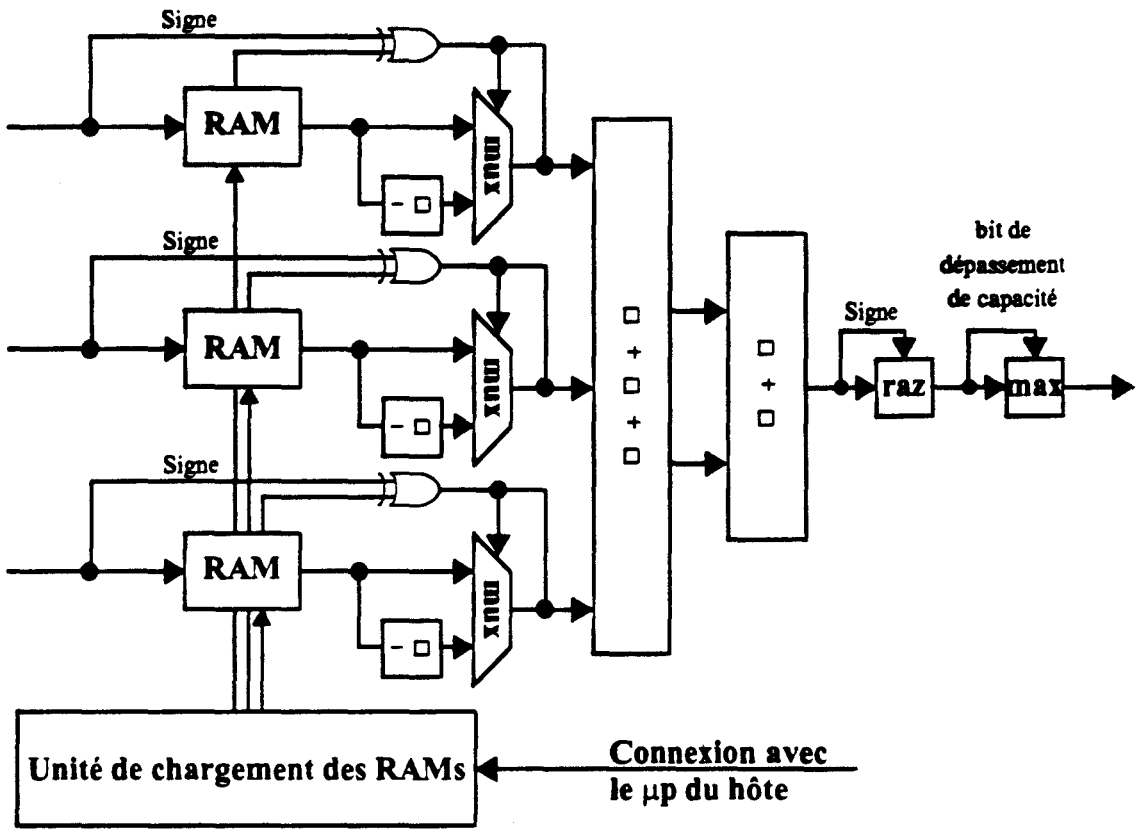


Figure 3•30 Calcul des produits scalaires en utilisant des tables de multiplication.

• Suivant la taille des RAMs et donc la durée du chargement, deux solutions sont envisageables. La première consiste à effectuer le chargement pendant les retours trame. L'autre solution consiste à multiplier par deux le nombre des RAMs. Pendant qu'une RAM est utilisée pour effectuer les multiplications du produit scalaire, l'autre est chargée avec la table de multiplication relative à l'image suivante.

3•4•1•2 Calcul par la notation S./L.N.S.

• Si les composantes des normales sont fournies à l'unité d'éclairement en S./L.N.S., il suffit de modifier le fonctionnement interne de l'unité de chargement des RAMs. Pour effectuer la conversion S./L.N.S. en binaire naturel, une ROM devra contenir la table des correspondances. Pour effectuer la multiplication par V_i, il suffit d'ajouter la valeur de V_i en S./L.N.S. avant de convertir en binaire naturel.

Pour que les RAMs effectuent la multiplication et la conversion, il suffit de calculer cette addition lors du transfert de la ROM vers les RAMs. Afin de diminuer la complexité de chaque RAM, des décaleurs variables peuvent leur être couplés. Seuls les bits servant de mantisse dans le codage S./L.N.S. entrent dans la RAM. Les bits, servant de partie entière dans le codage S./L.N.S., entrent dans le décaleur qui divise par une puissance de 2 le résultat fourni par la RAM.

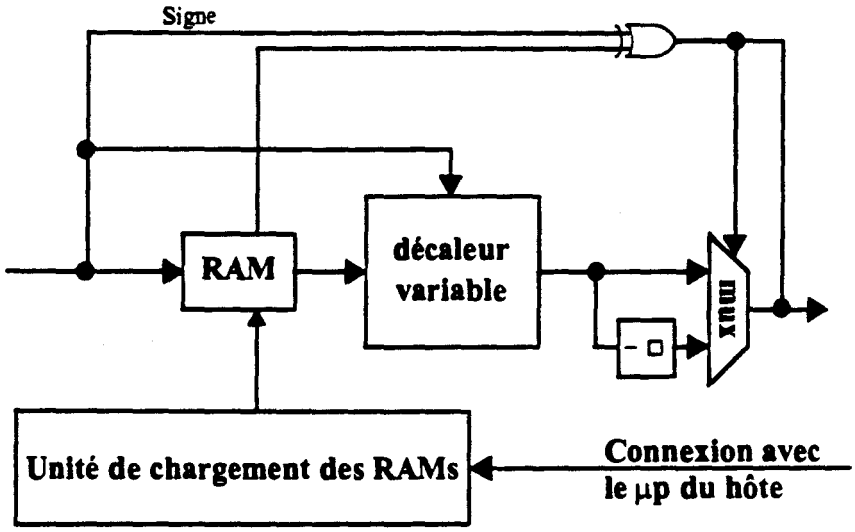


Figure 3•31 Multiplication par une constante et décodage S./L.N.S.

• Pour constituer l'unité de calcul du produit scalaire, on remplace chacun des trois blocs calculant les multiplications dans la Figure 3•30 par le bloc présenté Figure 3•31. Les valeurs S./L.N.S. des composantes des vecteurs $\hat{L}$ et $\hat{H}$ sont transmises par le processeur hôte à l'unité de chargement, après qu'ils ont été normalisés.

• Des deux architectures présentées, la première semble être plus simple, on y économise une ROM et un décaleur variable par rapport à la seconde. Cela introduit un surcoût, au niveau de l'unité de normalisation mais cela devrait être compensé par le gain sur le processeur d'éclairement dès que le nombre de sources devient important.

Cependant, il ne faut pas définitivement abandonner l'étude des solutions utilisant la notation S./L.N.S. Cela pourrait encore s'avérer avantageux pour l'élaboration d'un processeur d'éclairement avec la source ou l'observateur à distance finie. Dans ce cas, il est nécessaire d'intégrer une unité de normalisation dans chaque processeur d'éclairement. Le gain sur la complexité de cette unité peut alors s'avérer plus intéressant que la perte due à la place occupée par une ROM et un décaleur variable.

3•4•2 Détail de la structure de l'unité d'éclairement.

• Une fois, les deux produits scalaires calculés, l'un des résultats est utilisé pour estimer les trois composantes rouge, verte et bleue de la lumière diffuse, pendant que l'autre est utilisé pour calculer les trois composantes de la lumière spéculaire. Pour obtenir les composantes de la lumière spéculaire, on utilise la première des architectures proposées au paragraphe 3•3.

Au paragraphe 3•3•2, il est suggéré d'utiliser une RAM (par composante) dans laquelle on chargera pour chacune des 16 matières, la fonction $I \times Ks \times (x)^E$. L'unité de chargement du spéculaire est constituée d'un automate qui charge les RAMs en fonction d'instructions mises en mémoire dans une ROM de 64 X 1280 bits. Elle contient 64 micro-programmes de 1280 instructions. Chaque micro-programme a en charge le calcul d'un exposant spéculaire différent.

Les autres propositions, telle que l'unité d'arrangement des bits ou l'approximation rationnelle, ont été exposées en vue d'améliorer les qualités des images. Elles fournissent des solutions pour utiliser pleinement la précision fournie par l'unité de calcul du produit scalaire sans accroître démesurément la complexité de l'architecture. Mais cela nécessite, bien sûr, un calcul plus précis de la normalisation et du produit scalaire. Ces améliorations n'ont donc que peu d'utilité dans le cadre de l'élaboration de ce premier prototype du processeur d'éclairement.

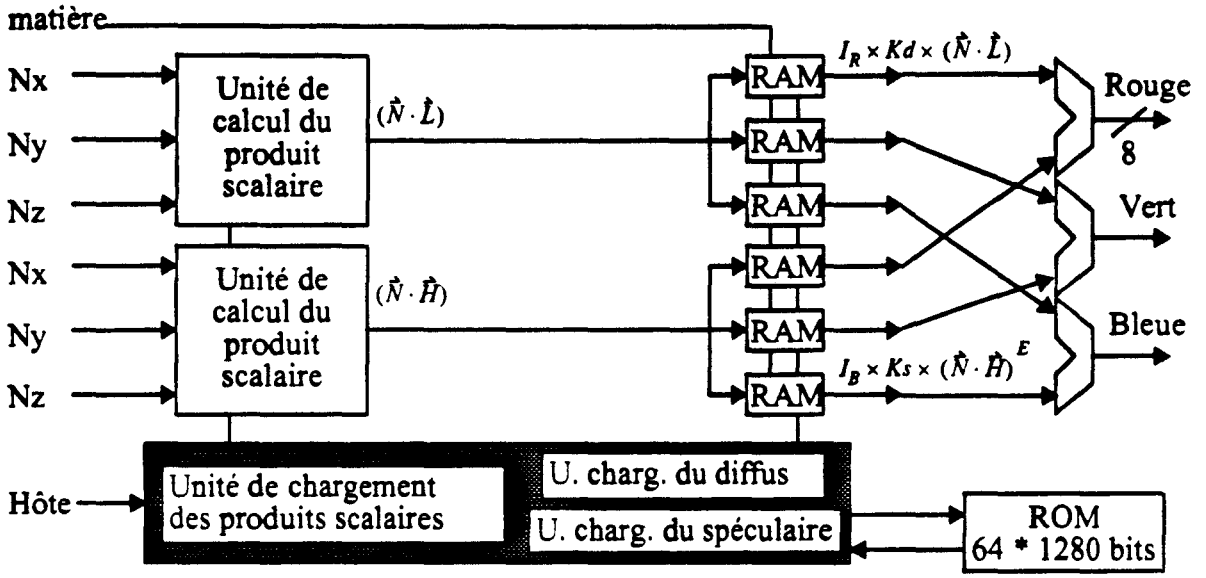


Figure 3•32 Schéma fonctionnel de l'unité d'éclairement.

Pour calculer chacune des trois composantes diffuses, il faut multiplier $(\hat{N}\cdot\hat{L})$ par $I \times Kd$. On emploie l'architecture du spéculaire en simplifiant l'unité de chargement puisque l'algorithme n'utilise ici que des fonctions linéaires. (C'est à dire les fonctions du spéculaire pour lesquelles $E = 1$.) L'unité de chargement du diffus ainsi simplifiée n'a donc, quant à elle, pas besoin de ROM pour opérer.

Finalement, trois additionneurs 8 bits ont en charge de sommer l'intensité diffuse et l'intensité spéculaire pour chacune des trois composantes rouge, verte et bleue

- Les phases de chargement et d'utilisation étant séparées, on pourrait faire en sorte de profiter au maximum de la vitesse lors du calcul de l'éclairement. Avec des SRAMs à 15 ns et des additionneurs construits sous forme de pipelines, on peut espérer atteindre une fréquence de fonctionnement de 50MHz.

3•4•3 Conclusion.

Pour achever cette étude, il faut déterminer les formats des données qui doivent être utilisés. Le chapitre suivant y sera donc consacré. On pourra ainsi reprendre l'architecture proposée dans ce chapitre. Les solutions consistant à remplacer des unités de multiplication par des tables trouveront ainsi des arguments. Les algorithmes de chargement pourront être définitivement conçus et leur temps d'exécution chiffré.

CHAPITRE IV: Développements mathématiques pour l'architecture.

Pour concevoir l'unité d'éclairement dont nous avons commencé la présentation au chapitre précédent, il est nécessaire de connaître la taille des données qui sont manipulées. Il y a d'abord les composantes du vecteur directeur de la normale puis toutes les données intermédiaires intervenant au cours du calcul de l'éclairement et de la normalisation.

Afin de calculer les tailles minimales requises pour ces données, il est nécessaire d'établir des formules régissant la transmission de l'erreur au cours des calculs. Nous établirons ensuite l'erreur maximale admissible pour s'assurer une précision satisfaisante sur les intensités lumineuses finales[1]. L'intensité diffuse et l'intensité spéculaire seront étudiées. Parmi les différentes méthodes évoquées au chapitre II pour le calcul de la composante spéculaire, notre proposition d'implémentation matérielle utilise uniquement le vecteur surbrillance. Les formules concernant les autres méthodes seront, malgré tout, établies en prévision de l'étude ultérieure d'une unité d'éclairement plus générale.

Nous examinerons ensuite la transmission théorique de l'erreur dans une unité idéale[2] de normalisation et nous montrerons l'utilité d'introduire une unité de pré-normalisation. L'organisation effective de cette unité sera présentée au chapitre suivant.

Enfin, pour terminer ce développement nous étudierons, tour à tour, chacune des architectures proposées, au chapitre précédent, pour l'unité de normalisation. Premièrement, nous comparerons les résultats précédents et nous en déduirons l'erreur maximale admissible sur les composantes du vecteur $\hat{N}$ compte tenu des variations de l'exposant spéculaire. Deuxièmement, suite à l'introduction de l'unité de pré-normalisation, nous indiquerons dans quelle mesure la complexité de l'unité de normalisation peut ainsi être réduite sans détériorer la qualité des images. Troisièmement, nous examinerons successivement les propositions d'implémentation de l'unité de normalisation utilisant ou non la notation logarithmique signée. (Celles-ci ont été présentées au chapitre précédent.) Nous cumulerons les erreurs successivement introduites au cours de la normalisation, à chaque fois qu'un bloc fournit son résultat sur une taille limitée. De nombreuses possibilités s'offriront pour déterminer la taille des données sortant de chaque bloc. Il s'agira donc de trouver un compromis permettant de diminuer globalement la complexité de l'architecture de l'unité de normalisation en ajustant chaque taille, l'erreur globale induite devant rester inférieure à la valeur préalablement déterminée afin d'obtenir des images de qualité suffisante.

4•1 La transmission théorique de l'erreur.

Tout au long de ce paragraphe, nous allons déterminer les formules qui régissent la transmission de l'erreur au sein des unités. Celles-ci sont supposées parfaites, c'est à dire qu'elles n'introduisent aucune erreur supplémentaire lors des calculs. Cela va permettre de déterminer, en outre, la précision des valeurs entrant dans l'unité d'éclairement pour obtenir, en sortie, les intensités lumineuses avec la précision requise.

---

1. Dans le cas d'un affichage 16 millions de couleurs, il faut s'assurer une erreur inférieure à un $512^{ème}$ sur les intensités du Rouge, du Vert et du Bleu.
2. C'est à dire : une unité dans laquelle aucune erreur supplémentaire n'est induite du fait de la solution choisie pour l'implémentation des calculs. (Exemple : erreur induite par la taille limitée des données.)

4•1•1 Au cours de la normalisation d'un vecteur.

• Déterminons la relation qui donne l'erreur après normalisation, en fonction de l'erreur héritée avant normalisation.

Soit n_x, n_y et n_z les composantes du vecteur $\vec{N}$ avant normalisation. Soit N_x, N_y et N_z les composantes du vecteur $\vec{N}$ après normalisation. On a :

$$N_x = \frac{n_x}{\|\vec{n}\|} \qquad N_y = \frac{n_y}{\|\vec{n}\|} \qquad N_z = \frac{n_z}{\|\vec{n}\|}$$

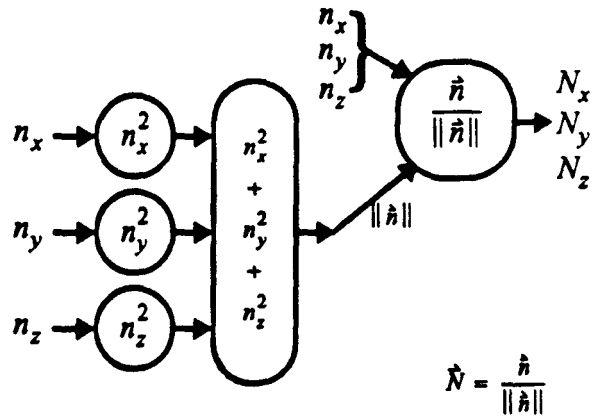


Figure 4•1 Diagramme de la normalisation d'un vecteur.

Après utilisation du logarithme et de la dérivation, on obtient pour chaque composante :

$$\frac{\partial N_i}{N_i} = \frac{\partial n_i}{n_i} - \frac{\partial \|\vec{n}\|}{\|\vec{n}\|}.$$

On peut en déduire : $\partial N_i = \dfrac{\partial n_i - N_i \times \partial \|\vec{n}\|}{\|\vec{n}\|}$ et chercher ensuite la formule sur les erreurs.

Posons δN_x, δN_y, et δN_z, les erreurs respectives de N_x, N_y et N_z.
Posons δn_x, δn_y, et δn_z, les erreurs respectives de n_x, n_y et n_z.

$$\delta N_i \le \frac{\delta n_i + |N_i| \times \delta \|\vec{n}\|}{\|\vec{n}\|} \quad \Rightarrow \quad Sup(\delta N_x, \delta N_y, \delta N_z) \le \frac{Sup(\delta n_x, \delta n_y, \delta n_z) + Sup(N_x, N_y, N_z) \times \delta \|\vec{n}\|}{\|\vec{n}\|}$$

Soit $\delta N = Sup(\delta N_x, \delta N_y, \delta N_z)$ et $\delta n = Sup(\delta n_x, \delta n_y, \delta n_z)$.

Après simplification, on obtient : $\boxed{\delta N \le \dfrac{\delta n + \delta \|\vec{n}\|}{\|\vec{n}\|}}$.

Reste à remplacer $\delta \|\vec{n}\|$ par sa valeur en fonction de δn.

$$\partial \|\vec{n}\| = \frac{\|\vec{n}\|}{2} \times [2n_x \times \partial n_x + 2n_y \times \partial n_y + 2n_z \times \partial n_z]$$

$$\delta \|\vec{n}\| \le \|\vec{n}\| \times |n_x + n_y + n_z| \times Sup(\delta n_x, \delta n_y, \delta n_z)$$

Lemme : La valeur maximale prise par la somme des valeurs absolues des composantes d'un vecteur normé est $\sqrt{3}$.

Démonstration :

$$\text{Soit } F: \begin{cases} \{(X, Y), X \ge 0, Y \ge 0, Z \ge 0, (X^2 + Y^2 + Z^2 = 1)\} \to \Re \\ X, Y \to X + Y + Z \end{cases} \Rightarrow F: \begin{cases} \{(X, Y), X \ge 0, Y \ge 0, X^2 + Y^2 \le 1\} \to \Re \\ X, Y \to X + Y + \sqrt{1 - X^2 - Y^2} \end{cases}$$

Déterminons la valeur maximum prise par la fonction $F(X, Y)$.

$$Sup(F(X, Y)) = Max\left(\begin{array}{l} \left\{ F(X, Y) / \left(\frac{\partial}{\partial X} F(X, Y) = \frac{\partial}{\partial Y} F(X, Y) = 0 \right) \right\} \cup \{F(0, 0)\} \\ \cup \left\{ F(X, Y) / \left(\left(\frac{\partial}{\partial X} F(X, Y) = 0 \right) \wedge (X^2 + Y^2 = 1) \right) \right\} \end{array} \right)$$

$$(1)\quad \begin{cases} \frac{\partial}{\partial X}F(X,Y) = 1 + \frac{X}{\sqrt{1-X^2-Y^2}} = 0 \\ \frac{\partial}{\partial Y}F(X,Y) = 1 + \frac{Y}{\sqrt{1-X^2-Y^2}} = 0 \end{cases} \Rightarrow \begin{cases} 2X^2 + Y^2 = 0 \\ X^2 + 2Y^2 = 0 \end{cases} \Rightarrow \begin{cases} X^2 = \frac{1}{3} \\ Y^2 = \frac{1}{3} \end{cases} \Rightarrow \begin{cases} X = \frac{\sqrt{3}}{3} \\ Y = \frac{\sqrt{3}}{3} \end{cases} \Rightarrow F(X,Y) = \sqrt{3}$$

$$(2)\quad F(X,Y) = 1$$

$$(3)\quad \begin{cases} (X^2 + Y^2 = 1) \Rightarrow \left(F(X,Y) = X + \sqrt{1+X^2} = f(X)\right) \\ \left(\frac{df}{dX}(X) = 1 + \frac{X}{\sqrt{1+X^2}} = 0\right) \Rightarrow \left(X = \frac{\sqrt{2}}{2}\right) \Rightarrow \left(f(\frac{\sqrt{2}}{2}) = \sqrt{2}\right) \Rightarrow \left(F(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}) = \sqrt{2}\right) \end{cases}$$

Et on obtient :

D'où, on en déduit la valeur maximale de la fonction $F(X,Y)$.

$$Sup(F(X,Y)) = Max\{\sqrt{3}, 1, \sqrt{2}\} = \boxed{\sqrt{3} = Sup\{(X+Y+Z); (X^2+Y^2+Z^2=1)\}}$$

Donc : $|n_x + n_y + n_z| \le \sqrt{3} \times \|\hat{n}\|$ et

$$\boxed{\delta \|\hat{n}\| \le \|\hat{n}\|^2 \times \sqrt{3} \times \delta n}.$$

D'où : $\boxed{\delta N \le \left[\frac{1}{\|\hat{n}\|} + \sqrt{3}\|\hat{n}\|\right] \times \delta n}$.

Soit $f(x) = \frac{1}{x} + \sqrt{3} \times x$. Alors $\delta N \le f(\|\hat{n}\|) \times \delta n$

et $\delta N \le Sup(f(\|\hat{n}\|)) \times \delta n = K \times \delta n$.

- Comme le montre le tracé de la fonction, plus les valeurs de $\|\hat{n}\|$ sont prises dans un grand intervalle, plus K a une valeur élevée.

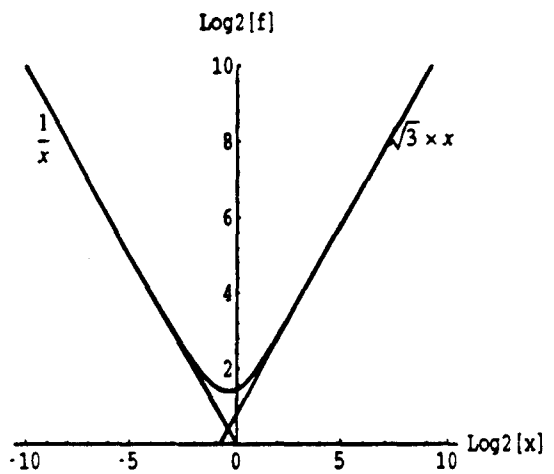


Figure 4•2 Graphe de la fonction $f(x)$.

4•1•2 La transmission de l'erreur dans le produit scalaire.

4•1•2•1 Dans le cas d'un vecteur normé.

- Calculons la transmission de l'erreur dans le cas du calcul du produit scalaire entre un vecteur $\hat{A}$, normé, constant, et un vecteur $\hat{B}$, normé mais dont les composantes présentent une certaine erreur par rapport à leurs valeurs effectives.

Notons $\delta\hat{B}$ le vecteur erreur entre le vrai vecteur $\hat{B}$ et celui qui est fourni $\hat{B} + \delta\hat{B}$. Soient ∂B_x, ∂B_y et ∂B_z les composantes de $\delta\hat{B}$. Enfin, posons $PS(\hat{A}, \hat{B})$ le produit scalaire de A et B.

$$PS(\hat{A}, \hat{B}) = A_x \cdot B_x + A_y \cdot B_y + A_z \cdot B_z$$

$$PS(\hat{A}, \hat{B} + \delta\hat{B}) = PS(\hat{A}, \hat{B}) + (A_x \cdot \partial B_x + A_y \cdot \partial B_y + A_z \cdot \partial B_z)$$

$$|PS(\hat{A}, \hat{B} + \delta\hat{B}) - PS(\hat{A}, \hat{B})| \le |Sup(\partial B_x, \partial B_y, \partial B_z)| \times |A_x + A_y + A_z| \le |Sup(\partial B_x, \partial B_y, \partial B_z)| \times (|A_x| + |A_y| + |A_z|)$$

Le vecteur $\hat{A}$ est normé, donc ces composantes obéissent à l'égalité suivante : $A_x^2 + A_y^2 + A_z^2 = 1$

Au paragraphe 4•1•1, on a montré que $|a+b+c| \le \sqrt{3}$ si $a^2 + b^2 + c^2 = 1$.

Donc : $|PS(\hat{A}, \hat{B} + \delta\hat{B}) - PS(\hat{A}, \hat{B})| \le |Sup(\partial B_x, \partial B_y, \partial B_z)| \times \sqrt{3}$. Soit $\delta B = |Sup(\delta B_x, \delta B_y, \delta B_z)|$, on obtient alors $\boxed{\delta PS(\hat{A}, \hat{B}) \le \sqrt{3} \times \delta B}$.

4•1•2•2 Dans le cas d'un vecteur non normé.

• Calculons la transmission de l'erreur dans le cas du calcul du produit scalaire entre un vecteur $\vec{A}$, normé, constant, et un vecteur $\vec{B}$, normé. Soit $\vec{b}$ le vecteur correspondant à $\vec{B}$ mais non normé. Les composantes de ce vecteur présentent une certaine erreur par rapport à leurs valeurs effectives.

$$\vec{A}\cdot\vec{B} = \vec{A}\cdot\frac{\vec{b}}{\|\vec{b}\|} = PS(\vec{A},\vec{B}).$$

Soit δPS l'erreur faite sur le produit scalaire PS.

Soit b_x, b_y et b_z les composantes du vecteur $\vec{b}$ (avant normalisation).

Soit A_x, A_y et A_z les composantes du vecteur constant $\vec{A}$.

On a $PS = \vec{A}\cdot\frac{\vec{b}}{\|\vec{b}\|} = \frac{A_x\times b_x}{\|\vec{b}\|}+\frac{A_y\times b_y}{\|\vec{b}\|}+\frac{A_z\times b_z}{\|\vec{b}\|}$.

Après utilisation du logarithme et de la dérivation, on obtient :

$$\partial PS = \left(\frac{A_x\times b_x}{\|\vec{b}\|}\times[\partial b_x-\partial\|\vec{b}\|]\right)+\left(\frac{A_y\times b_y}{\|\vec{b}\|}\times[\partial b_y-\partial\|\vec{b}\|]\right)+\left(\frac{A_z\times b_z}{\|\vec{b}\|}\times[\partial b_z-\partial\|\vec{b}\|]\right)$$

$$\partial PS = \left(\frac{A_x\times b_x}{\|\vec{b}\|}\right)\times\partial b_x+\left(\frac{A_y\times b_y}{\|\vec{b}\|}\right)\times\partial b_y+\left(\frac{A_z\times b_z}{\|\vec{b}\|}\right)\times\partial b_z-\left(\frac{A_x\times b_x}{\|\vec{b}\|}+\frac{A_y\times b_y}{\|\vec{b}\|}+\frac{A_z\times b_z}{\|\vec{b}\|}\right)\times\partial\|\vec{b}\|$$

$$\partial PS = \left(\frac{b_x}{\|\vec{b}\|}\times A_x\right)\times\partial b_x+\left(\frac{b_y}{\|\vec{b}\|}\times A_y\right)\times\partial b_y+\left(\frac{b_z}{\|\vec{b}\|}\times A_z\right)\times\partial b_z-PS\times\partial\|\vec{b}\|$$

Remplaçons les accroissements par leurs erreurs :

$$\delta PS\le\left|\frac{b_x}{\|\vec{b}\|}\times A_x\right|\times\delta b_x+\left|\frac{b_y}{\|\vec{b}\|}\times A_y\right|\times\delta b_y+\left|\frac{b_z}{\|\vec{b}\|}\times A_z\right|\times\delta b_z+|PS|\times\delta\|\vec{b}\|$$

Soit $\delta b = Sup(\delta b_x,\delta b_y,\delta b_z)$. Au paragraphe 4•1•1, on a montré que $\delta\|\vec{b}\|\le\|\vec{b}\|^2\times\sqrt{3}\times\delta b$

$$\delta PS\le\left|\frac{b_x}{\|\vec{b}\|}\times A_x\right|\times\delta b_x+\left|\frac{b_y}{\|\vec{b}\|}\times A_y\right|\times\delta b_y+\left|\frac{b_z}{\|\vec{b}\|}\times A_z\right|\times\delta b_z+\left(|PS|\times\|\vec{b}\|^2\times\sqrt{3}\right)\times\delta b$$

$$\delta PS\le\left[\left(\frac{|b_x|}{\|\vec{b}\|}\times|A_x|\right)+\left(\frac{|b_y|}{\|\vec{b}\|}\times|A_y|\right)+\left(\frac{|b_z|}{\|\vec{b}\|}\times|A_z|\right)\right]\times\delta b+\left(|PS|\times\|\vec{b}\|^2\times\sqrt{3}\right)\times\delta b$$

On a $\left(\frac{|b_x|}{\|\vec{b}\|}\times|A_x|\right)+\left(\frac{|b_y|}{\|\vec{b}\|}\times|A_y|\right)+\left(\frac{|b_z|}{\|\vec{b}\|}\times|A_z|\right) = \frac{\begin{bmatrix}|b_x|\\|b_y|\\|b_z|\end{bmatrix}\cdot\begin{bmatrix}|A_x|\\|A_y|\\|A_z|\end{bmatrix}}{\sqrt{|b_x|^2+|b_y|^2+|b_z|^2}}\le 1$, $|PS|\le 1$ et $\|\vec{b}\|^2\le 3$.

Donc $\boxed{\delta PS\le(1+3\times\sqrt{3})\times\delta b<6{,}2\times\delta b}$.

4•1•3 Au cours du calcul du diffus.

• On calcule l'éclairement diffus en utilisant la formule ci-contre. Partant de la valeur de la normale, on obtient l'intensité diffuse en utilisant un produit scalaire puis une multiplication. Dans le cas du calcul de l'intensité lumineuse pour estimer la transmission de l'erreur à partir de celle héritée du calcul de la normale, il faut examiner le comportement d'un produit scalaire sur une erreur.

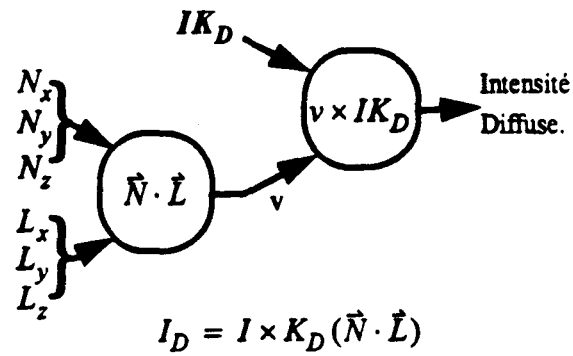


Figure 4•3 Diagramme du calcul de l'éclairement diffus.

4•1•3•1 La transmission de l'erreur depuis le vecteur normé.

• En utilisant la formule de transfert de l'erreur sur le produit scalaire, on calcule la transmission de l'erreur sur l'éclairement diffus. On définit $\delta N = |Sup(\delta N_x, \delta N_y, \delta N_z)|$.

$I_D(\hat{N}) = IK_D(\hat{N}, \hat{L}) = IK_D(PS(\hat{N}, \hat{L}))$ donc $\delta I_D \leq \sqrt{3} IK_D \times \delta N$ et comme $IK_D \leq 1$, on obtient :

$$\boxed{\delta I_D \leq \sqrt{3} IK_D \times \delta N \leq \sqrt{3} \times \delta N}.$$

• Si on veut que la composante diffuse de l'intensité lumineuse soit calculée avec une précision de $1/256^{ème}$ par rapport à l'intensité maximum (intensité = 1), c'est à dire que la partie décimale de l'intensité lumineuse soit codée sur 8 bits, il faut une précision d'au moins $1/512^{ème}$ sur les composantes du vecteur $\hat{N}$ normé. Il faut donc que les composantes du vecteur $\hat{N}$ normé soient obtenues avec 9 bits significatifs pour la partie fractionnaire.

4•1•3•2 La transmission de l'erreur depuis le vecteur non normé.

• En incorporant la transmission de l'erreur au cours de la normalisation dans la formule précédente, on obtient :

$$\boxed{\delta I_D \leq \left[\frac{\sqrt{3}}{\|\vec{n}\|} + 3\|\vec{n}\|\right] \times \delta n}.$$

4•1•4 Au cours du calcul du spéculaire par le vecteur réfléchi.

• Quand on utilise le vecteur réfléchi, on calcule l'éclairement spéculaire par la formule :

$$I_S = I \times K_S[(2(\vec{N}\cdot\vec{L})\vec{N} - \vec{L})\cdot\vec{O}]^e$$

Restreignons-nous simplement au cas de la projection orthogonale où l'on rejette l'observateur à l'infini. Le vecteur $\vec{O}$ normé est donc constant, unitaire, dirigé suivant l'axe Oz. Cela permet de simplifier la formule.

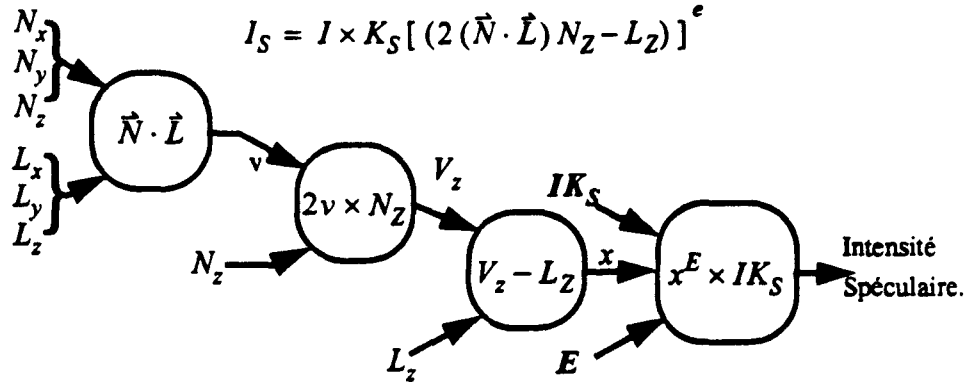


Figure 4•4 Diagramme du calcul de l'éclairement spéculaire en utilisant le vecteur réfléchi.

4•1•4•1 La transmission de l'erreur depuis le vecteur normé.

• En utilisant un développement limité au premier ordre, on obtient une estimation de l'erreur faite sur l'éclairement spéculaire, en fonction de l'erreur commise sur les valeurs de chaque composante des normales rendues unitaires.

$$I_S(\hat{N}) = [R_z]^E \times IK_S = [\,(N_x \cdot L_x + N_y \cdot L_y + N_z \cdot L_z) \times 2N_z - L_z]^E \times IK_S$$

$$\left|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})\right| \le \left|2E \cdot IK_S \times [\,(N_x \cdot L_x + N_y \cdot L_y + N_z \cdot L_z) \times 2N_z - L_z]^{E-1} \times N_z\right| \times \left|L_x \delta N_x + L_y \delta N_y + 2L_z \delta N_z\right|$$

$$\left|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})\right| \le 2E \cdot IK_S \times [\,|R_z|]^{E-1} \times |N_z| \times |Sup(\delta N_x, \delta N_y, \delta N_z)| \times |L_x + L_y + 2L_z|$$

Le vecteur $\hat{L}$ est normé, donc ses composantes obéissent à l'égalité : $L_x^2 + L_y^2 + L_z^2 = 1$.

$$\text{Soit F: } \begin{cases} \{(X,Y), X \ge 0, Y \ge 0, Z \ge 0, (X^2+Y^2+Z^2=1)\} \to \Re \\ X, Y \to X+Y+2Z \end{cases} \Rightarrow \text{F: } \begin{cases} \{(X,Y), X \ge 0, Y \ge 0, X^2+Y^2 \le 1\} \to \Re \\ X, Y \to X+Y+2\sqrt{1-X^2-Y^2} \end{cases}$$

Déterminons la valeur maximale prise par la fonction $F(X,Y)$.

$$Sup(F(X,Y)) = Max\left(\left\{F(X,Y) / \left(\frac{\partial}{\partial X}F(X,Y) = \frac{\partial}{\partial Y}F(X,Y) = 0\right)\right\} \cup \{F(0,0)\} \cup \left\{F(X,Y) / \left(\left(\frac{\partial}{\partial X}F(X,Y) = 0\right) \wedge (X^2+Y^2=1)\right)\right\}\right)$$

$$(1) \quad \begin{cases} \frac{\partial}{\partial X}F(X,Y) = 1 + \frac{2X}{\sqrt{1-X^2-Y^2}} = 0 \\ \frac{\partial}{\partial Y}F(X,Y) = 1 + \frac{2Y}{\sqrt{1-X^2-Y^2}} = 0 \end{cases} \Rightarrow \begin{cases} 5X^2+Y^2=0 \\ X^2+5Y^2=0 \end{cases} \Rightarrow \begin{cases} X^2 = \frac{1}{6} \\ Y^2 = \frac{1}{6} \end{cases} \Rightarrow \begin{cases} X = \frac{\sqrt{6}}{6} \\ Y = \frac{\sqrt{6}}{6} \end{cases} \Rightarrow F(X,Y) = \sqrt{6}$$

$$(2) \quad F(X,Y) = 2$$

$$(3) \quad \begin{cases} (X^2+Y^2=1) \Rightarrow \left(F(X,Y) = X + \sqrt{1+X^2} = f(X)\right) \\ \left(\frac{\partial f}{\partial X} = 1 + \frac{X}{\sqrt{1+X^2}} = 0\right) \Rightarrow \left(X = \frac{\sqrt{2}}{2}\right) \Rightarrow \left(f\left(\frac{\sqrt{2}}{2}\right) = \sqrt{2}\right) \Rightarrow \left(F\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right) = \sqrt{2}\right) \end{cases}$$

D'où, on en déduit la valeur maximale de la fonction $F(X,Y)$.

$$Sup(F(X,Y)) = Max\{\sqrt{6}, 2, \sqrt{2}\} = \boxed{\sqrt{6} = Sup\{(X+Y+2Z); (X^2+Y^2+Z^2=1)\}}$$

Donc $\left|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})\right| \le 2E \cdot IK_S \times [\,|R_z|]^{E-1} \times |N_z| \times |Sup(\delta N_x, \delta N_y, \delta N_z)| \times \sqrt{6}$

$0 \le IK_S \le 1$

Soit $\delta N = |Sup(\delta N_x, \delta N_y, \delta N_z)|$. Et comme $\begin{cases} 0 \le [\,|R_z|]^{E-1} \le 1 \\ 0 \le |N_z| \le 1 \end{cases}$, on obtient : $\boxed{\delta I_S \le E\sqrt{24}\,\delta N}$

• On obtient la valeur théorique maximum du facteur multiplicatif sur l'erreur d'entrée lorsque le calcul est effectué d'un bloc. D'autres erreurs peuvent être introduites par le découpage fonctionnel du calcul, chaque entité pouvant induire une erreur supplémentaire dont la cause serait la précision insuffisante des valeurs intermédiaires.

En décomposant le calcul en deux entités (le produit scalaire puis le reste du calcul), on obtient la formule d'erreur suivante :

$$I_S(\hat{N}) = [R_z]^E \times IK_S = [(PS(\hat{N},\hat{L})) \times 2N_z - L_z]^E \times IK_S = I_S(N_z, PS(\hat{N},\hat{L}))$$

$$I_S(N_z+\partial N_z, PS+\partial PS) \approx I_S(N_z, PS) + IK_S \times 2 \cdot E \times [PS \times 2N_z - L_z]^{E-1} \times (\partial PS \cdot N_z + \partial N_z \cdot PS)$$

$$|I_S(N_z+\partial N_z, PS+\partial PS) - I_S(N_z, PS)| \le |2E \cdot IK_S \times [PS \times 2N_z - L_z]^{E-1}| \times |\delta PS \cdot N_z + \delta N_z \cdot PS|$$

$$|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})| \le 2E \cdot IK_S \times [|R_z|]^{E-1} \times (|N_z|\delta PS + |PS|\delta N_z)$$

Comme : $\begin{cases} 0 \le IK_S \le 1 & 0 \le |N_z| \le 1 \\ 0 \le [|R_z|]^{E-1} \le 1 & 0 \le |PS| \le 1 \end{cases}$, on obtient : $\boxed{\delta I_S \le 2E(\delta PS + \delta N_z)}$.

4•1•4•2 La transmission de l'erreur depuis le vecteur non normé.

En incorporant la transmission de l'erreur au cours de la normalisation dans la première des deux formules précédentes, on obtient :

$$\boxed{\delta I_S \le \left[\frac{\sqrt{24}}{\|\hat{n}\|} + 6\sqrt{2}\|\hat{n}\|\right] \times E \times \delta n}$$

4•1•5 <u>Au cours du calcul du spéculaire par le vecteur réfléchi inverse.</u>

• Quand on utilise le vecteur réfléchi inverse, on calcule l'éclairement spéculaire par la formule :

$$I_S = I \times K_S[(2(\vec{N}\cdot\vec{O})\vec{N} - \vec{O}) \cdot \vec{L}]^E$$

Restreignons-nous simplement au cas de la projection orthogonale où l'on rejette l'observateur à l'infini. Le vecteur $\vec{O}$ normé est donc constant, unitaire, dirigé suivant l'axe Oz. Cela permet de simplifier la formule.

$$I_S = I \times K_S[(2N_Z \times \vec{N} - \vec{O}) \cdot \vec{L}]^E$$

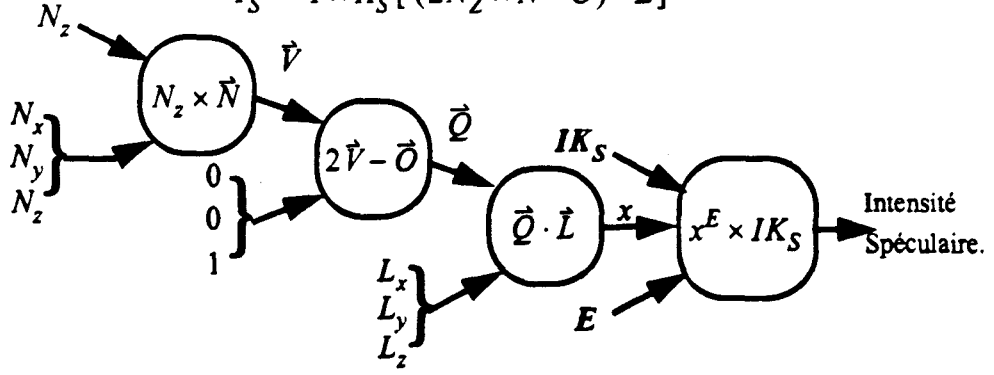


Figure 4•5 Diagramme du calcul de l'éclairement spéculaire en utilisant le vecteur réfléchi inverse.

• En utilisant un développement limité au premier ordre, on obtient une estimation de l'erreur faite sur l'éclairement spéculaire en fonction de celle commise sur les valeurs de chaque composante des normales rendues unitaires.

$$I_S(\hat{N}) = [\vec{Q}\cdot\hat{L}]^E \times IK_S = [2N_x\times N_z\times L_x + 2N_y\times N_z\times L_y + (2N_z^2-1)\times L_z]^E \times IK_S$$

$$|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})| \le |E\cdot IK_S\times[\vec{Q}\cdot\hat{L}]^{E-1}| \times |[2N_xL_x]\delta N_x + [2N_xL_y]\delta N_y + [2N_xL_x + 2N_yL_y + 4N_zL_z]\delta N_z|$$

$$|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})| \le |E\cdot IK_S\times[\vec{Q}\cdot\hat{L}]^{E-1}| \times 2|N_zL_x + N_zL_y + N_xL_x + N_yL_y + 2N_zL_z| \times |Sup(\delta N_x, \delta N_y, \delta N_z)|$$

$$|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})| \le |E\cdot IK_S\times[\vec{Q}\cdot\hat{L}]^{E-1}| \times |Sup(\delta N_x, \delta N_y, \delta N_z)| \times 2|\hat{N}\cdot\hat{L} + N_z[L_x+L_y+L_z]|$$

$$|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})| \le |E\cdot IK_S\times[\vec{Q}\cdot\hat{L}]^{E-1}| \times |Sup(\delta N_x, \delta N_y, \delta N_z)| \times 2(1 + |N_z[L_x+L_y+L_z]|)$$

Le vecteur $\hat{L}$ est normé, donc ses composantes obéissent à l'égalité : $L_x^2 + L_y^2 + L_z^2 = 1$.

On a montré au paragraphe 4•1•2•1 que : $\sqrt{3} = Sup\{(X+Y+Z);(X^2+Y^2+Z^2=1)\}$.

Donc $|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})| \le |E\cdot IK_S\times[\vec{Q}\cdot\hat{L}]^{E-1}| \times |Sup(\delta N_x, \delta N_y, \delta N_z)| \times 2|1 + N_z\sqrt{3}|$

Soit $\delta N = |Sup(\delta N_x, \delta N_y, \delta N_z)|$. Et comme $\begin{cases} 0 \le IK_S \le 1 \\ 0 \le [\vec{Q}\cdot\hat{L}]^{E-1} \le 1 \\ 0 \le |N_z| \le 1 \end{cases}$, on obtient : $\boxed{\delta I_S \le E(2+2\sqrt{3})\delta N}$

• La formule que l'on obtient ci dessus nous montre que nous avons besoin d'une précision, sur la valeur du vecteur normé de la normale, d'autant plus importante que l'exposant spéculaire a une valeur élevée.

4•1•6 Au cours du calcul du spéculaire par le vecteur surbrillance.

• On calcule l'éclairement diffus en utilisant la formule détaillée ci-contre. Partant de la valeur de la normale, on obtient l'intensité diffuse en utilisant un produit scalaire puis une élévation à la puissance et une multiplication. Le calcul de l'erreur va donc utiliser la formule de transmission de l'erreur par un produit scalaire et un développement limité afin de résoudre la transmission de l'erreur pour l'élévation à la puissance.

$$I_S = I\times K_S[\hat{N}\cdot\hat{H}]^e$$

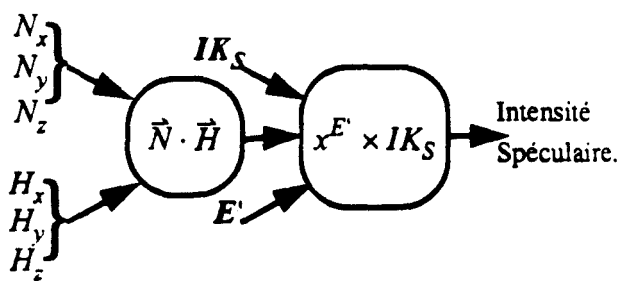


Figure 4•6 Diagramme du calcul de l'éclairement spéculaire en utilisant le vecteur surbrillance.

4•1•6•1 La transmission de l'erreur depuis le vecteur normé.

• En utilisant un développement limité au premier ordre, on obtient une estimation de l'erreur faite sur l'éclairement spéculaire en fonction de celle commise sur les valeurs de chaque composante des normales rendues unitaires.

$$I_S(\hat{N}) = [N_x\cdot H_x + N_y\cdot H_y + N_z\cdot H_z]^{E'} \times IK_S$$

$$I_S(\hat{N}+\partial\hat{N}) \approx I_S(\hat{N}) + IK_S\times E'\times[N_x\cdot H_x + N_y\cdot H_y + N_z\cdot H_z]^{E'-1}\times(H_x\partial N_x + H_y\partial N_y + H_z\partial N_z)$$

$$|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})| \le |E'\cdot IK_S\times[N_x\cdot H_x + N_y\cdot H_y + N_z\cdot H_z]^{E'-1}| \times |H_x\delta N_x + H_y\delta N_y + H_z\delta N_z|$$

$$|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})| \le E'\cdot IK_S\times[|N_x\cdot H_x + N_y\cdot H_y + N_z\cdot H_z|]^{E'-1} \times |Sup(\delta N_x, \delta N_y, \delta N_z)| \times |H_x + H_y + H_z|$$

Le vecteur $\hat{H}$ est normé donc ces composantes obéissent à l'équation suivante : $H_x^2 + H_y^2 + H_z^2 = 1$

On sait que : $Sup\{(X+Y+Z)/(X^2+Y^2+Z^2=1)\} = \sqrt{3}$ (Cf §4•1•1)

Donc : $|I_S(\hat{N}+\delta\hat{N}) - I_S(\hat{N})| \le E' \cdot IK_S \times [|N_x \cdot H_x + N_y \cdot H_y + N_z \cdot H_z|]^{E'-1} \times |Sup(\delta N_x, \delta N_y, \delta N_z)| \times \sqrt{3}$

Soit $\delta N = |Sup(\delta N_x, \delta N_y, \delta N_z)|$. Et comme $\begin{cases} 0 \le IK_S \le 1 \\ 0 \le [|N_x \cdot H_x + N_y \cdot H_y + N_z \cdot H_z|]^{E'-1} \le 1 \end{cases}$

on obtient : $\boxed{\delta I_S \le E' \sqrt{3} \delta N}$.

• La formule que l'on obtient ci dessus nous montre que nous avons besoin d'une précision, sur la valeur du vecteur normé de la normal, d'autant plus importante que l'exposant spéculaire a une valeur élevée. Néanmoins, il ne faut pas perdre de vue que le modèle de Blinn est un modèle empirique, comme le modèle de Phong. Il n'est donc pas nécessaire de disposer d'une précision aussi importante dès lors que les erreurs n'introduisent pas d'aberration visuelle sensible.

4•1•6•2 La transmission de l'erreur depuis le vecteur non normalisé.

En incorporant la transmission de l'erreur au cours de la normalisation dans la formule précédente, on obtient :

$$\boxed{\delta I_S \le \left[\frac{\sqrt{3}}{\|\vec{n}\|} + 3\|\vec{n}\|\right] \times E' \times \delta n}$$

4•2 L'Unité de Pré-Normalisation.

4•2•1 Sa nécessité.

Dans le but de réduire la complexité matérielle de l'unité de normalisation, il faut donc tenter de diminuer la taille des mots utilisés pour coder les composantes du vecteur $\hat{N}$.

4•2•1•1 Le format théorique de sortie de l'unité de normalisation.

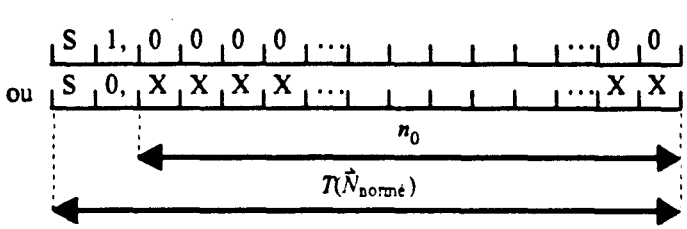
Au paragraphe 4•1, nous avons calculé l'influence de la précision pour le vecteur $\hat{N}$ normé sur la qualité de l'image. On a obtenu les formules suivantes :

Soit $\delta N = |Sup(\delta N_x, \delta N_y, \delta N_z)|$ la précision pour le vecteur $\hat{N}$ normé.

On a pour le diffus $\boxed{\delta I_D \le \sqrt{3} IK_D \times \delta N \le \sqrt{3} \times \delta N}$

pour le spéculaire $\boxed{\delta I_S \le E\sqrt{24}\delta N}$ ou $\boxed{\delta I_S \le E' \sqrt{3}\delta N}$ selon la méthode employée.

Après normalisation, les composantes du vecteur $\hat{N}$ sont comprises entre -1 et 1. Le codage des composantes nécessite donc 1 bit de signe et 1 bit pour la partie entière. On ajuste ensuite n_0 le nombre de bits utilisés en partie décimale pour obtenir l'erreur voulue



Exemple :

$\delta I_D \le \sqrt{3} \times \frac{0,5}{2^{n_0}} \le \delta I_{D\_MAX}$

$\Rightarrow$

$n_0 \ge -Log_2(0,5 \times \sqrt{3} \times \delta I_{D\_MAX})$

Figure 4•7 Codage des composantes du vecteur $\hat{N}$ normalisé.

4•2•1•2 Le format théorique d'entrée de l'unité de normalisation.

• De même que nous venons de déterminer la taille des composantes du vecteur $\vec{N}$ après normalisation, il faut établir la taille $\pi(\vec{n})$ des mots que l'on doit utiliser en entrée de l'unité de normalisation. Au paragraphe 4•1•1, nous avons prouvé que :

$$\delta N \leq Sup(f(\|\vec{n}\|)) \times \delta n = K \times \delta n \text{ avec } f(x) = \frac{1}{x} + \sqrt{3} \times x.$$

• Comme le montre le tracé de la fonction, plus les valeurs de $\|\vec{n}\|$ sont prises dans un grand intervalle, plus K a une valeur élevée. Pour diminuer la valeur de K, il faudrait borner l'intervalle des valeurs de $\|\vec{n}\|$. Si on s'impose $\|\vec{n}\| \in [0,5;2[$, les valeurs de la fonction $f(x)$ restent inférieures à 3,9641. On obtient donc la propriété suivante :

$\|\vec{n}\| \in [0,5;2[\quad \Rightarrow \quad \delta N < 4 \times \delta n$ et $|n_x| < 2$, $|n_y| < 2$ et $|n_z| < 2$.

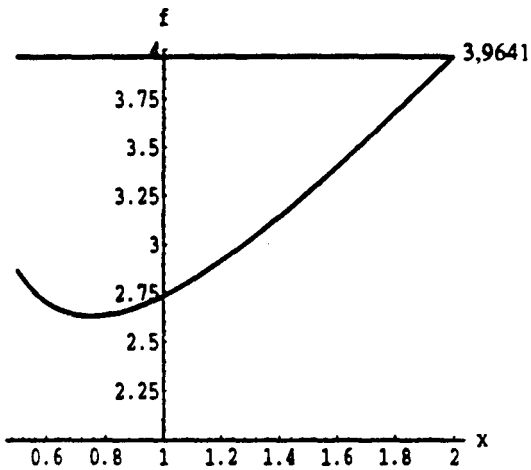


Figure 4•8 Représentation de la fonction $f(x)$.

• Comme on l'a vu sur l'exemple de l'ellipsoïde, (cf Figure 3•2) la norme du vecteur $\vec{N}$ peut varier dans un intervalle qui ne vérifie pas la contrainte que l'on réclame pour la normalisation. Pour remédier à ce problème, on se propose de scinder l'opération de normalisation en deux parties. La première partie transforme le vecteur $\vec{N}$ pour qu'il vérifie la propriété puis la seconde partie effectue la normalisation avec les critères que l'on vient de définir.

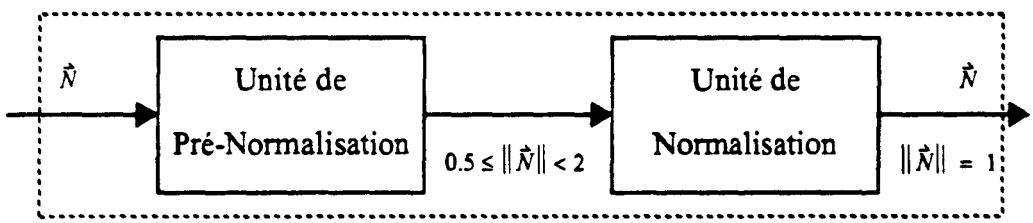


Figure 4•9 Diagramme de la normalisation.

L'unité de pré-normalisation devra être simple, de coût réduit et permettre de simplifier l'unité de normalisation qui restera toutefois assez complexe. Elle ne devra pas utiliser de divisions par des valeurs quelconques mais seulement des divisions par des puissances de 2. On n'utilisera ainsi que de simples décaleurs variables dont le coût est très modique.

4•2•2 Définition formelle.

• On définit la pré-normalisation comme suit :

Soit un vecteur $\vec{V}$ de composantes (V_x, V_y, V_z).
Soit (V_i, V_j, V_k) une permutation de (V_x, V_y, V_z) telle que $|V_i| \geq |V_j| \geq |V_k|$.
Soit $n \in Z$ tel que $|V_i| \times 2^n \in [0,5;1[$.

$$\|2^n\vec{V}\| = \sqrt{(2^n|V_i|)^2 + (2^n|V_j|)^2 + (2^n|V_k|)^2}$$

$$0{,}25 \le (2^n|V_i|)^2 < 1 \qquad 0 \le (2^n|V_j|)^2 < 1 \qquad 0 \le (2^n|V_k|)^2 < 1$$

$$0{,}25 \le (2^n|V_i|)^2 + (2^n|V_j|)^2 + (2^n|V_k|)^2 < 3 \Rightarrow \begin{cases} 0{,}25 \le \|2^n\vec{V}\|^2 < 3 \\ \Rightarrow \boxed{0{,}5 \le \|2^n\vec{V}\| < \sqrt{3}} \end{cases}$$

Soit $\vec{U} = 2^n\vec{V}$, l'image du vecteur $\vec{V}$ après l'opération de pré-normalisation. Cette opération que l'on vient d'introduire va donc consister à calculer la valeur de n et à effectuer la multiplication de chacune des composantes du vecteur par 2^n.

Le vecteur $\vec{U}$ vérifie la propriété suivante : $0.5 \le \|\vec{U}\| < \sqrt{3} < 2$.

- Le diagramme de la normalisation (cf Figure 4•9) est donc bien vérifié par l'unité de pré-normalisation telle que nous venons de la définir de façon formelle.

4•2•3 Transmission théorique de l'erreur dans l'unité de pré-normalisation.

Calculons l'influence exacte de cette unité sur la transmission de l'erreur lors de la normalisation.

Soit δn la précision pour le vecteur $\vec{N}$ avant normalisation.

Soit δN la précision pour le vecteur $\vec{N}$ après normalisation.

On a montré au paragraphe 4•1•1 que : $\delta N \le \left[\frac{1}{\|\vec{n}\|} + \sqrt{3}\|\vec{n}\|\right] \times \delta n$.

On vient aussi de montrer que, suite à la pré-normalisation : $0.5 \le \|\vec{n}\| < \sqrt{3}$

On en déduit donc : $\boxed{\delta N < 3{,}58 \times \delta n}$.

4•3 Les formats de données dans l'unité de normalisation.

Dans ce paragraphe, nous allons déterminer les formats de données qui doivent être utilisés au cours des calculs successifs dans l'unité de normalisation. En fixant ces formats nous allons introduire des erreurs au cours des calculs. Il va donc falloir estimer ces erreurs et les intégrer dans les formules théoriques, de transmission de l'erreur, établies au paragraphe 4•1.

4•3•1 Le choix de la précision d'entrée de l'unité de normalisation.

- Il faut déterminer quelle est la précision nécessaire sur les composantes du vecteur de la normale sortant de l'unité. Au paragraphe 4•1, différentes formules ont été établies concernant la précision requise pour les composantes du vecteur après la normalisation.

Soit $\delta n = Sup(\delta n_x, \delta n_y, \delta n_z)$ la précision pour le vecteur $\vec{n}$ (avant normalisation).

On a pour le diffus

$$\delta I_D \le \left[\frac{\sqrt{3}}{\|\vec{n}\|} + 3\|\vec{n}\|\right] \times \delta n < (1 + 3\sqrt{3}) \times \delta n \qquad (1)$$

$$\delta I_D \le 6.20 \times \delta n$$

et pour le spéculaire $$\delta I_S \le \left[\frac{\sqrt{24}}{\|\vec{n}\|} + 6\sqrt{2}\|\vec{n}\|\right] \times E \times \delta n < (2\sqrt{2} + 6\sqrt{6}) \times E \times \delta n \qquad (2)$$

$$\delta I_S \le 17{,}53 \times E \times \delta n$$

ou $$\delta I_S \le \left[\frac{\sqrt{3}}{\|\vec{n}\|} + 3\|\vec{n}\|\right] \times E' \times \delta n < (1 + 3\sqrt{3}) \times E' \times \delta n \qquad (3)$$

$$\delta I_S \le 6{,}20 \times E' \times \delta n$$

(selon la méthode choisie)

4•3•1•1 Les conditions imposées pour le calcul du diffus.

• Si on ne se préoccupe que de l'éclairement diffus, on peut aisément déterminer la précision requise pour les composantes du vecteur avant la normalisation. Le processeur vidéo utilise trois convertisseurs analogiques-digitaux 8 bits, pour le rouge, le vert et le bleu.

L'erreur souhaitable pour le diffus est donc $\delta I_D \le \frac{1}{256}$.

En utilisant la relation (1), on en déduit l'inégalité suivante :

$$\delta I_D < 6{,}20 \times \delta n \le \frac{1}{256} \quad \Rightarrow \quad \delta n \le \frac{1}{1587{,}2} \le \frac{1}{2048}.$$

Cela signifie qu'il ne faut, en sortie de l'unité de pré-normalisation, que 11 bits pour coder la valeur absolue de chacune des composantes du vecteur de la normale. (Il ne faut cependant pas oublier le bit de signe pour chaque composante).

4•3•1•2 Influence de la précision pour le calcul du spéculaire.

• Dans le cas de la projection orthogonale et des sources à l'infini, on a montré au paragraphe 2•1•2 qu'il était plus économique d'utiliser la méthode du vecteur surbrillance $\vec{H}$. En conservant la précision estimée, dans le cas de la seule lumière diffuse, on va pouvoir calculer l'erreur induite sur la lumière spéculaire suivant les valeurs prises par l'exposant spéculaire.

En utilisant la relation (3), on en déduit l'inégalité suivante :

$$\delta I_S < \frac{6{,}20 \times E'}{2048} \approx 0{,}003026 \times E'$$

• Cela permet de tracer la courbe théorique donnant la valeur maximale de l'erreur sur l'intensité spéculaire, en fonction de l'exposant spéculaire. On va ainsi déterminer le plus grand exposant spéculaire que l'on peut admettre pour une valeur maximale donnée de l'erreur.

• Par exemple, la figure suivante montre que l'exposant spéculaire doit être inférieur à 65 pour garantir une erreur inférieure à 20%.

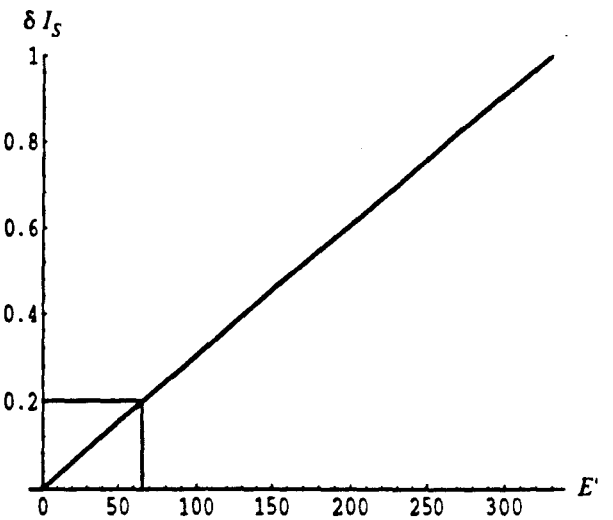


Figure 4•10 Evolution de la précision sur l'intensité spéculaire en fonction de l'exposant spéculaire.

4•3•1•3 Les conditions imposées pour le calcul du spéculaire.

• La précision sur le vecteur $\hat{N}$ avant normalisation a été calculée pour satisfaire une erreur minimum sur l'intensité de lumière diffuse. On va maintenant déterminer cette précision pour qu'elle satisfasse les exigences du calcul de la lumière spéculaire.

Les intensités de lumière spéculaire pour le rouge, le vert et le bleu sont obtenues par la formule suivante :

$$I_S = IK_S \times \left[\frac{\hat{n}}{\|\hat{N}\|} \cdot \hat{H}\right]^E$$

Ce calcul passe par l'évaluation de la fonction : $f : \begin{cases} \Re \to \Re \\ X \to IK_S \times X^E \end{cases}$.

Le chapitre 3 présente et compare différentes architectures permettant l'évaluation de cette fonction en vue du calcul de l'intensité de lumière spéculaire.

On a alors $I_S = f\left(\frac{\hat{n}}{\|\hat{N}\|} \cdot \hat{H}\right) = f(PS)$ avec $PS = \frac{\hat{n}}{\|\hat{N}\|} \cdot \hat{H}$.

• L'unité calculant la fonction $f(PS)$ opère avec des valeurs d'entrée dont la précision est fixée par l'architecture choisie. On ne va donc pas calculer la précision sur le vecteur $\hat{N}$ avant normalisation en fonction de la dynamique de l'exposant spéculaire, mais la calculer à partir de la précision demandée pour la valeur du produit scalaire :

$$\hat{N} \cdot \hat{H} = \frac{\hat{n}}{\|\hat{N}\|} \cdot \hat{H} = PS. \text{ Soit } \delta n = Sup(\delta n_x, \delta n_y, \delta n_z).$$

Au paragraphe 4•1•2•2, on a montré $\boxed{\delta PS \leq (1 + 3 \times \sqrt{3}) \times \delta n < 6{,}2 \times \delta n}$.

Soit $\pi(\hat{n})$ la taille du mot binaire utilisé pour coder $\hat{N}$ avant normalisation.

Soit $\pi(PS)$ la taille du mot binaire utilisé pour coder le résultat du produit scalaire $\hat{N} \cdot \hat{H}$.

Si on code sur m bits, la partie décimale des composantes du vecteur $\hat{N}$ avant la normalisation, chaque codage n_i d'une composante correspond à l'ensemble des valeurs $[(n_i - \delta n), (n_i + \delta n)]$ avec : $\delta n = \frac{0{,}5}{2^m}$. $\Rightarrow \delta PS < \frac{3{,}1}{2^m} = \frac{0{,}38}{2^{(m-3)}}$.

Si on code sur l bits, la partie décimale du produit scalaire, on y ajoute une erreur qui est au maximum égale à $\frac{0{,}5}{2^l}$. Donc on a toutes erreurs cumulées :

$$\delta PS < \frac{0{,}38}{2^{(m-3)}} + \frac{0{,}5}{2^l}.$$

	Valeur de l'erreur	Amélioration
$l = m - 1$	$\delta PS < \frac{2{,}05}{2^l}$	
		$\frac{0{,}772}{2^l}$
$l = m - 2$	$\delta PS < \frac{1{,}278}{2^l}$	
		$\frac{0{,}398}{2^l}$
$l = m - 3$	$\delta PS < \frac{0{,}88}{2^l}$	
		$\frac{0{,}19}{2^l}$
$l = m - 4$	$\delta PS < \frac{0{,}69}{2^l}$	
		$\frac{0{,}09}{2^l}$
$l = m - 5$	$\delta PS < \frac{0{,}60}{2^l}$	

Figure 4•11 Influence de la dynamique du codage de PS et de $\hat{N}$ avant normalisation sur l'erreur.

• Au paragraphe 3•3•2•1, nous avons convenu qu'une transformation du produit scalaire serait opérée afin que le résultat soit nul s'il devait être négatif et ne soit jamais égal à 1. Ainsi, le codage du produit scalaire ne fait intervenir ni signe ni valeur entière. On en déduit qu'il faut 4 bits de plus pour coder les composantes du vecteur $\hat{N}$ avant normalisation que pour le codage du résultat du produit scalaire $\hat{N} \cdot \hat{H}$.

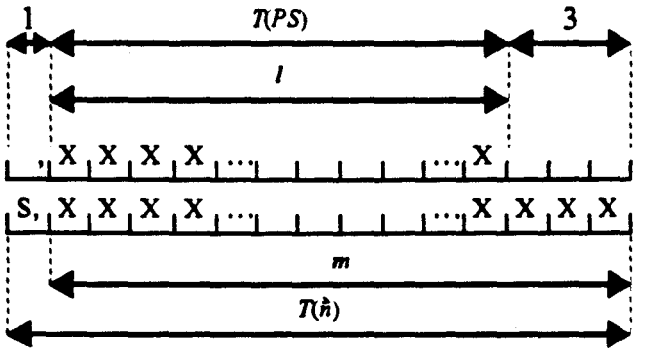


Figure 4•12 Correspondance entre le codage du vecteur $\hat{N}$ de la normale et le codage du produit scalaire $\hat{N} \cdot \hat{H}$.

• Dans le cas de l'architecture présentée au paragraphe 3•3•2, le résultat du produit scalaire $\hat{N} \cdot \hat{H}$ est codé sur 10 bits. Il est donc, dans ce cas, parfaitement inutile que la sortie de l'unité de pré-normalisation fournisse les valeurs absolues des composantes du vecteur $\hat{N}$ sur plus de 13 bits.

• Au paragraphe 3•3•3, l'Unité "d'Arrangement des Bits" a été introduite afin de faire reculer les limites induites par l'architecture précédente. De même, on détermine la taille des mots binaires sortant de l'unité de pré-normalisation à partir de celle des mots binaires entrant dans l'U.A.B.

Le tableau ci-dessous donne la taille du codage des valeurs absolues des composantes du vecteur $\hat{N}$ avant normalisation suivant la configuration de l'U.A.B. utilisée.

exposant / mantisse	0 bit↓	1 bit↓	2 bits↓	3 bits↓	4 bits↓
10 bits→	13 bits	14 bits	16 bits		
9 bits→	12 bits	13 bits	15 bits	19 bits	
8 bits→	11 bits	12 bits	14 bits	18 bits	26 bits
7 bits→		11 bits	13 bits	17 bits	25 bits
6 bits→			12 bits	16 bits	24 bits

Figure 4•13 Evolution de la précision de l'unité de pré-normalisation en fonction de la topologie de l'unité d'arrangement des bits.

• Parmi les diverses configurations, au paragraphe 3•3•3•3, notre choix s'était porté sur 7 bits de mantisse et 3 bits d'exposant. Pour cette architecture, il ne serait pas utile de coder sur plus de 17 bits les valeurs absolues des composantes du vecteur $\hat{N}$ avant normalisation. Au- delà, la qualité de l'image ne serait plus améliorée.

4•3•2 Les simplifications induites par la pré-normalisation.

• Au paragraphe 4•2•3, on a établi une relation entre l'erreur présente en entrée de l'unité de normalisation et l'erreur présente en sortie de la même unité. Cette relation peut être appliquée grâce à l'introduction de l'unité de pré-normalisation devant l'unité de normalisation.

On a : $\delta N < 3{,}58 \times \delta n$.

Si on code sur m bits la partie décimale des composantes du vecteur $\hat{N}$ avant la normalisation, chaque codage n_i d'une composante correspond à l'ensemble des valeurs $](n_i - \delta n), (n_i + \delta n)[$ avec : $\delta n = \frac{0{,}5}{2^m}$. $\delta N < \frac{1{,}79}{2^m} = \frac{0{,}45}{2^{(m-2)}}$.

Si on code sur l bits, la partie décimale des composantes du vecteur $\hat{N}$ après la normalisation, on ajoute une erreur qui est au maximum égale à $\frac{0{,}5}{2^l}$. Donc on a, toutes erreurs cumulées :

$\delta N < \frac{0{,}45}{2^{(m-2)}} + \frac{0{,}5}{2^l}$.

• Pour choisir entre les différentes valeurs proposées pour l, il faut déterminer quelle est l'erreur qui sera répercutée sur le produit scalaire, c'est l'opération suivante dans le calcul de l'éclairement. Au paragraphe 4•1•2•1, on a montré que $\delta PS \leq \sqrt{3} \times \delta N$.

	Valeur de l'erreur		
	après la normalisation		après le produit scalaire
$l = m - 2$	$\delta N < \frac{0{,}95}{2^{(m-2)}}$	$\delta N < \frac{0{,}95}{2^l}$	$\delta PS < \frac{0{,}823}{2^{(m-3)}}$
$l = m - 1$	$\delta N < \frac{0{,}70}{2^{(m-2)}}$	$\delta N < \frac{1{,}40}{2^l}$	$\delta PS < \frac{0{,}605}{2^{(m-3)}}$
$l = m$	$\delta N < \frac{0{,}58}{2^{(m-2)}}$	$\delta N < \frac{2{,}29}{2^l}$	$\delta PS < \frac{0{,}496}{2^{(m-3)}}$
$l = m + 1$	$\delta N < \frac{0{,}51}{2^{(m-2)}}$	$\delta N < \frac{4{,}08}{2^l}$	$\delta PS < \frac{0{,}442}{2^{(m-3)}}$
$l = m + 2$	$\delta N < \frac{0{,}48}{2^{(m-2)}}$	$\delta N < \frac{7{,}66}{2^l}$	$\delta PS < \frac{0{,}415}{2^{(m-3)}}$

Figure 4•14 Influence de la précision en entrée et en sortie sur l'erreur.

• En choisissant $l = m$, on obtient une erreur théorique, après le calcul du produit scalaire, qui est de l'ordre de la valeur voulue. Mais cela suppose que la sortie du produit scalaire n'est pas perturbée par une erreur supplémentaire due à la limitation du nombre de bits utilisés pour coder son résultat. Si une telle erreur devait apparaître, on obtiendrait : $\delta PS < \frac{0{,}996}{2^{(m-3)}}$.

4•3•3 Etude théorique des formats de l'unité de normalisation de base.

• La figure suivante reprend le diagramme de base de l'unité de normalisation présenté au paragraphe 3•2•3•1. Au paragraphe 3•2•3, il n'avait pas été possible de choisir entre les différentes solutions. C'est la taille respective des unités utilisées dans chaque solution qui constitue le critère de choix approprié. Il faut donc calculer pour chaque solution la taille de tous les mots entrant et sortant des unités afin de pouvoir déterminer la taille de celles-ci.

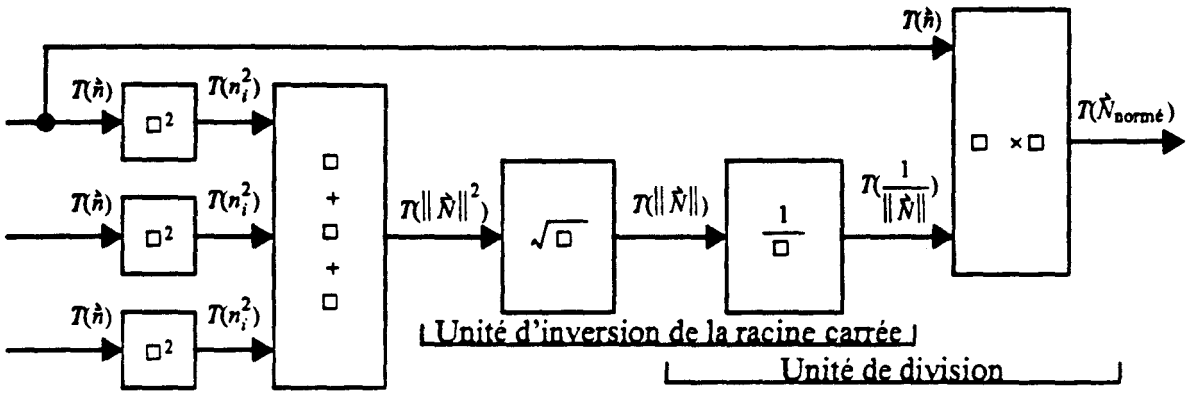


Figure 4•15 Diagramme de la normalisation d'une composante du vecteur de la normale.

Soit $\mathcal{T}(\hat{n})$ la taille du mot binaire utilisé pour coder $\hat{N}$ avant normalisation.

Soit $\mathcal{T}(\hat{N}_{\text{normé}})$ la taille du mot binaire utilisé pour coder $\hat{N}$ après normalisation.

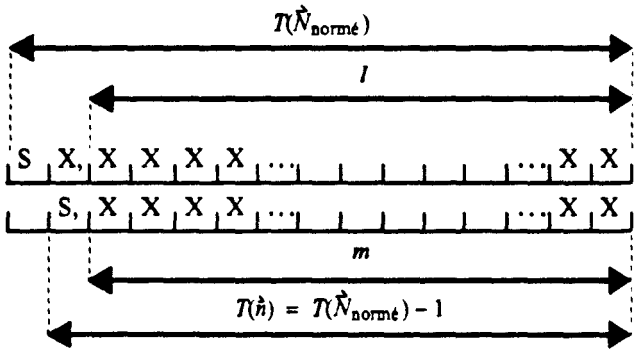


Figure 4•16 Codage des composantes du vecteur $\hat{N}$ en entrée et en sortie de l'unité de normalisation.

il faut déterminer :

→ $\mathcal{T}(\frac{1}{\|\hat{N}\|})$ la taille des mots binaires que l'on doit utiliser pour coder $\frac{1}{\|\hat{N}\|}$.

→ $\mathcal{T}(\|\hat{N}\|)$ la taille des mots binaires que l'on doit utiliser pour coder $\|\hat{N}\|$.

→ $\mathcal{T}(\|\hat{N}\|^2)$ la taille des mots binaires que l'on doit utiliser pour coder $\|\hat{N}\|^2$.

→ $\mathcal{T}(n_i^2)$ la taille des mots binaires que l'on doit utiliser pour coder n_x^2, n_y^2 et n_z^2.

4•3•3•1 Etude de la taille de la partie entière des données.

• Examinons d'abord la taille de la partie entière des données circulant d'un opérateur à l'autre, au sein de l'unité de normalisation.

Comme $\|\hat{N}\| \in [0,5;1,73[$, il ne faut aucun bit de signe et seulement 1 bit pour coder la partie entière de la norme du vecteur $\hat{N}$.

Comme $\|\hat{N}\| \in [0,5;1,73[$, on a $1/\|\hat{N}\| \in]0,57;2]$ il ne faut aucun bit de signe et seulement 2 bits pour coder la partie entière de l'inverse de la norme du vecteur $\hat{N}$.

Comme $\|\hat{N}\|^2 \in [0,25;3[$, il ne faut aucun bit de signe et seulement 2 bits pour coder la partie entière du carré de la norme du vecteur $\hat{N}$.

Comme $n_i \in [0;1[$, on a $n_i^2 \in [0;1[$. Il ne sera pas nécessaire de coder la partie entière du carré de chaque composante du vecteur $\hat{N}$. Il ne faudra pas non plus de bit de signe.

4•3•3•2 Etude de la taille de la partie décimale des données.

• Pour compléter ce travail et connaître la largeur des chemins de données passant d'un opérateur à un autre, il ne reste plus qu'à déterminer la taille de la partie décimale des mots binaires.

Pour cela, nous allons étudier la transmission de l'erreur d'opération en opération au sein de l'unité de normalisation.

-a-

Soit δn_i^2 la contribution, à l'erreur sur le carré d'une composante, de l'erreur en entrée δn.

$(n_i^2) = (n_i)^2 \Rightarrow \partial n_i^2 = 2 \times n_i \times \partial n_i \Rightarrow \delta n_i^2 \leq 2 \times \delta n$. (car $|n_i| \leq 1$)

Pour calculer l'erreur sur le carré de la norme, il faut prendre en compte la contribution de l'erreur en entrée mais aussi celle de l'erreur qui est ajoutée à cause de la restriction de dynamique induite par les opérateurs d'élévation au carré qui précédent.

Si on code la partie décimale du résultat de l'élévation au carré sur $m1$ bits, on introduit en sortie de chaque opération une erreur $\varepsilon 1$ qui est au maximum égale à $\frac{0,5}{2^{m1}}$.

-b-

Soit δn^2 la contribution, à l'erreur sur le carré de la norme, de l'erreur en entrée et de l'erreur totale ε introduite par les étages précédents.

$(n^2) = n_x^2 + n_y^2 + n_z^2 + \varepsilon \Rightarrow \partial n^2 = 2 \times n_x \times \partial n_x + 2 \times n_y \times \partial n_y + 2 \times n_z \times \partial n_z + \varepsilon \Rightarrow \delta n^2 \leq 6 \times \delta n + \varepsilon.$

Dans le pire des cas $\varepsilon = \frac{1,5}{2^{m1}}$. Donc $\delta n^2 \leq 6 \times \delta n + \frac{1,5}{2^{m1}}$.

Si on code le carré de la norme sur $m1$ bits, (ou plus, mais cela ne présente aucun intérêt) on n'introduit pas d'erreur supplémentaire. Par contre, si on désire coder le carré de la norme sur $m2$ bits (avec $m2 < m1$), on introduit une erreur supplémentaire qui sera au maximum égale à $0,5/2^{m2}$. Ainsi l'erreur totale introduite, du fait du codage successif des résultats, sera bornée par la valeur :

$$\left[\frac{1,5}{2^{m1}} + \frac{0,5}{2^{m2}}\right].$$

-c-

Soit $\delta\|\hat{N}\|$ la contribution, à l'erreur sur la norme, de l'erreur en entrée et de l'erreur totale ε introduite par les étages précédents.

$$\|\hat{N}\| = \sqrt{n_x^2+n_y^2+n_z^2+\varepsilon} \quad \Rightarrow \quad \partial\|\hat{N}\| = \frac{n_x}{\|\hat{N}\|}\times\partial n_x + \frac{n_y}{\|\hat{N}\|}\times\partial n_y + \frac{n_z}{\|\hat{N}\|}\times\partial n_z + \frac{\partial\varepsilon}{2\|\hat{N}\|}$$

$$\Rightarrow \quad \delta\|\hat{N}\| \le \left[\frac{n_x}{\|\hat{N}\|}+\frac{n_y}{\|\hat{N}\|}+\frac{n_z}{\|\hat{N}\|}\right]\times Sup(\delta n_x, \delta n_y, \delta n_z) + \delta\varepsilon$$

Posons $a = \frac{n_x}{\|\hat{N}\|}$, $b = \frac{n_y}{\|\hat{N}\|}$ et $c = \frac{n_z}{\|\hat{N}\|}$ alors $a^2+b^2+c^2 = 1$ et d'après la démonstration faite au paragraphe 4•1•1, on a $Sup(a+b+c) = \sqrt{3}$. D'autre part $\frac{1}{\sqrt{3}} \le \frac{1}{\|\hat{N}\|} \le 2$ et $\varepsilon \approx 0 \pm \delta\varepsilon$.

$$\Rightarrow \quad \delta\|\hat{N}\| \le \sqrt{3}\times\delta n + \varepsilon.$$

Selon les choix adoptés pour l'étage précédent, on aura :

$$\delta\|\hat{N}\| \le \sqrt{3}\times\delta n + \frac{1,5}{2^{m1}} \text{ ou } \delta\|\hat{N}\| \le \sqrt{3}\times\delta n + \left[\frac{1,5}{2^{m1}}+\frac{0,5}{2^{m2}}\right].$$

Si on code la norme sur $m3$ bits, on introduit une erreur supplémentaire qui est au maximum égale à $\frac{0,5}{2^{m3}}$.

-d-

Pour obtenir la valeur inverse de la norme, deux solutions s'offrent à nous : soit on utilise une unité d'extraction de la racine carrée puis une unité d'inversion séparée, soit on utilise une seule et même unité. Ce dernier cas peut être considéré semblable au cas avec deux unités sans qu'il y ait introduction d'erreur intermédiaire au cours du calcul.

Soit $\delta(1/\|\hat{N}\|)$ la contribution, à l'erreur sur l'inverse de la norme. L'erreur en entrée de l'unité de normalisation, l'erreur totale ε introduite en entrée de l'opérateur racine carrée et ε' l'erreur éventuelle introduite en sortie de l'opérateur racine carrée y participent.

$$\frac{1}{\|\hat{N}\|} = \frac{1}{\sqrt{n_x^2+n_y^2+n_z^2+\varepsilon}+\varepsilon'} \quad \Rightarrow$$

$$\partial\frac{1}{\|\hat{N}\|} = -\frac{2n_x\times\partial n_x + 2n_y\times\partial n_y + 2n_z\times\partial n_z + \partial\varepsilon + 2\sqrt{n_x^2+n_y^2+n_z^2+\varepsilon}\times\partial\varepsilon'}{2\sqrt{n_x^2+n_y^2+n_z^2+\varepsilon}\times\left[\sqrt{n_x^2+n_y^2+n_z^2+\varepsilon}+\varepsilon'\right]^2}$$

Comme $\varepsilon \approx 0 \pm \delta\varepsilon$ et $\varepsilon' \approx 0 \pm \delta\varepsilon'$ on peut effectuer les simplifications suivantes :

$$\partial\frac{1}{\|\hat{N}\|} = -\left[\frac{\frac{n_x}{\|\hat{N}\|}\times\partial n_x + \frac{n_y}{\|\hat{N}\|}\times\partial n_y + \frac{n_z}{\|\hat{N}\|}\times\partial n_z}{\|\hat{N}\|^2} + \frac{\partial\varepsilon}{2\times\|\hat{N}\|^3} + \frac{\partial\varepsilon'}{\|\hat{N}\|^2}\right]$$

$$\Rightarrow \quad \delta\frac{1}{\|\hat{N}\|} \le \frac{\frac{|n_x|}{\|\hat{N}\|}\times\delta n_x + \frac{|n_y|}{\|\hat{N}\|}\times\delta n_y + \frac{|n_z|}{\|\hat{N}\|}\times\delta n_z}{\|\hat{N}\|^2} + \frac{\delta\varepsilon}{2\times\|\hat{N}\|^3} + \frac{\delta\varepsilon'}{\|\hat{N}\|^2}$$

$$\Rightarrow \quad \delta\frac{1}{\|\hat{N}\|} \le \frac{\sqrt{3}\times\delta n + \delta\varepsilon'}{\|\hat{N}\|^2} + \frac{\delta\varepsilon}{2\times\|\hat{N}\|^3}$$

or $\frac{1}{3} \le \frac{1}{\|\hat{N}\|^2} \le 4$ et $\frac{1}{6\sqrt{3}} \le \frac{1}{2\|\hat{N}\|^3} \le 4$ donc $\delta\frac{1}{\|\hat{N}\|} \le 4\sqrt{3}\times\delta n + 4\times\varepsilon' + 4\times\varepsilon$

Dans le cas d'une seule unité, l'erreur est : $\delta \frac{1}{\|\hat{N}\|} \leq 6{,}93 \times \delta n + \frac{6{,}0}{2^{m1}}$ ou $\delta \frac{1}{\|\hat{N}\|} \leq 6{,}93 \times \delta n + \frac{6{,}0}{2^{m1}} + \frac{2{,}0}{2^{m2}}$.

Dans le cas de deux unités séparées, l'erreur est :

$$\delta \frac{1}{\|\hat{N}\|} \leq 6{,}93 \times \delta n + \frac{6{,}0}{2^{m1}} + \frac{2{,}0}{2^{m3}} \text{ ou } \delta \frac{1}{\|\hat{N}\|} \leq 6{,}93 \times \delta n + \frac{6{,}0}{2^{m1}} + \frac{2{,}0}{2^{m2}} + \frac{2{,}0}{2^{m3}}.$$

Si on code l'inverse de la norme sur $m4$ bits, on introduit une erreur supplémentaire qui est au maximum égale à $\frac{0{,}5}{2^{m4}}$.

-e-

Il ne reste plus que la dernière étape avant le résultat complet. Il faut étudier l'erreur finale pour les différentes architectures.

Soit δN la contribution, de l'erreur en entrée de l'unité de normalisation, de l'erreur totale ε introduite par tous les modules précédents la multiplication.

$N_x = \left(\frac{1}{\|\hat{N}\|} + \varepsilon\right) \times n_x = \frac{n_x}{\|\hat{N}\|} + n_x \times \varepsilon$ Du fait de la propriété de distribution de la multiplication par rapport à l'addition, on peut séparer en deux le calcul de l'erreur :

$$\delta N < 3{,}58 \times \delta n + \varepsilon$$

Il ne reste plus qu'à introduire l'erreur produite par la restriction du nombre de bits sur le résultat de la multiplication. Soit m le nombre de bits servant à coder la partie décimale des composantes du vecteur $\hat{N}$ avant ou après normalisation. On obtient finalement la formule suivante :

$$\delta N < \frac{1{,}79}{2^m} + \frac{6{,}0}{2^{m1}} + \frac{2{,}0}{2^{m2}} + \frac{2{,}0}{2^{m3}} + \frac{0{,}5}{2^{m4}} + \frac{0{,}5}{2^m} = \frac{2{,}29}{2^m} + \frac{6{,}0}{2^{m1}} + \frac{2{,}0}{2^{m2}} + \frac{2{,}0}{2^{m3}} + \frac{0{,}5}{2^{m4}}$$

Reste à déterminer les différentes valeurs de $m1$, $m2$, $m3$ et $m4$ en fonction de la valeur de m.

4•3•3•3 Applications aux différentes architectures

-a-

Premièrement, étudions le cas, où l'on utilise un maximum d'unités disjointes dont la multiplication. A l'examen du tableau présenté en annexe Figure D•1, les conclusions ne sont pas très intéressantes. Il faut de larges chemins de données entre les blocs constituant l'unité de normalisation. Si nous élargissions volontairement d'un bit la taille de la partie décimale des composantes du vecteur avant normalisation, la formule deviendrait :

$\delta N < \frac{1{,}40}{2^m} + \frac{6{,}0}{2^{m1}} + \frac{2{,}0}{2^{m2}} + \frac{2{,}0}{2^{m3}} + \frac{0{,}5}{2^{m4}}$. Dans ce cas on peut apprécier les résultats en consultant le tableau présenté en annexe Figure D•2. La Figure D•3 présente une configuration optimale parmi les solutions proposées dans le tableau précédemment cité.

-b-

Deuxièmement, étudions le cas, où l'on utilise une unité intégrée d'inversion et de calcul de la racine carrée. On obtient finalement la formule suivante :

$$\delta N < \frac{2{,}29}{2^m} + \frac{6{,}0}{2^{m1}} + \frac{2{,}0}{2^{m2}} + \frac{0{,}5}{2^{m4}}$$

A l'examen du tableau présenté en annexe Figure D•4, les conclusions ne sont pas beaucoup plus intéressantes que celles relatives à l'architecture en modules séparés. Il faut de larges chemins de données entre les blocs constituant l'unité de normalisation. Si nous élargissions volontairement d'un bit la taille de la partie décimale des composantes du vecteur avant normalisation, la formule deviendrait :

$\delta N < \frac{1,40}{2^m} + \frac{6,0}{2^{m1}} + \frac{2,0}{2^{m2}} + \frac{0,5}{2^{m4}}$. Dans ce cas on peut apprécier les résultats en consultant le tableau présenté en annexe Figure D•5. La Figure D•6 présente une configuration optimale parmi les solutions proposées dans le tableau précédemment cité.

-c-

Troisièmement, étudions le cas, où l'on utilise des unités de division. On obtient finalement la formule suivante :

$$\delta N < \frac{2,29}{2^m} + \frac{6,0}{2^{m1}} + \frac{2,0}{2^{m2}} + \frac{2,0}{2^{m3}}$$

A l'examen du tableau présenté en annexe Figure D•7, les conclusions ne sont pas beaucoup plus intéressantes que celles relatives aux deux études précédentes. Il faut de larges chemins de données entre les blocs constituant l'unité de normalisation. Si nous élargissions volontairement d'un bit la taille de la partie décimale des composantes du vecteur avant normalisation, la formule deviendrait :

$\delta N < \frac{1,40}{2^m} + \frac{6,0}{2^{m1}} + \frac{2,0}{2^{m2}} + \frac{2,0}{2^{m3}}$. Dans ce cas on peut apprécier les résultats en consultant le tableau présenté en annexe Figure D•8. La Figure D•9 présente une configuration optimum parmi les solutions proposées dans le tableau précédemment cité.

4•3•3•4 Conclusion.

• Quelle que soit l'architecture choisie, il est intéressant d'augmenter la taille de la sortie de l'unité de pré-normalisation, cela permet d'avoir une plus grande marge d'erreur et de faciliter ainsi la fabrication de l'unité de normalisation.

Comme on peut le remarquer au regard des divers tableaux précédents, les solutions sont multiples. On peut toutefois constater que les architectures utilisant une unité de division ou une unité intégrée, d'extraction de la racine carrée et d'inversion, offrent des possibilités plus avantageuses.

La figure ci-dessous présente l'architecture optimale choisie pour chacune des trois solutions. (Sur la figure, la valeur placée au dessus de chaque flèche indique la taille totale du chemin de données. Le triplet en dessous de chaque flèche indique la présence du signe, la taille de la partie entière et enfin la taille de la partie décimale.)

• En examinant les trois solutions, les remarques suivantes peuvent être faites : Premièrement, il ne semble pas exister de composants commerciaux discrets pouvant s'intégrer dans un pipeline et effectuant des divisions. Deuxièmement, si un choix doit être fait entre les deux solutions utilisant des unités de multiplication, l'utilisation d'une unité intégrant le calcul de la racine carrée et le calcul de l'inversion s'avère être la solution la plus intéressante.Tant que la taille du chemin de données entrant dans l'unité d'inversion de la racine carrée est petite, l'unité peut être réalisée au moyen d'une mémoire. Cette solution pourra uniquement être appliquée lorsque la précision requise est faible.

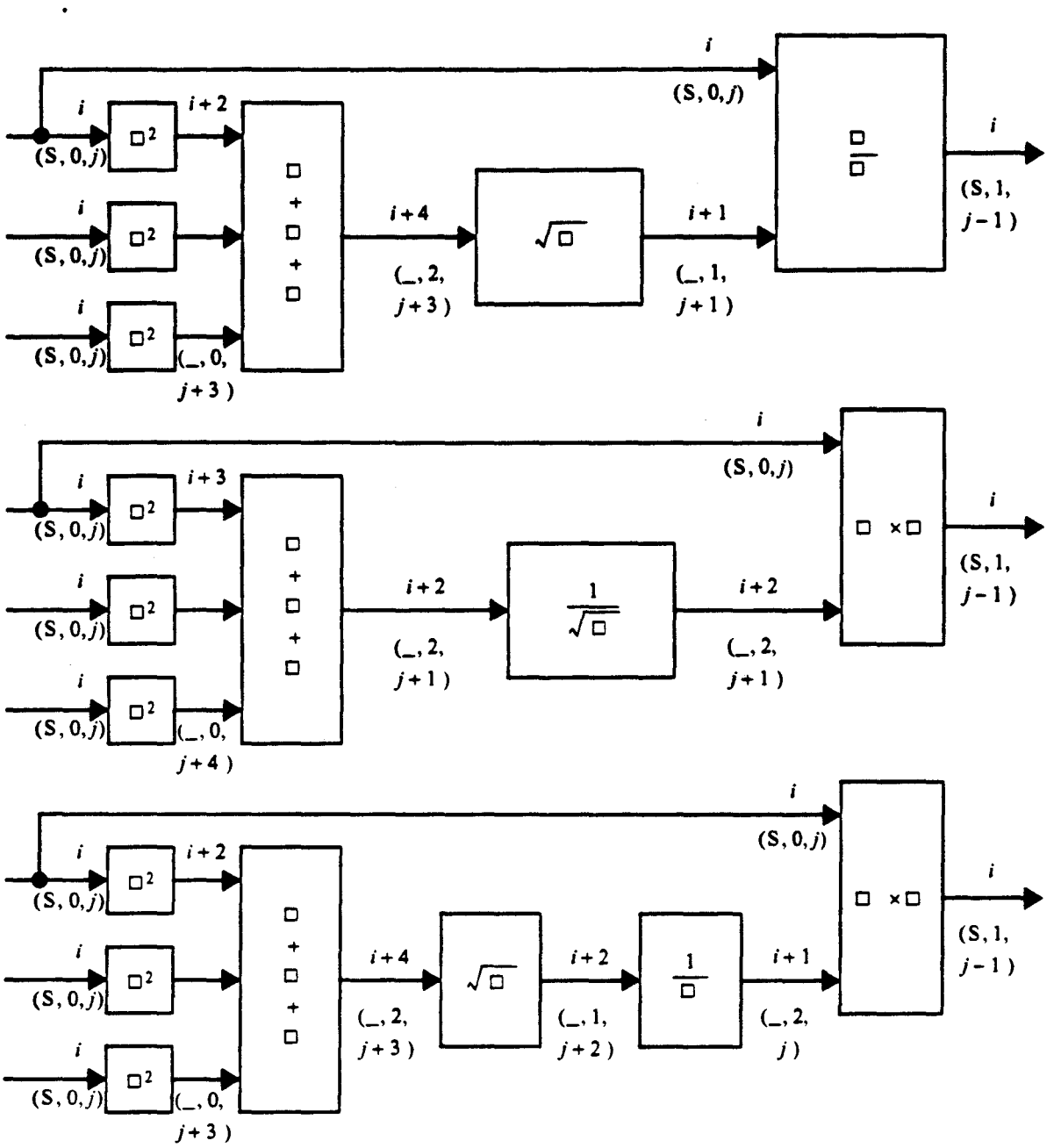


Figure 4•17 Les architectures pour l'une unité de normalisation.

4•3•4 Etude théorique des formats de l'unité de normalisation en S./L.N.S.

• Au paragraphe 3•2•3•2, un autre type d'architecture a été proposé. Il s'agit d'utiliser au cours de certains calculs (les multiplications entre autres) la notation S./L.N.S. Toute l'étude théorique des formats de données, effectuée dans le paragraphe 4•3•3 pour la notation en binaire naturel, doit être reprise en tenant compte des modifications de la nouvelle architecture.

4•3•4•1 Etude de la taille de la partie entière des données.

• Tout d'abord, occupons nous de la question la plus facile : la taille de la partie entière des données circulant au format S./L.N.S., au sein de l'unité de normalisation. Commençons par les composantes du vecteur $\hat{N}$ avant normalisation.

$0 \le |n_i| < 1 \Rightarrow -\infty \le Log_2(|n_i|) < 0$. Il n'est donc pas nécessaire de conserver en S./L.N.S le signe du logarithme, la valeur absolue suffira. On a : $0 < -Log_2(|n_i|) \le \infty$. Mais une question se pose : "Comment coder ∞?"

La plus petite valeur non nulle, qui peut être prise par N_i, est $\frac{1}{4096}$. Or $-Log_2(\frac{1}{4096}) = 12$. Le nombre 12 se code sur 4 bits en binaire naturel non signé. Il ne faut donc que 4 bits pour la partie entière du codage S./L.N.S. des composantes non nulles du vecteur $\hat{N}$. Pour coder les composantes nulles, il suffit de choisir une partie entière supérieure à 12 et inférieure ou égale à 15. Ainsi, on s'attribue un codage à ∞ sans devoir accroître la taille des nombres en S./L.N.S.

Il ne faudra donc que 4 bits pour mémoriser toute la partie non décimale du codage S./ L.N.S. Etudions maintenant les composantes du vecteur $\hat{N}$ après normalisation.

$0 \le |N_i| \le 1 \Rightarrow -\infty \le Log_2(|N_i|) \le 0$ Il n'est donc pas nécessaire de conserver en S./L.N.S le signe du logarithme, la valeur absolue suffira. On a : $0 \le -Log_2(|N_i|) \le \infty$. Mais la question du codage de ∞ se pose une fois encore.

La plus petite valeur non nulle, qui peut être prise par N_i, est $\frac{1}{2048}$. Or $-Log_2(\frac{1}{2048}) = 11$. Le nombre 11 se code sur 4 bits en binaire naturel non signé. Il ne faut donc que 4 bits pour la partie entière du codage S./L.N.S. des composantes non nulles du vecteur $\hat{N}$. Pour les coder, il suffit de choisir une partie entière supérieure à 11 et inférieure ou égale à 15, ainsi on s'attribue un codage de ∞ sans devoir accroître la taille des nombres en S./L.N.S.

Il ne faudra donc que 4 bits pour mémoriser toute la partie non décimale du codage S./ L.N.S des composantes du vecteur $\hat{N}$ après normalisation.

Enfin, l'extraction de la racine carrée, l'inversion et la multiplication peuvent toutes trois être effectuées en S./L.N.S. Etudions donc le codage S./L.N.S. du carré de la norme.

$\frac{1}{4} < \|\hat{N}\|^2 \le 3 \Rightarrow Log_2(\frac{1}{4}) < Log_2(\|\hat{N}\|^2) \le Log_2(3) \Rightarrow -2 < Log_2(\frac{1}{\|\hat{N}\|}) \le 1{,}59$ On est ainsi obligé de conserver le signe du logarithme. Pour harmoniser le codage, on procédera, malgré tout, comme précédemment : $-1{,}59 \le -Log_2(\frac{1}{\|\hat{N}\|}) < 2$. La question du codage de ∞ ne se pose pas.

Il ne faudra donc que 3 bits pour mémoriser toute la partie non décimale du codage S./ L.N.S du carré de la norme.

4•3•4•2 Etude de la taille de la partie décimale des données.

• Pour compléter ce travail et connaître la largeur des chemins de données passant d'un opérateur à l'autre, il ne reste plus qu'à déterminer la taille de la partie décimale des mots binaires. Pour cela, déterminons la transmission de l'erreur concernant les données qui circulent au format S./L.N.S.

-a-

Soit $A = Log_2(n_x^2 + n_y^2 + n_z^2 + \varepsilon) = ln(n_x^2 + n_y^2 + n_z^2 + \varepsilon)/ln(2)$. Soit δA la contribution, à l'erreur sur l'inverse de la norme codée en S./L.N.S., de l'erreur en entrée et de l'erreur totale ε introduite par les étages précédents.

$\varepsilon = 0 + \partial\varepsilon$ est borné (en valeur absolue) par $\left[\frac{1,5}{2^{m1}}\right]$ ou par $\left[\frac{1,5}{2^{m1}} + \frac{0,5}{2^{m2}}\right]$ quand $m1 > m2$.

$$\Rightarrow \quad \partial A = \frac{2n_x \times \partial n_x + 2n_y \times \partial n_y + 2n_z \times \partial n_z + \partial\varepsilon}{ln(2) \times [n_x^2 + n_y^2 + n_z^2 + \varepsilon]}$$

$$\Rightarrow \quad \delta A \leq \frac{2\left[\frac{n_x}{\|\hat{N}\|} + \frac{n_y}{\|\hat{N}\|} + \frac{n_z}{\|\hat{N}\|}\right] \times Sup(\delta n_x, \delta n_y, \delta n_z)}{ln(2) \times \|\hat{N}\|} + \frac{\delta\varepsilon}{ln(2) \times \|\hat{N}\|^2}$$

Posons $a = \frac{n_x}{\|\hat{N}\|}$, $b = \frac{n_y}{\|\hat{N}\|}$ et $c = \frac{n_z}{\|\hat{N}\|}$ alors $a^2 + b^2 + c^2 = 1$ et d'après la démonstration faite au paragraphe 4•1•1, on a $Sup(a+b+c) = \sqrt{3}$. D'autre part $\frac{1}{\sqrt{3}} \leq \frac{1}{\|\hat{N}\|} \leq 2$.

$$\Rightarrow \quad \delta A \leq \frac{4\sqrt{3}}{ln(2)} \times \delta n + \frac{4}{ln(2) \times 2} \times \varepsilon.$$

Selon les choix adoptés pour l'étage précédent, on aura :

$$\delta A \leq \frac{5}{2^m} + \frac{8,66}{2^{m1}} \text{ ou } \delta A \leq \frac{5}{2^m} + \left[\frac{8,66}{2^{m1}} + \frac{2,89}{2^{m2}}\right].$$

Si on code la norme sur $m5$ bits, on introduit une erreur supplémentaire qui est au maximum égale à $\frac{0,5}{2^{m5}}$.

-b-

- Soit $B = Log_2(n_i) = ln(n_i)/ln(2)$. Soit δB la contribution de l'erreur en entrée, à l'erreur sur une composante du vecteur $\hat{N}$ codée en S./L.N.S.

$$\Rightarrow \quad \partial B = \frac{\partial n_i}{ln(2) \times n_i}. \text{ Or } 0 \leq n_i < 1 \quad \Rightarrow \quad 1 < \frac{1}{n_i} \leq \infty \quad \Rightarrow \quad \frac{\partial n_i}{ln(2)} < \partial B \leq \infty.$$

Cette erreur ne présente aucun intérêt car elle peut devenir très grande pour de petites valeurs de n_i. Dans ce cas B est aussi très grand et lors de la conversion du codage S./L.N.S. en binaire naturel bien plus que l'erreur absolue c'est l'erreur relative qui va intervenir, pour calculer l'erreur finale.

Si on code la valeur S./L.N.S. sur $m6$ bits, on introduit une erreur supplémentaire qui est au maximum égale à $\frac{0,5}{2^{m6}}$.

- Voyons maintenant la transmission des erreurs sur la valeur du produit scalaire.

Soit $A = ln(n_x^2 + n_y^2 + n_z^2 + \varepsilon)/ln(2)$, $B = ln(n_i)/ln(2)$, $C = \frac{A}{2} - B + \varepsilon'$ et $D = exp(-C \times ln(2))$.

ε' est majoré en valeur absolue par $\frac{0,25}{2^{m5}} + \frac{0,5}{2^{m6}}$.

-c-

Soit δD la contribution, à l'erreur sur la valeur binaire naturelle d'une composante, de l'erreur en entrée et des erreurs induites par l'architecture.

$$\Rightarrow \quad D = exp\left[ln(n_i) - \frac{ln(n_x^2 + n_y^2 + n_z^2 + \varepsilon)}{2} - ln(2) \times \varepsilon'\right] = \left[\frac{n_i}{\sqrt{n_x^2 + n_y^2 + n_z^2 + \varepsilon}}\right] \times exp[-ln(2) \times \varepsilon'].$$

$$\Rightarrow \quad D = \left[\frac{n_i}{\|\hat{N}\| \times \sqrt{1 + \frac{\varepsilon}{\|\hat{N}\|}}}\right] \times exp[-ln(2) \times \varepsilon'] \approx N_i \times \left[1 - \frac{0,5 \times \varepsilon}{\|\hat{N}\|}\right] \times exp[-ln(2) \times \varepsilon']$$

$$\Rightarrow \quad \partial D = \partial N_i - \frac{N_i \times \partial \varepsilon}{2 \times \|\vec{N}\|} - N_i \times ln(2) \times \partial \varepsilon'$$

$$\Rightarrow \quad \delta D \leq \frac{1,79}{2^{m+1}} + \varepsilon + 0,70 \times \varepsilon' \leq \frac{1,79}{2^{m+1}} + \left[\frac{1,5}{2^{m1}} + \frac{0,5}{2^{m2}}\right] + \frac{0,18}{2^{m5}} + \frac{0,35}{2^{m6}}$$

A partir de δD on va pouvoir calculer δPS.

$$\Rightarrow \quad \delta PS \leq \sqrt{3} \times \delta D \leq \frac{3,11}{2^{m+1}} + \left[\frac{2,60}{2^{m1}} + \frac{0,87}{2^{m2}}\right] + \frac{0,31}{2^{m5}} + \frac{0,61}{2^{m6}}$$

Reste à déterminer les différentes valeurs de $m1$, $m2$, $m5$ et $m6$ pour que $\delta PS \leq \frac{0,5}{2^{(m-3)}}$ sachant que $m = 11$.

$m1$ $m2$ $m5$ $m6$	Valeur de l'erreur après: produit scalaire	$m1$ $m2$ $m5$ $m6$	Valeur de l'erreur après: produit scalaire
11 11 10 11	$\delta PS < \frac{0,674}{2^8}$	12 12 11 12	$\delta PS < \frac{0,434}{2^8}$
14 11 10 11	$\delta PS < \frac{0,498}{2^8}$	16 10 11 12	$\delta PS < \frac{0,499}{2^8}$

Figure 4•18 L'erreur pour différentes configurations de l'architecture S./L.N.S.

4•4 Conclusion.

• Au cours de ce chapitre, l'étude théorique menée, nous a premièrement permis de définir et d'introduire l'unité de pré-normalisation. Nous allons donc pouvoir concevoir l'architecture de cette unité au cours du chapitre suivant.

De plus, tous les formats de données de l'unité de normalisation ont été définis. Grâce à ces résultats, les coûts exacts des différentes propositions d'architecture pour l'unité de normalisation vont pouvoir être déterminés.

CHAPITRE V: Application au processeur de l'unité d'éclairement proposée.

Au chapitre III nous avons présenté une première proposition d'implémentation matérielle mais l'étude n'a pas été achevée car nous ne connaissions pas encore la taille nécessaire des chemins de données pour obtenir des images ayant la qualité voulue. Le chapitre IV a remédié à cette lacune par l'étude mathématique de la transmission de l'erreur. Il a également montré la nécessité d'introduire une unité de pré-normalisation devant l'unité de normalisation. Nous allons donc pouvoir compléter la présentation de l'unité d'éclairement en commençant par décrire l'architecture de l'unité de pré-normalisation.

Nous examinerons ensuite l'unité de normalisation et nous chiffrerons la complexité de l'architecture utilisant la notation binaire naturelle et celle utilisant la notation logarithmique signée. Pour ces deux architectures, nous effectuerons ensuite le bilan des simulations fonctionnelles et nous comparerons l'erreur maximale effective par rapport à la majoration théorique, calculée au chapitre III.

Ensuite, nous présenterons un exemple de solution technologique pour la réalisation matérielle de l'unité d'éclairement dans un VLSI. Nous utiliserons pour cela le compilateur de silicium Solo 1400 disponible au LIFL.

Nous terminerons la présentation de notre proposition d'implémentation en détaillant l'unité d'éclairement. Les architectures et les algorithmes utilisés pour les unités de calcul du diffus et du spéculaire seront exposés, suivant que la notation logarithmique signée est ou non utilisée.

5•1 L'architecture générale.

Reprenons l'architecture générale du post-éclairement de Phong tel que nous l'avions laissé au chapitre 3. Ajoutons-y l'unité de pré-normalisation issue de l'étude théorique menée au chapitre 4.

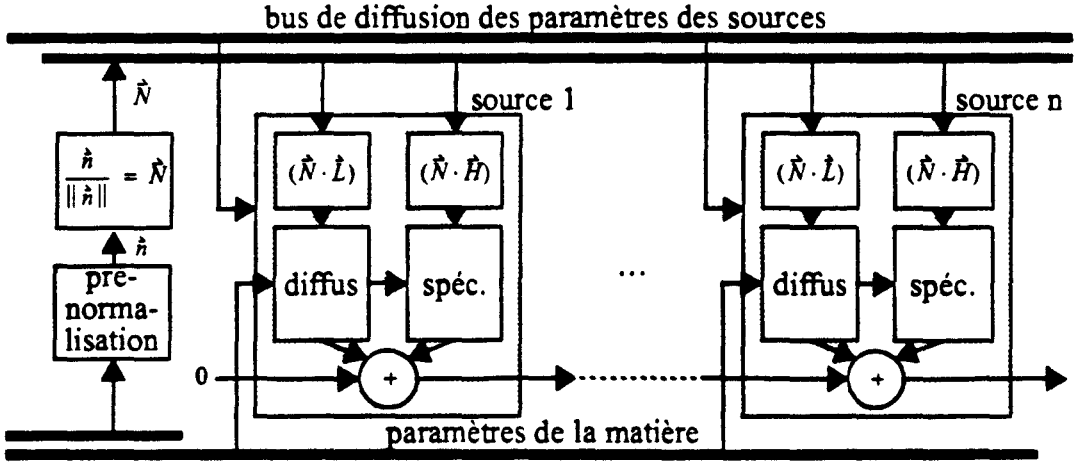


Figure 5•1 Architecture générale.

Par ailleurs, nous avions limité l'étude de l'architecture au chapitre 3 car nous ne disposions pas encore des formats de données qu'il faut utiliser pour obtenir la qualité visuelle demandée. Cette lacune sera comblée dans ce chapitre grâce aux résultats théoriques du chapitre 4.

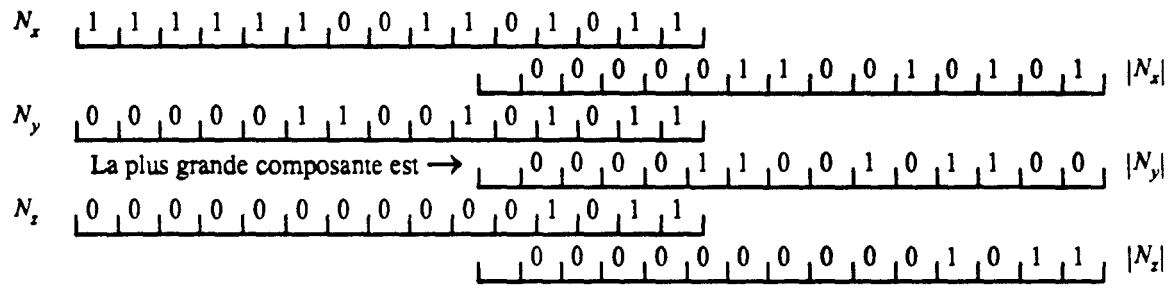
5•2 L'architecture de la pré-normalisation.

L'architecture de l'unité de pré-normalisation, dont la définition formelle a été introduite au paragraphe 4•2•2, n'a pas encore été définie. Il s'agit donc de trouver une unité simple, peu coûteuse, respectant la définition formelle qui a été fixée.

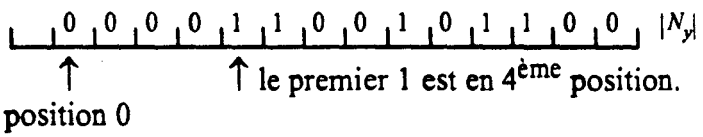
5•2•1 Algorithme et fonctionnement de la pré-normalisation.

• L'unité de pré-normalisation doit effectuer les décalages sur les composantes du vecteur $\vec{N}$ issues des unités de conversion des objets et pixels. L'amplitude du décalage est fixée par la valeur n choisie de telle façon que la propriété énoncée au paragraphe 4•2•2 soit vérifiée. Il faut donc fixer l'algorithme qui permet de trouver la valeur de n.

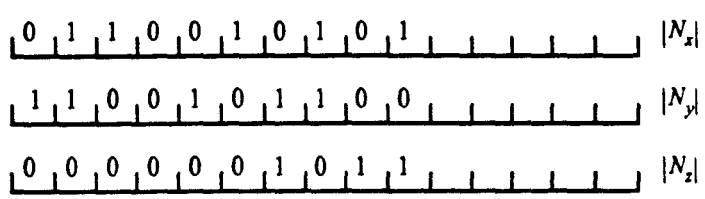
• Il faut d'abord calculer les valeurs absolues des composantes du vecteur puis déterminer la composante ayant la plus grande valeur absolue.



Ensuite, on peut déterminer par quelle puissance de deux doit être multipliée la plus grande composante pour qu'elle appartienne à l'intervalle [0.5,1[. Cela revient, en fait, à calculer la position du premier des bits de poids fort ayant pour valeur 1.



• Enfin, la valeur de n ayant été déterminée, l'unité de décalage peut terminer la pré-normalisation.



En sortie de l'unité de pré-normalisation, il reste à fournir les composantes du vecteur $\vec{N}$ avec la précision demandée, en ne conservant que les bits de poids forts issus du décalage.

5•2•2 L'architecture d'une unité de pré-normalisation.

• La figure ci-dessous présente en détails les opérations qui doivent être effectuées pour "pré-normaliser" le vecteur $\hat{N}$.

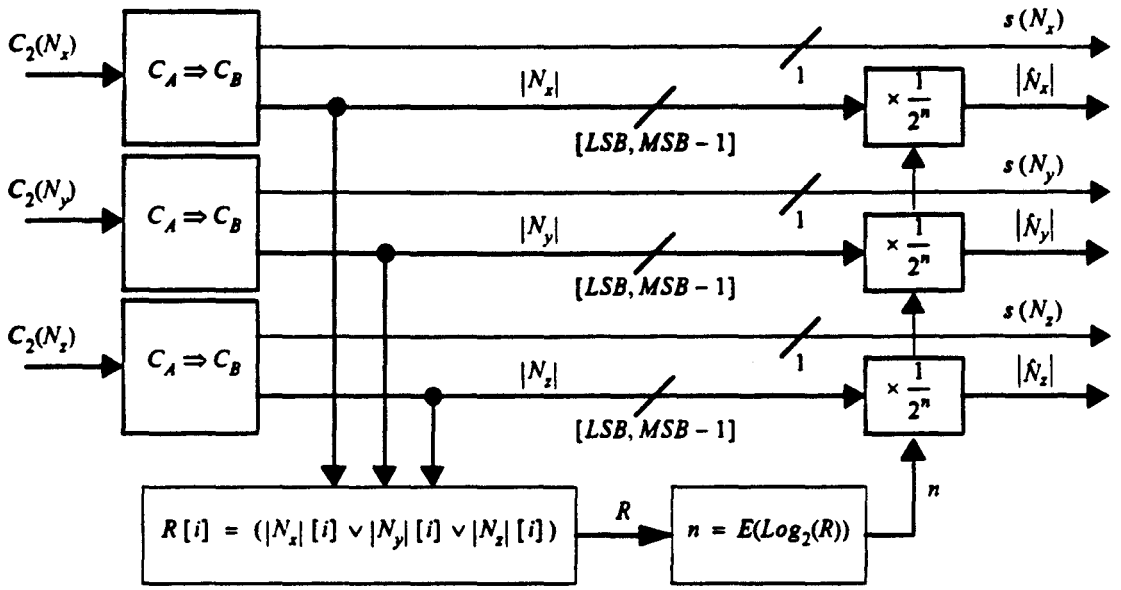


Figure 5•2 Le diagramme de l'unité de pré-normalisation.

• La conversion d'un nombre A, codé en complément à deux vers le codage valeur absolue plus signe, se fait en utilisant l'algorithme suivant :

$$
\begin{array}{lll}
signe(A) & \leftarrow & A[MSB] \\
T[i] & \leftarrow & (A[i] \wedge \neg A[MSB]) \vee (\neg A[i] \wedge A[MSB]) \qquad i \in [0, MSB-1] \\
|A[0, MSB-1]| & \leftarrow & T[0, MSB-1] + A[MSB]
\end{array}
$$

L'addition finale est l'opération la plus coûteuse de l'algorithme de conversion, du fait de la propagation de la retenue. Le temps d'exécution de l'opération sera d'autant plus important que le nombre de bits utilisés est grand. A l'endroit où la pré-normalisation est utilisée, les normales sont encore sur un nombre conséquent de bits. L'erreur commise, en omettant l'addition finale, est faible. De plus, cette erreur n'est commise que pour la conversion des nombres négatifs et elle peut encore être corrigée, par la suite, si nécessaire.

On effectue la conversion du codage en complément à deux vers le codage séparant le signe de la valeur absolue, pour les trois composantes N_x, N_y et N_z du vecteur $\hat{N}$.

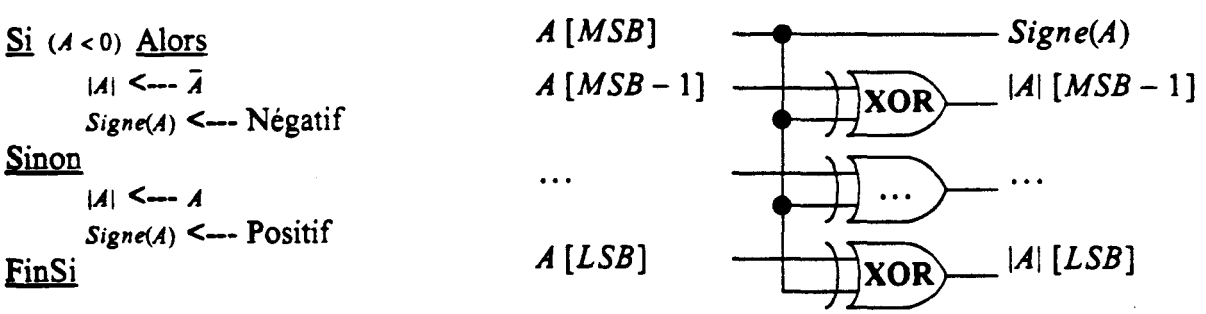


Figure 5•3 Le programme de conversion et l'architecture du module.

• Il faut maintenant déterminer la composante qui a la plus grande valeur absolue, puis dans cette composante, la position du bit de poids le plus élevé de valeur 1. On peut facilement démontrer que cela revient à déterminer la position du bit de poids le plus élevé de valeur 1 dans le codage de R.

On définit : $R[i] = |N_x|[i] \vee |N_y|[i] \vee |N_z|[i] \qquad i \in [LSB, MSB-1]$.

• Le module calculant le nombre de décalages à partir de R est un encodeur de priorité qui peut facilement être réalisé avec un PLA. Comme le montre la figure ci-dessous, on recherche la position du premier 1 dans R.

$R[LSB, MSB-1]$	n	Commande des décalages. 1	2	4	8
1 X X X X X X X X X X X X X X	0	0	0	0	0
0 1 X X X X X X X X X X X X X	1	1	0	0	0
0 0 1 X X X X X X X X X X X X	2	0	1	0	0
0 0 0 1 X X X X X X X X X X X	3	1	1	0	0
0 0 0 0 1 X X X X X X X X X X	4	0	0	1	0
...	...	...			

Figure 5•4 Description du fonctionnement et de la programmation du PLA.

• L'étage final de l'unité de pré-normalisation est un décaleur variable n'agissant que sur la partie non signée des composantes de la normale. Il est constitué d'une succession de multiplexeurs agissant chacun sous le commandement d'un bit de n.

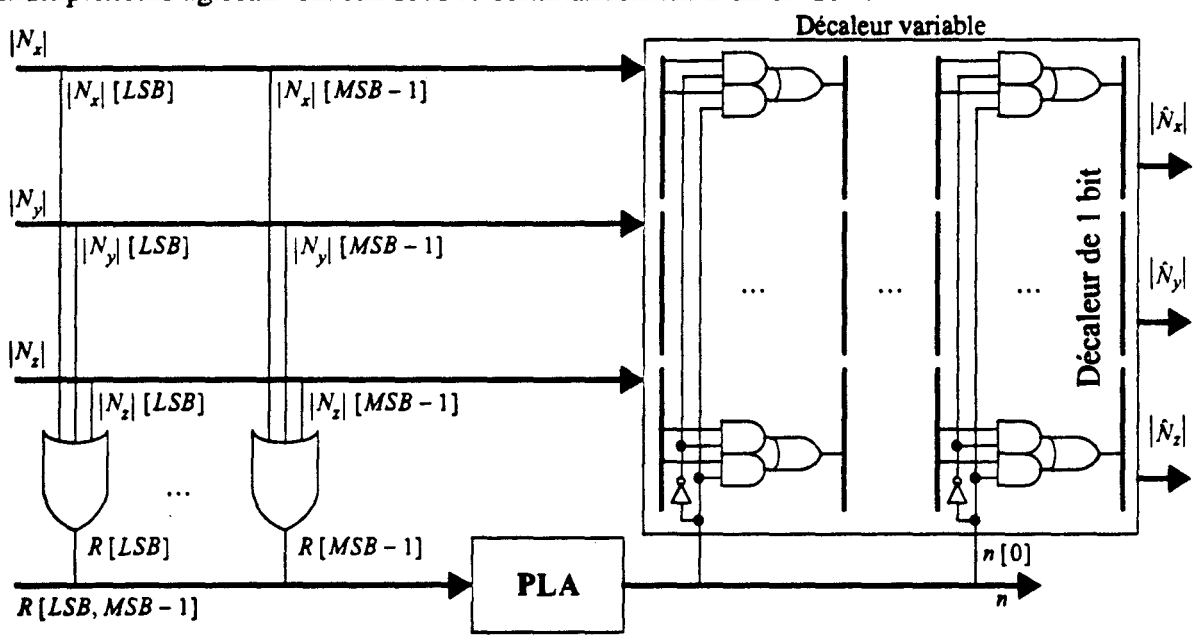


Figure 5•5 L'architecture de l'unité de calcul de R, de n et du décaleur variable.

5•2•3 La place de l'unité de pré-normalisation dans le pipeline de rendu.

• Dans le cas de la machine I.M.O.G.E.N.E., les vecteurs des normales sont calculés dans les processeurs objets. Puis ils transitent dans l'unité de décision qui effectue l'élimination des parties cachées. Enfin ils arrivent à l'unité de normalisation. C'est sur ce parcours que doit être placée l'unité de pré-normalisation.

Si on place celle-ci juste devant l'unité de normalisation (cf Figure 5•6 a), la machine ne devra en contenir qu'une seule mais les vecteurs des normales transiteront dans l'unité de décision sur de larges chemins de données. Par contre, si on la place juste derrière chaque processeur objet (cf Figure 5•6 b), les vecteurs des normales transiteront sur des chemins de données de taille réduite mais la machine devra en contenir autant que de processeurs objets.

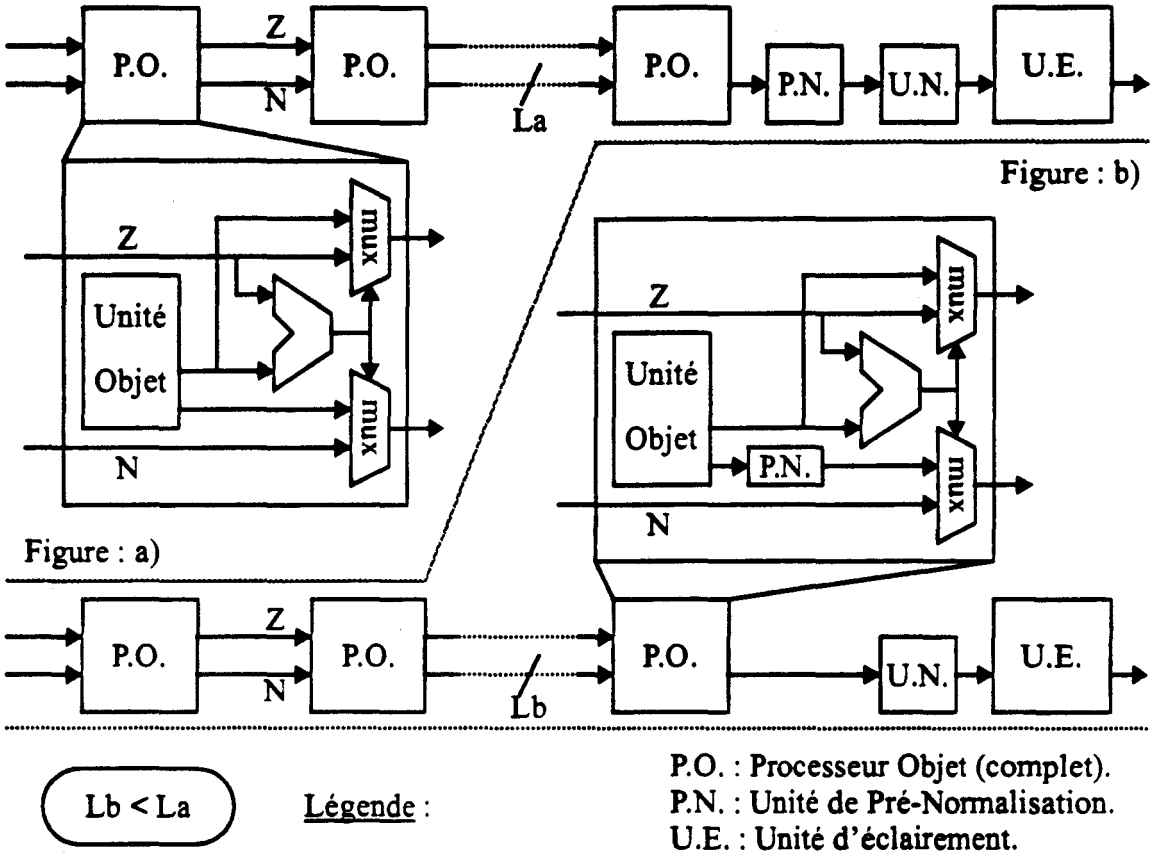


Figure 5•6 Disposition de l'unité de pré-normalisation dans le pipeline d'I.M.O.G.E.N.E.

Les processeurs objets de la machine I.M.O.G.E.N.E. sont (ou seront) en grande partie réalisés avec des VLSIs. Pour le processeur objet de la primitive facette, un VLSI a déjà été défini. Vis à vis du problème de la place de l'unité de pré-normalisation dans la machine I.M.O.G.E.N.E., la décision a été prise de façon à diminuer le coût du VLSI. Il faut savoir que le coût du boîtier représente une bonne part du prix du circuit intégré, lors de sa réalisation.

La taille des chemins de données des vecteurs directeurs des normales a une influence directe sur le nombre de broches des VLSIs. Comme l'unité de pré-normalisation est de taille très petite, il est intéressant de l'inclure dans chaque processeur objet pour diminuer les connections externes. [33]

• On pourrait imaginer aussi que l'unité d'éclairement doit être placée sur une machine utilisant un véritable tampon en profondeur (Z-buffer). Dans ce cas il faut garder en mémoire, pour chaque pixel de l'écran ou de la zone écran, la profondeur et la normale. La taille de la mémoire utilisée dépendra de la taille de l'écran ou de la zone écran, de la largeur du codage binaire, de la profondeur et de la largeur du codage binaire du vecteur normal.

Dans un tel système, la place de l'unité de normalisation est à la sortie du tampon en profondeur et non à l'entrée. Ainsi on ne normalise que les vecteurs $\hat{N}$ qui contribuent réellement au calcul de l'éclairement.

De façon à diminuer la quantité de mémoire nécessaire, il est, par contre, souhaitable de disposer l'unité de pré-normalisation en entrée du tampon de profondeur. Sa complexité étant très petite, il ne sera pas difficile de la concevoir de telle façon qu'elle fonctionne à la vitesse d'entrée des données dans le tampon en profondeur.

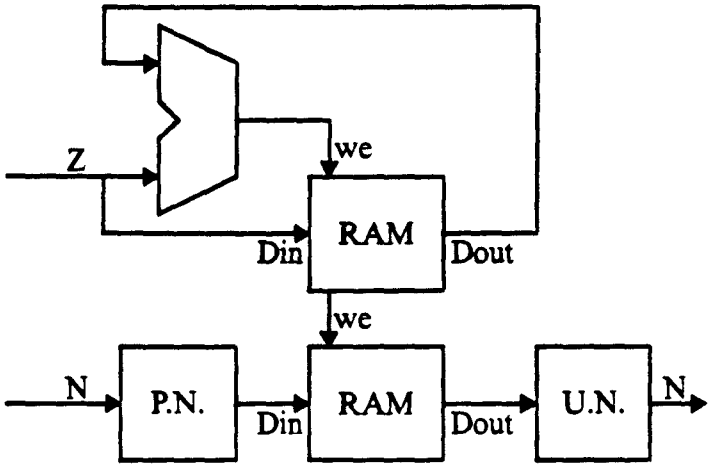


Figure 5•7 Place de l'unité de pré-normalisation dans le cas du tampon en profondeur.

5•3 L'architecture de la normalisation.

5•3•1 Coût de l'unité de normalisation de base en binaire naturel.

• Au paragraphe 4•3•1, nous avons établi qu'il fallait au minimum 11 bits pour coder la valeur absolue d'une composante du vecteur $\hat{N}$ issu de l'unité de pré-normalisation. Au paragraphe 4•3•2, on a montré qu'il était intéressant d'accroître de 1 cette taille afin de se réserver une plus grande marge d'erreur qui s'avère utile pour la construction de l'unité de normalisation. Afin de construire la version de base, nous allons limiter au maximum le coût.

On suppose donc : $O(\hat{n}) = 13$ et $O(\vec{N}_{normé}) = 13$.

• Plusieurs solutions peuvent être envisagées pour la réalisation de l'unité de normalisation travaillant en binaire naturel. On peut au choix effectuer ou non les regroupements illustrés sur la figure suivante.

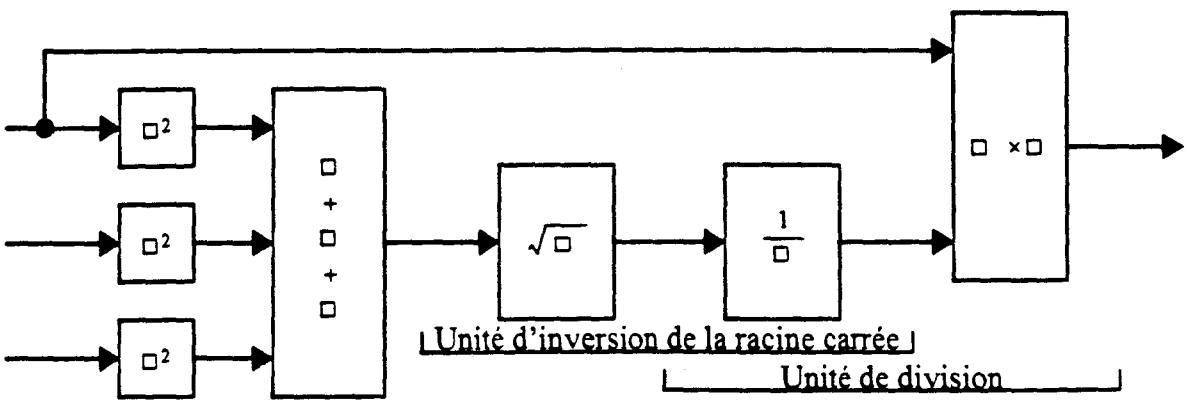


Figure 5•8 Les architectures utilisant le binaire naturel.

• Pour concevoir les autres unités fonctionnelles composant l'unité de normalisation, nous allons profiter au maximum de la taille réduite de leurs entrées et sorties. L'étude effectuée au paragraphe 4•3•3 nous a montré que parmi les diverses solutions, l'utilisation d'une unité d'inversion de la racine carrée, couplée à trois unités de multiplication s'avère être la solution la plus économique.

5•3•1•1 L'utilisation d'une unité d'inversion de la racine carrée.

• Le calcul de l'élévation au carré des composantes est indépendant du signe de celles-ci. On limite donc à 12 le nombre de bits entrant dans l'unité d'élévation au carré. En sortie de cette même unité, le nombre de bits est réduit à 16 d'après l'étude sur la transmission de l'erreur présentée au paragraphe 4•3•2. Vu sa taille, cette unité peut donc se résumer à une mémoire ROM possédant 12 bits d'adresses, 16 bits de données et programmée avec les valeurs appropriées. (Lors de la réalisation matérielle, il sera nécessaire d'utiliser deux ROMs 8 bits ayant un temps d'accès très court et d'une capacité de 4 K octets. Faute de disposer parmi les composants électroniques discrets de telles ROMs, il me parait encore plus judicieux de les remplacer par des SRAMs rapides que l'on chargera à la mise sous tension.)

• L'unité d'addition comportera donc 3 entrées pour des entiers positifs codés sur 16 bits et une sortie du résultat sur 15 bits. Plutôt que de mettre deux additionneurs 16 bits l'un derrière l'autre, pour effectuer l'addition des trois nombres, on peut simplifier en utilisant localement la notation redondante. On décompose l'opération en deux blocs. Le premier bloc transforme les trois nombres à additionner en deux nombres dont l'addition fournira le même résultat. ($A+B+C = E+2\times F$) La vitesse de cette transformation peut être importante, car elle ne requiert pas de propagation de retenue. (Voir Figure 5•9.) Le second bloc réalisera, quant à lui, l'addition sur 17 bits des deux nombres (E et $2\times F$) issus du premier bloc. En sortie on ne gardera que les 15 bits de poids fort qui sont les seuls à nous intéresser.

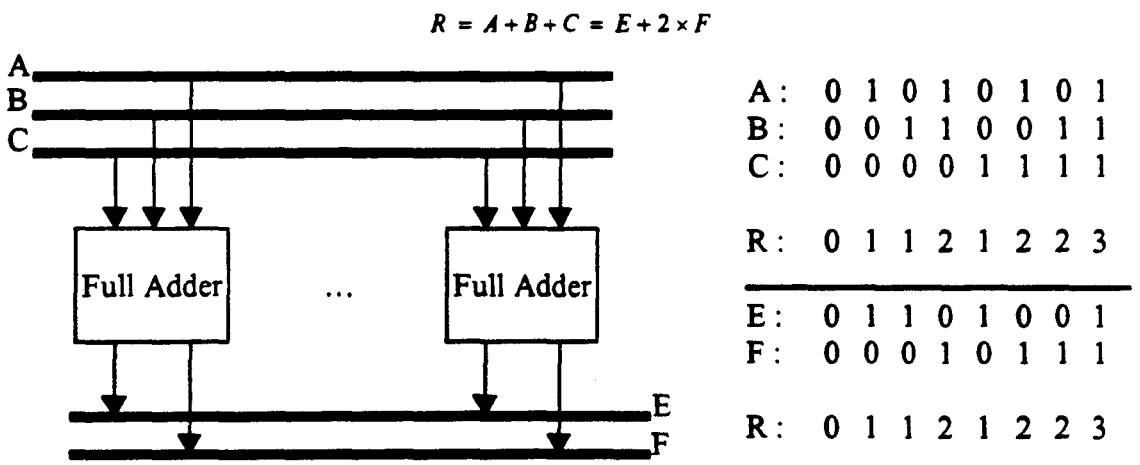


Figure 5•9 Architecture de l'additionneur partiel.

• En entrée de l'unité calculant la racine carrée puis l'inverse, les nombres sont codés sur 15 bits. En sortie, il en est de même. Vu sa taille, cette unité peut, elle aussi, se résumer à une mémoire ROM. (Lors de la réalisation, il faudra mettre deux ROMs 8 bits de 32 K octets ou bien deux RAMs 8 bits de taille équivalente dont on chargera le contenu à la mise sous tension.) Certes sa taille est conséquente mais, programmée avec les valeurs appropriées, elle réalisera efficacement le calcul.

Cette approche va présenter un intérêt supplémentaire. Comme nous en avons déjà fait état pour un problème similaire au paragraphe 3•3•2•1, il est à déplorer que le 12ème bit de la sortie de l'unité de normalisation ne serve que pour le codage de la valeur 1. La solution consiste à modifier, quelque peu, la fonctionnalité de l'unité de normalisation. Son but ne sera plus de satisfaire la condition $\|\hat{N}\| = 1$ mais la condition :

$$\|\hat{N}\| = \frac{2047}{2048}$$

Ainsi le $12^{ème}$ bit est toujours nul et peut donc être supprimé du codage. Bien sûr, ce petit changement modifie la formule de transmission de l'erreur qui devient dans le cas que nous avons choisi :

$$\delta N < \frac{1,79}{2^{m+1}} + \frac{6,0}{2^{m1}} + \frac{2,0}{2^{m2}} + \frac{0,5}{2^{m4}} + \frac{0,5}{2^{m}} \times \frac{2^{m}}{2^{m}-1} = \frac{1,79}{2^{12}} + \frac{6,0}{2^{16}} + \frac{2,0}{2^{13}} + \frac{0,5}{2^{13}} + \frac{0,5}{2^{11}-1} \leq \frac{2,22}{2^{11}} \Rightarrow \delta PS < \frac{0,48}{2^{8}}$$

Cela n'induit aucune perturbation notable dans la transmission de l'erreur. Il est donc possible d'effectuer une telle modification de la fonctionnalité de l'unité de normalisation.

• Pour que cette condition puisse être vérifiée et que l'éclairement soit correctement calculé, quelques modifications, dans l'unité de normalisation et dans le processeur d'éclairement, seront bien sûr, nécessaires. Pour commencer, il faudrait que le module calculant à l'origine l'inverse de la racine carrée fournisse son résultat en l'ayant préalablement multiplié par $\frac{2047}{2048}$. Cela ne pose pas de problème, il suffit de programmer la ROM en tenant compte de ce facteur multiplicatif.

Ensuite, il faut modifier la norme des vecteurs $\hat{L}$ et $\hat{H}$ pour compenser le facteur multiplicatif lors du calcul du produit scalaire. ($\|\hat{L}\| = \|\hat{H}\| = \frac{2048}{2047}$) D'autres suggestions concernant les modifications à apporter au processeur d'éclairement seront étudiées dans le paragraphe le concernant.

• Enfin reste les trois modules de multiplication, fournissant en sortie, les trois composantes du vecteur $\hat{N}$ normalisé. Les multiplications n'affectant pas le signe des composantes, seule la valeur absolue codée sur 12 bits sera présentée à la première entrée de l'unité de multiplication. Les 14 bits issus de l'unité de calcul de l'inverse de la racine carrée, le seront à la seconde entrée. (Il n'est plus nécessaire d'utiliser 15 bits, car le facteur multiplicatif utilisé opère une diminution, de la plus haute valeur issue de l'unité d'inversion de la racine carrée, de 16384 à 16376. Cette dernière valeur se code sur seulement 14 bits.)

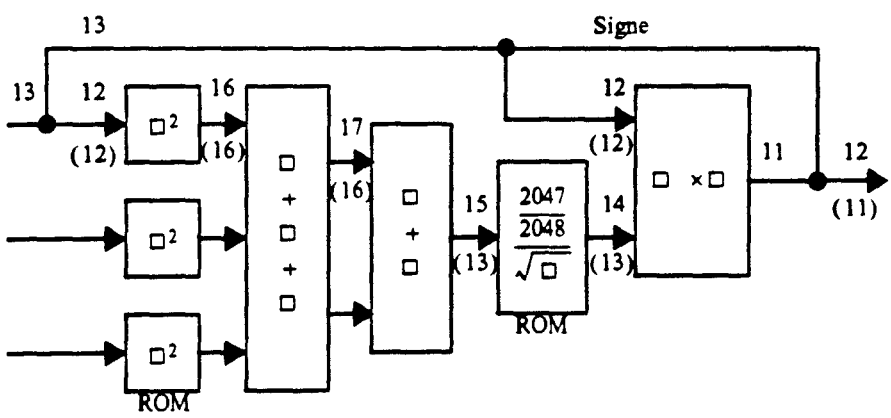


Figure 5•10 Synoptique de l'unité de normalisation en version de base.

• Récapitulons : il faut trois ROMs (4K fois 16 bits), une ROM (32K fois 14 bits), 3 multiplicateurs (12 et 14 bits d'entrées respectives et 11 bits de sortie), un additionneur (17 bits) et 16 cellules d'addition (full-adder 1 bit). Soit un total de 640 K bits répartis au sein de 4 mémoires mortes. L'élément le plus coûteux, qui figure dans cette version de l'unité de base de normalisation, est l'unité de multiplication. Pour que cette unité puisse faire partie intégrante d'une machine calculant l'éclairement de Phong en temps réel, c'est à dire recalculant l'image au rythme du balayage écran, il faudrait utiliser des multiplicateurs câblés fonctionnant suffisamment rapidement. Si on concevait un VLSI incluant tous les modules composant l'unité de normalisation, exception faite des ROMs qui devraient être ajoutées en externe, le boîtier du VLSI comporterait 111 broches. (Sans compter les alimentations, horloges et autre signaux de contrôle.)

5•3•1•2 L'utilisation des autres solutions.

L'utilisation d'une unité de division est relativement compromise du fait de sa complexité. Mais, ce n'est pas une solution à écarter définitivement. Dans un composant VLSI, il est possible d'intégrer des pipelines de division comme le propose G. KNITTEL dans [31]. L'utilisation de la notation redondante devrait permettre la diminution du coût d'un tel pipeline en accélérant les calculs et en diminuant ainsi le nombre d'étages du pipeline.

5•3•2 Coût de l'unité de normalisation en S./L.N.S.

• Comme on a pu le remarquer au paragraphe 4•3•4, il existe plusieurs solutions. Suivant le choix que l'on va effectuer, la dynamique de sortie des composantes du vecteur $\hat{N}$ normé n'est pas la même. Plus on aura besoin d'une large dynamique, plus l'architecture de l'unité calculant le produit scalaire et la conversion en binaire naturel sera compliquée. Or cette unité est présente dans chaque processeur d'éclairement. Il est donc préférable de compliquer l'architecture interne de l'unité de normalisation dès lors que cela permet de diminuer la dynamique de sortie. Il convient donc de choisir la solution suivante :

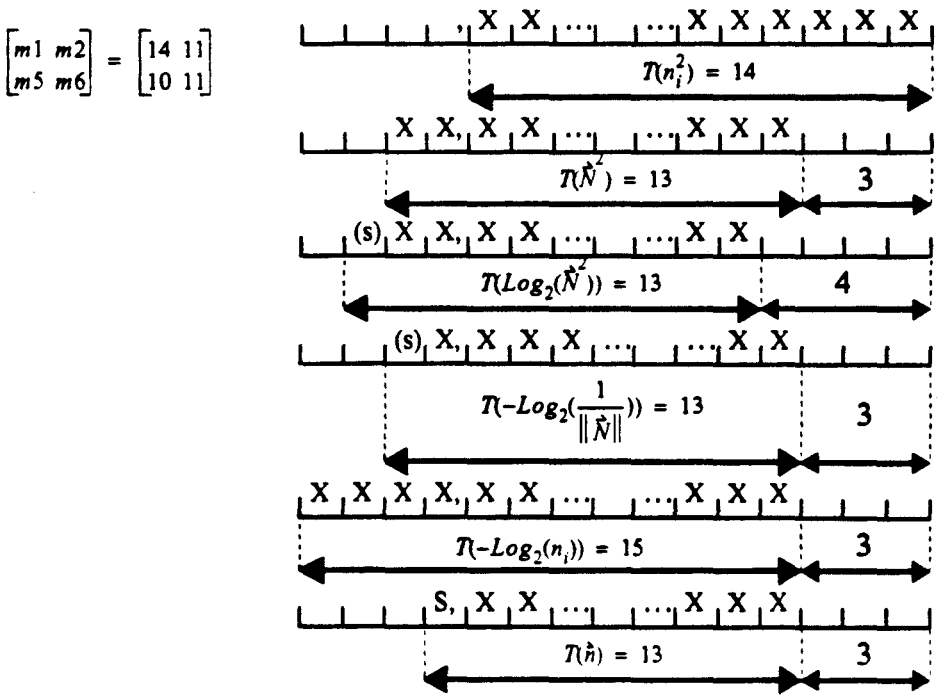


Figure 5•11 Le codage S/L.N.S. au sein de l'unité de normalisation.

• L'opération d'extraction de la racine carrée s'effectue par un simple décalage en codage S./L.N.S. L'inversion s'obtient en prenant le complément à deux du nombre L.N.S.

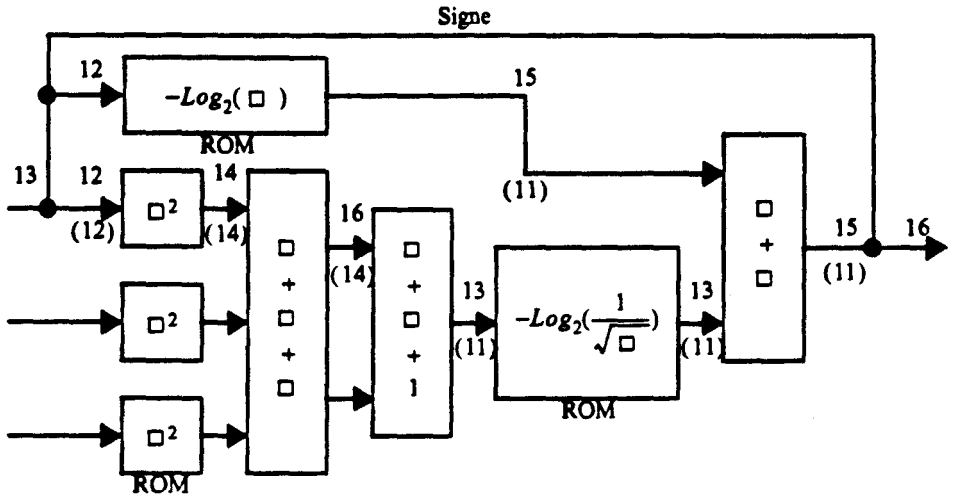


Figure 5•12 Synoptique de l'unité de normalisation avec S/L.N.S. (en version de base)

• Chacune des trois unités qui transforment en S/L.N.S. les composantes du vecteur $\hat{N}$ avant normalisation, est réalisée avec une ROM de 4K mots de 15 bits. (Soit physiquement 2 ROMs de 4K octets) Les unités d'élévation au carré sont réalisées, chacune, avec une ROM de 4K mots de 14 bits. (Soit matériellement 2 ROMs de 4K octets) La ROM, calculant l'inverse de la racine carrée, est remplacée par une ROM de 8K mots de 13 bits. (Soit matériellement 2 ROMs de 8K octets)

• De façon à respecter scrupuleusement le calcul théorique de la transmission de l'erreur, il est important d'effectuer de vrais arrondis mathématiques. On ne peut donc pas simplement éliminer les bits devenus inutiles quand on change la dynamique des données. Lorsqu'on diminue de i bits la dynamique, il est nécessaire d'ajouter au préalable la valeur $2^{(i-1)}$. Il faut en tenir compte lors de la programmation des ROMs. Il est également nécessaire d'ajouter la valeur 4 au cours de l'addition servant au calcul de la norme. Cet ajout peut être scindé. On peut intégrer simplement cette opération dans l'architecture, en se servant de la retenue d'entrée de l'additionneur pour ajouter 1 et en programmant chacune des 3 ROMs de façon à ajouter 1 à chaque nombre entrant dans l'additionneur. Cette solution présente l'avantage de ne pas accroître le nombre de ROMs dont la programmation est différente. (C'est ce genre de détail qui peut s'avérer important pour une version industrielle.)

• Pour chacun des trois additionneurs finals, il faut intégrer un test indiquant si le résultat est négatif. Car il convient, dans ce cas, de le rendre nul grâce à une série de portes "et". Théoriquement, le nombre ne devrait jamais être négatif, mais il s'avère que les erreurs introduites peuvent parfois se combiner de façon à fournir un résultat légèrement négatif, alors qu'il devrait être nul.

Il faut également détecter les résultats qui sont supérieurs à 32767. Cela représente un nombre infini qu'il convient de coder simplement 32767. Un test de dépassement de capacité et une série de portes "ou" permettront d'effectuer convenablement et efficacement ce travail.

```
Pour i = 0 à 4095 Faire
   ROM_SQ[i] = ( i*i + 1536) >> 10
Fait
```

```
Pour i = 0 à 4095 Faire
   val = 4096 * ln( 4096 / i ) / ln(2)
   ROM_LOG[i] = int( val+0,5 )
Fait
```

```
Pour i = 512 à 6143 Faire
   val = 2048 * ln( i / 4096 ) / ln(2)
   Si val est positif
      Alors
         ROM_INV[i] = int( val+0,5)
      Sinon
         ROM_INV[i] = int( val+0,5)
   FinSi
Fait
```

Légende :

ln : est le log népérien.
int : est la fonction qui supprime la partie décimale d'un nombre réel

Figure 5•13 Les algorithmes de programmation des ROMs.

5•3•3 Simulation des unités de normalisation.

Vu la taille réduite des entrées de l'unité de normalisation, il est possible de simuler son action sur l'ensemble des vecteurs pouvant être normalisés. Les trois entrées des composantes du vecteur ont des rôles symétriques et le signe n'intervient pas dans la normalisation. Il est donc possible de restreindre l'espace des vecteurs afin de ne tester qu'une seule fois les vecteurs semblables à la permutation près des composantes ou à quelques changements de signes près.

L'ensemble des vecteurs à normaliser est : $\left\{ (N_x, N_y, N_z) / \begin{pmatrix} 2048 \le N_x < 4096 \\ 0 \le N_y < N_x \\ 0 \le N_z < N_y \end{pmatrix} \right\}$.

Il comporte 10.027.882.496 éléments. Chaque vecteur $\vec{N}$ est normalisé par l'unité présentée Figure 5•10.

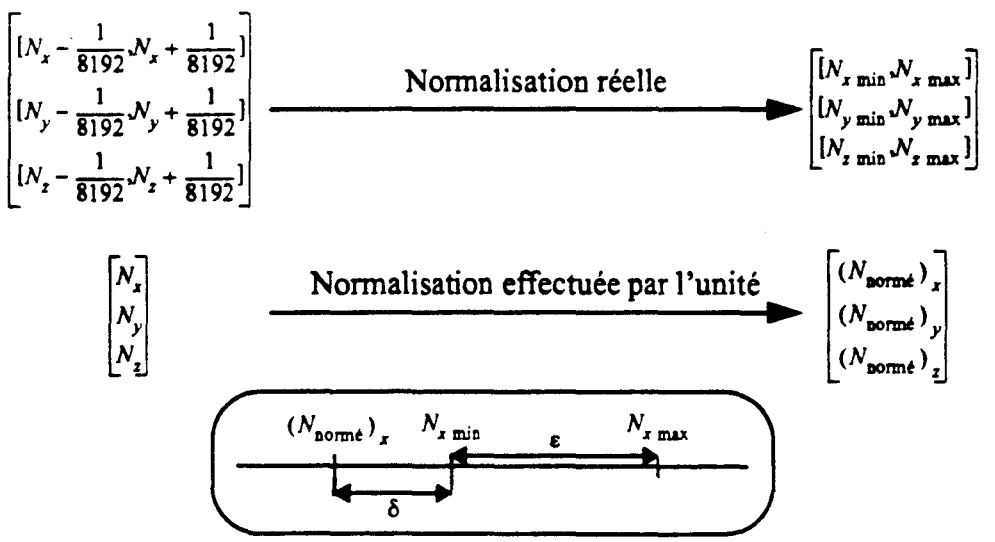


Figure 5•14 Comparaison, avec les composantes réelles, des composantes résultant de l'unité de normalisation.

• Les composantes du vecteur $\hat{N}$ avant la normalisation sont codées sur 12 bits. (signe non compris) Sur chaque composante, on a une incertitude de 1/4096 quant à sa valeur réelle. Comme on a cherché à garder le maximum de précision, on considère que l'erreur sur la valeur réelle d'une composante est inférieure à 1/2 LSB. Chaque triplet (N_x, N_y, N_z) correspond donc à un ensemble de vecteurs réels. On peut déterminer les bornes de l'ensemble des vecteurs réels, correspondant au triplet, après normalisation. Pour que la simulation puisse déterminer le degré d'imprécision induite par l'architecture de l'unité de normalisation, on compare les composantes résultantes aux bornes de l'intervalle des valeurs réelles.

L'erreur sur la composante après normalisation est définie comme la distance entre la valeur de la composante résultante et la plus éloignée des bornes de l'intervalle contenant les composantes réelles. On calcule la valeur maximale pour tous les vecteurs normalisés lors de la simulation et on obtient :

$$\varepsilon = 0{,}000285 = \frac{0{,}582959}{2047} \text{ et } \delta = 0{,}000817 = \frac{1{,}671297}{2047}.$$

• On procède à la même simulation pour l'unité, utilisant le codage S/L.N.S., présentée Figure 5•12. On obtient :

$$\delta = 0{,}000964 = \frac{1{,}973244}{2047}$$

• Pour les deux architectures testées, les simulations ont donné des résultats en accord avec la théorie. Cela nous montre même que l'on dispose d'une certaine marge d'erreur pour la conception de l'unité d'éclairement. Si on compare les deux architectures, il faut au total 672 K bits de ROM dans le cas du calcul en binaire naturel et seulement 452 K bits de ROM dans le cas du calcul utilisant la notation S./L.N.S. De plus, la seconde architecture utilise des additions au lieu de multiplications, elle est donc plus intéressante, du simple point de vue de la fabrication de l'unité de normalisation. Ce choix devra être confirmé par l'étude du processeur d'éclairement. Comme l'unité de normalisation pourra être couplée avec plusieurs unités d'éclairement, il n'est pas souhaitable de déplacer une partie de la complexité de l'unité de normalisation dans les processeurs d'éclairement.

5•3•4 Un exemple de solutions technologiques pour un VLSI.

L'étude qui précède nous indique deux solutions attrayantes pour fabriquer une unité de normalisation. L'utilisation de ROMs, ou éventuellement de RAMs initialisées à la mise sous tension, présente un intérêt dans le cadre d'une carte d'éclairement utilisant des composants discrets. Examinons maintenant la possibilité d'intégrer l'unité de normalisation dans un VLSI. La taille des ROMs devient alors un facteur critique pour la réalisation. Il convient donc de vérifier si l'implémentation effective des opérateurs (multiplication, extraction de la racine carrée, inversion ou bien division) ne serait pas de taille plus réduite. Si le but visé en réalisant un VLSI est d'accroître les performances visuelles, (devant alors augmenter la précision des calculs et la taille des ROM.) cela sera tout naturellement le cas. (La taille des ROMs croît de façon exponentielle alors que la taille des opérateurs de multiplication, de division et d'extraction de la racine carrée augmente de façon polynomiale.)

• Bien que tous les opérateurs n'aient pas encore été définis, les premiers essais effectués avec le logiciel SOLO1400 ont permis d'apporter quelque réponses précises sur le sujet.

• Voyons, d'abord la place occupée par les ROMs dans le cas de la seconde architecture : (utilisant la notation S./L.N.S.) Il est nécessaire de concevoir 2 types de ROMs. La première de 4 K mots de 29 bits fournira, à partir de la partie décimale d'une composante de la normale, son carré sur 14 bits et son logarithme sur 15 bits. Cette ROM est crée par SOLO1400 (en utilisant la technologie 1,5 µm) sous la forme d'un bloc de 2500 µm sur 5200 µm soit une surface de 14 mm^2. Il faut trois de ces ROMs dans l'unité de normalisation. Le second type de ROM dont la capacité est de 8 K mots de 13 bits est crée sous la forme d'un bloc de 2100 µm sur 4700 µm soit une surface de 10 mm^2. La place totale occupée par toutes les ROMs est de 50 mm^2. A cela il faut ajouter un certain nombre de portes logiques, pour créer les additionneurs. La complexité de l'ensemble de ces portes, peut facilement être estimée à 8400 transistors en prévoyant de découper l'unité de normalisation sous la forme de trois étages de pipeline. En se référant à d'autres chips, on peut estimer cette place à environ 12 mm^2. La surface du chip devrait donc avoisiner les 62 mm^2 sans compter la place prise par les pattes d'entrée-sortie.

La vitesse de fonctionnement d'une telle unité de normalisation est fixée par la vitesse des ROMs. SOLO1400 nous indique que la plus lente des ROMs utilisées ici, a un temps de lecture maximum de 88 ns. A ce temps, il faut encore ajouter le temps de set-up maximum, des bascules D utilisées pour délimiter les étages de pipeline (11 ns). Soit un total de 99 ns et donc une fréquence de fonctionnement maximale de 10 MHz.

• Cette fréquence de fonctionnement peut paraître faible, comparée à celle que l'on peut obtenir en utilisant des RAMs statiques FLASH externes qui ont des temps d'accès pouvant descendre jusqu'à 15 ns. Integrated Device Technologie, Inc propose par exemple le IDT7MC4032 un module SRAM technologie CMOS de 16 K mots de 32 bits ayant un temps d'accès de 15ns. [27]

• Voyons maintenant la place occupée par les ROMs et les unités de multiplication dans le cas de la première architecture (utilisant la notation binaire naturelle). Du fait des contraintes chronologiques de SOLO1400, il est nécessaire de scinder la ROM servant au calcul de l'inverse de la racine carrée en quatre ROMs. La surface totale occupée par ces quatre ROMs est de 40 mm^2. La taille d'une ROM effectuant l'élévation au carré d'une composante du vecteur normal est de 8,2 mm^2 tandis que la taille d'une unité de multiplication 12 bits par 12 bits est de 2,5 mm^2. Dans le cas de la conception d'un circuit intégré avec SOLO1400, il est donc préférable d'utiliser des multiplicateurs plutôt que des ROMs.

La taille de chacun des trois multiplicateurs finals de l'unité de normalisation est de 3,1 mm^2. A tout cela il faut ajouter la place occupée par les portes des additionneurs, des multiplexeurs à coupler aux quatre ROMs et des bascules D utilisées pour délimiter les étages de pipeline. En se référant à d'autres chips, on peut estimer cette place à environ 11 mm^2. La surface du chip devrait donc avoisiner les 67 mm^2, sans compter la place prise par les pattes d'entrée-sortie.

La vitesse de fonctionnement d'une telle unité de normalisation est fixée par la vitesse des ROMs et des multiplicateurs. SOLO1400 nous indique que la plus lente des ROMs utilisées ici, a un temps de lecture maximum de 90 ns. Les multiplicateurs ont des temps de propagation plus courts; ils ne pénalisent donc pas la vitesse du chip. La fréquence de fonctionnement maximale sera de 10 MHz.

• L'unité d'inversion de la racine carrée occupe environ 50% de la surface du chip. En utilisant une RAM statique FLASH externe, il sera possible de récupérer cette place dans le but de doubler les multiplicateurs et ainsi accroître la fréquence de fonctionnement dans des proportions intéressantes. Le plus lent des multiplicateurs utilisés fournit un résultat en 51 ns; en ajoutant à cela le temps de set-up des bascules D, on obtient 62 ns. Si on utilise deux unités travaillant en parallèle, un nouveau vecteur sortira de l'unité de normalisation toutes les 31 ns, soit une fréquence de fonctionnement maximale de 33 MHz. Le chip devrait disposer d'environ 100 pattes sans compter les alimentations. La conception devra prendre en compte la nécessité de charger la RAM externe à la mise sous tension.

5•3•5 L'utilisation de processeurs de traitement de signal.

Outre la possibilité de concevoir l'unité de normalisation en composant discret ou sous la forme d'un VLSI, il y a encore la solution consistant à programmer un ou plusieurs processeurs de traitement de signal. Tant qu'il ne sera pas nécessaire d'accroître la précision des calculs, cette solution ne peut rivaliser avec l'architecture qui a été proposée. Comme nous l'avons montré au chapitre II la masse de calculs est extrêmement importante.

Certes la technologie de ces composants fait des progrès, leur puissance augmente et ils peuvent travailler sur des données de plus en plus larges, chaque année. Cela apporterait donc une solution attrayante si nous voulions accroître la qualité de l'image et devions ainsi augmenter la précision des calculs. L'augmentation de la taille des ROMs de notre architecture deviendrait démesurée par rapport à la complexité d'un réseau de DSPs développant la même puissance.

Pour établir le point limite où le rapport de force, entre notre architecture et un tel réseau, basculerait, il faudrait déterminer le processeur le mieux adapté. Le TMS 320C40 de Texas Instrument [50], disposant d'instructions flottantes rapides pour la division ou le calcul de l'inverse de l'extraction de la racine carrée, arrivera vraisemblablement en bonne place du palmarès.

5•4 Le processeur d'éclairement.

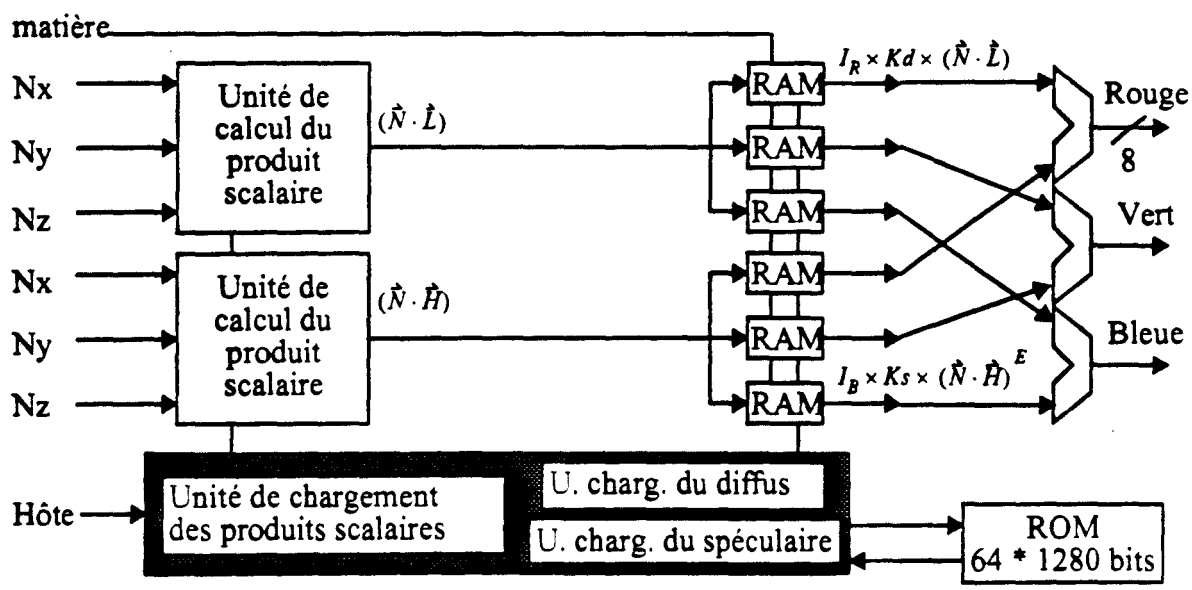


Figure 5•15 Architecture d'un processeur d'éclairement.

Maintenant que l'unité de normalisation est entièrement définie, on peut étudier le cas du processeur d'éclairement. Suivant l'architecture choisie pour l'unité de normalisation et plus précisément le type de notation employée, deux architectures ont été proposées dans le chapitre 3 au paragraphe 3•4. Les différences entre les deux architectures résident au niveau des modules de calcul des produits scalaires.

5•4•1 La version pour le binaire naturel.

La première des deux architectures concerne l'utilisation de la notation binaire naturel. L'architecture est alors la suivante :

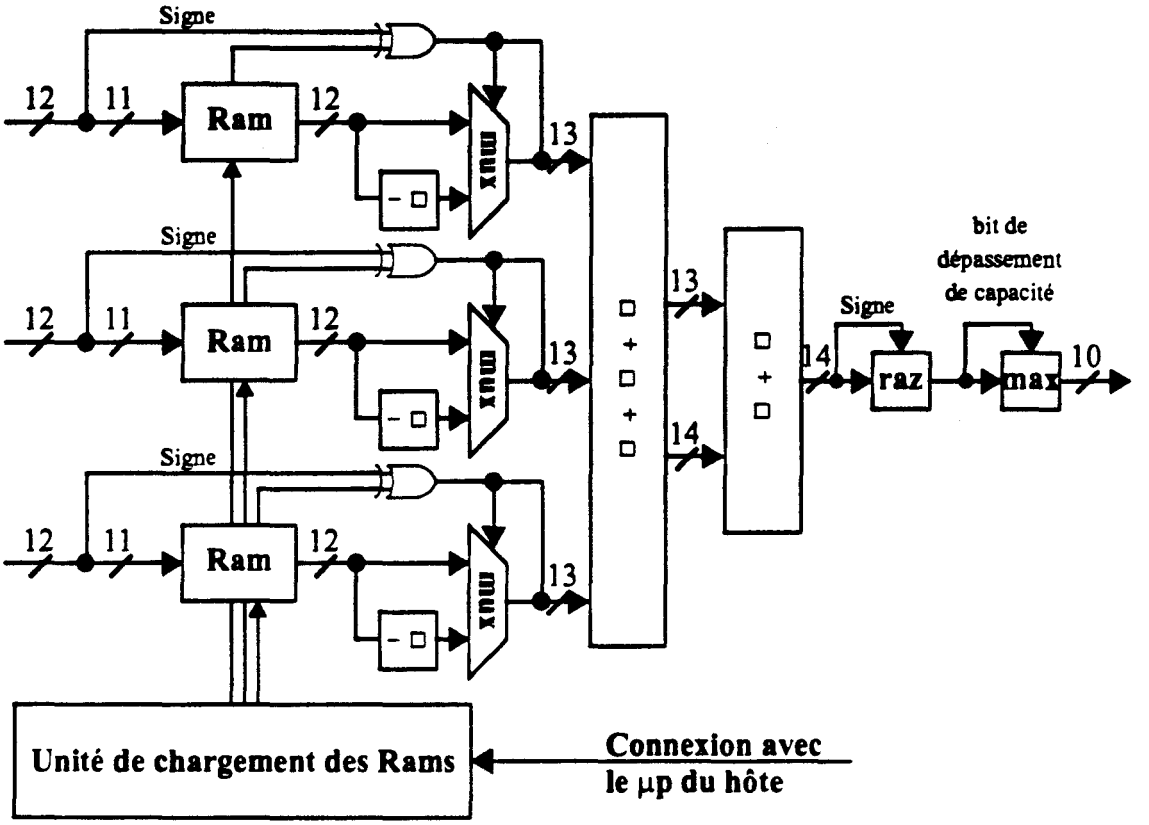


Figure 5•16 Rappel de l'architecture servant à calculer le produit scalaire avec notation binaire naturelle.

• La taille des six RAMs servant de table de multiplication pour les deux produits scalaires est fixée par la largeur du chemin de données utilisé pour faire transiter le vecteur normal. Cette taille a été fixée par l'étude théorique menée au paragraphe 4•3•3. Chaque RAM contient 2048 valeurs. Pour charger ces six tables par additions successives, il faut 12.288 cycles.

Deux solutions étaient envisageables pour effectuer ce chargement. La première consiste à effectuer le chargement durant les 600 ms du retour trame. Dans ce cas, il faut calculer une nouvelle valeur toutes les 48 ns. Il faudrait donc que l'unité de chargement fonctionne à 21 MHz. La seconde consiste à entrelacer deux bancs mémoires. On dispose ainsi de l'intégralité du temps de l'image précédente pour charger les RAMs en vue de l'image suivante. Dans le cas d'un affichage à 25 images par seconde, on dispose de 40 ms. Il faut alors calculer chaque valeur en moins de 3,26 ms. L'automate de chargement doit alors fonctionner à une fréquence de 0,3 MHz.

• Les vecteurs $\hat{L}$ et $\hat{H}$ sont normalisés par le processeur hôte avant d'être transmis à l'unité de chargement. Dans l'unité de normalisation on s'est imposé que $\|\hat{N}\| = 2047/2048$ car l'on ne voulait pas devoir ajouter un bit dans le seul but de coder la valeur 1. Pour cette même raison, on s'impose ici : $\|\hat{L}\| \times \|\hat{N}\| = \|\hat{H}\| \times \|\hat{N}\| = 1023/1024$. Il faut donc que $\|\hat{L}\| = \|\hat{H}\| = 2046/2047$.

```
RAM1[0] = val = 0
Pour i =1 à 2047 Faire
   RAM1[i] = val = val + ( Lx )
FinPour
RAM2[0] = val = 0
Pour i =1 à 2047 Faire
   RAM2[i] = val = val + ( Ly )
FinPour
RAM3[0] = val = 0
Pour i =1 à 2047 Faire
   RAM3[i] = val = val + ( Lz )
FinPour
```

```
RAM4[0] = val = 0
Pour i =1 à 2047 Faire
   RAM4[i] = val = val + ( Hx )
FinPour
RAM5[0] = val = 0
Pour i =1 à 2047 Faire
   RAM5[i] = val = val + ( Hy )
FinPour
RAM6[0] = val = 0
Pour i =1 à 2047 Faire
   RAM6[i] = val = val + ( Hz )
FinPour
```

Figure 5•17 Algorithme de chargement des RAMs pour les produits scalaires.

5•4•2 La version pour la notation S./L.N.S.

La seconde des deux architectures concerne l'utilisation de la notation S./L.N.S. L'architecture est alors la suivante :

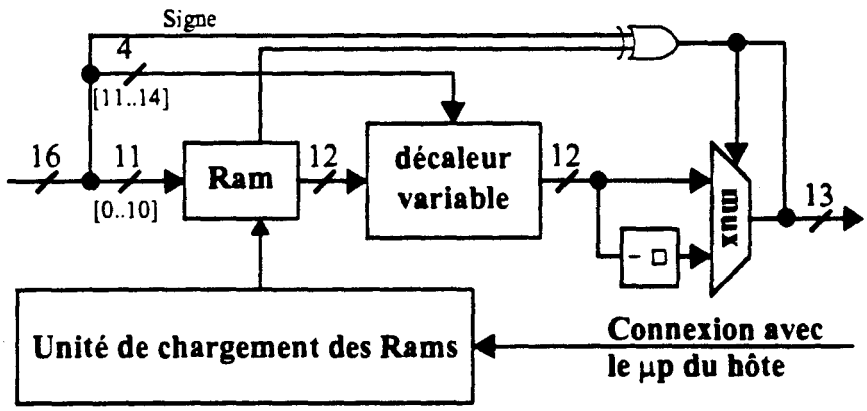


Figure 5•18 Architecture du produit scalaire en S./L.N.S.

• Pour constituer l'unité de calcul du produit scalaire, on remplace chacun des trois blocs calculant les multiplications dans la Figure 5•16, par le bloc présenté Figure 5•18. Les valeurs S./L.N.S. des composantes des vecteurs $\hat{L}$ et $\hat{H}$ sont transmises par le processeur hôte à l'unité de chargement, après qu'ils ont été normalisés. ($\|\hat{L}\| = \|\hat{H}\| = 1023/1024$)

```
j = Lx
Pour i = 0 à 2047 Faire
   Si (j< 2048) Alors
      RAM1[i] = ROM[j]
   Sinon
      RAM1[i] = ROM[ j & 2047] / 2
   finSi
   j=j+1
FinPour
j = Ly
...
...
...
...
FinPour
j = Hz
Pour i = 0 à 2047 Faire
   Si (j< 2048) Alors
      RAM6[i] = ROM[j]
   Sinon
      RAM6[i] = ROM[ j & 2047] / 2
   finSi
   j=j+1
FinPour
```

Figure 5•19 Algorithme de chargement des RAMs pour le décodage S./L.N.S. dans les produits scalaires.

• L'avantage de cette approche réside dans le coût modique des mémoires comparé à celui des unités de multiplication fonctionnant à la même fréquence, si l'on désire réaliser le processeur d'éclairement avec des composants discrets. Cette approche est possible uniquement pour l'élaboration du processeur d'éclairement avec les sources et l'observateur à l'infini. La taille de la RAM croît de façon exponentielle par rapport au nombre de bits servant à coder les composantes de la normale. Si l'on voulait augmenter la qualité et donc élargir la dynamique du vecteur normal, il faudrait recourir à de véritables unités de multiplication dont la complexité croît de façon polynomiale avec la taille des composantes de la normale.

5•4•3 Le chargement des RAMs.

• Pour effectuer le chargement de la RAM, deux solutions sont envisageables. La première consisterait à modifier le contenu de la RAM pendant le retour trame avant l'image suivante. Cela ne laisse pas énormément de temps. Pour une image 512 sur 512 affichée avec un débit de 50 images par seconde, le retour trame dure environ 600 ms. Cela ne laisse guère plus de 36 ns pour effectuer chaque calcul. La deuxième solution consiste à utiliser deux RAMs constituant un "double-buffer". Le chargement de la première RAM est effectué pendant que la deuxième est utilisée pour calculer l'intensité spéculaire, et vice versa. Le contenu de chaque RAM est modifié en alternance, une image sur deux.

Cela permet de disposer d'un laps de temps plus important pour exécuter l'algorithme de chargement de la RAM. Toujours pour une image 512 sur 512 affichée avec un débit de 50 images par seconde, on dispose, cette fois, de 20 ms. (Ce temps ne dépend pas de la résolution de l'écran mais seulement de la fréquence de rafraîchissement de l'image.) On dispose maintenant de 1.2 ms pour évaluer une case du tableau au lieu de 36 ns, soit un temps 33 fois plus important. Le choix entre ces deux possibilités dépendra à la fois du programme et de la technologie que l'on se proposera d'utiliser. Voyons tout d'abord l'algorithme.

5•4•4 Synthèse d'un V.L.S.I pour l'automate de chargement.

• Que ce soit l'unité de chargement des produits scalaires ou l'unité de chargement du diffus, il s'agit d'un automate assez simple. A chaque étape on calcule un cumul par addition d'une même valeur et on place la partie de poids fort de ce cumul dans la mémoire. La seule difficulté réside dans la fabrication de l'additionneur car il doit pouvoir travailler suffisamment rapidement. Pour résoudre ce problème, il suffit de dissocier le calcul de la partie de poids fort du calcul de la partie de poids faible, sous forme de deux additionneurs à propagation de retenue travaillant en pipeline. La retenue finale, calculée par le premier additionneur, sera transmise au second pour qu'elle soit utilisée au cycle d'horloge suivant. Ce système s'avère efficace et facile à concevoir.

Il est possible de concevoir des unités de chargement délivrant à la mémoire une nouvelle valeur, toutes les 33 ns. A cette vitesse, il faut environ 541 ms pour charger les trois RAMs du diffus en parallèle; et environ 406 ms pour charger successivement les six RAMs dans les deux unités de calcul du produit scalaire. Sachant qu'un retour trame dure 600 ms, cela constitue une solution correcte au problème posé.

• C'est l'unité de chargement du spéculaire qui s'avère être la plus compliquée. La figure ci-dessous reprend en détail la Figure 3•17 présentée au chapitre 4. Grâce au logiciel de conception de VLSI SOLO1400, une implémentation de cette architecture a été réalisée par des élèves ingénieurs E.U.D.I.L. sous ma direction.

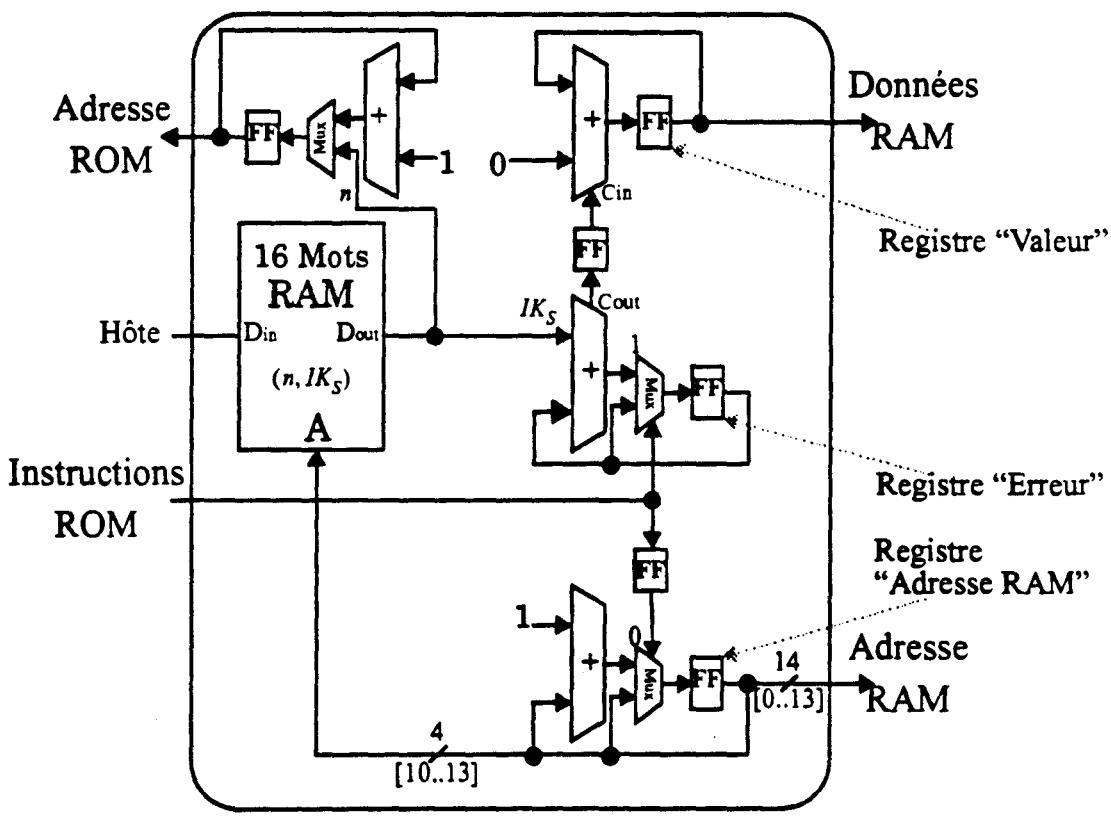


Figure 5•20 Architecture de l'unité de chargement du spéculaire.

• Il faut adjoindre dans l'architecture présentée à la Figure 5•20 la logique de contrôle qui permet d'arbitrer le chargement des coefficients dans la RAM ainsi que le déclenchement du remplissage pendant les retours trame. Au début de chaque micro-programme, c'est à dire quand on change de matière, le compteur "Adresse ROM" et les deux étages de l'additionneur ("Registre Erreur" et "Registre Valeur") sont remis à zéro.

• L'automate permet de charger la RAM pour 15 matières différentes ayant une composante spéculaire (La 16ème matière est réservée pour obtenir un fond d'écran de teinte unie. La valeur du coefficient spéculaire reste nulle et il ne convient donc pas de la recalculer à chaque retour trame.) La fréquence de fonctionnement choisie est de 33 MHz. Le chargement de la RAM prend ainsi environ 576 ms. Les résultats de simulation sont satisfaisants et le VLSI conçu mesure 13,5 mm2 sans compter la place occupée par les "pads" d'entrée-sortie et par le routage.

5•5 Bilan Global.

La réalisation d'une architecture spécialisée pour l'éclairement de Phong en temps réel n'est pas aisée tant la masse de calculs requise est importante. Nous avons montré au chapitre II que la complexité dépend directement de la nature des sources. On obtient la solution la plus simple quand celles-ci sont disposées à l'infini dans une scène visualisée en projection orthogonale.

La qualité demandée lors de la synthèse de l'image est le second point qui influe fortement sur la complexité. Lors de la réalisation matérielle d'une architecture spécialisée la précision, nécessaire pour les calculs, fixe la taille des données qui doivent être manipulées et a donc une influence directe sur le nombre de transistors de l'architecture. Au cours du chapitre IV, nous avons donc étudié ce problème et nous avons conclu qu'il était possible d'obtenir une qualité suffisante en utilisant des données de taille relativement réduite.

• Ainsi l'architecture de l'unité de normalisation d'un vecteur a pu être présentée en détail. Rappelons que cette unité trouve sa place dans la première étape du calcul de l'éclairement car il est nécessaire de normaliser le vecteur de la normale avant de le distribuer à chaque processeur d'éclairement.

Nous avons ainsi pu démontrer, simulations fonctionnelles à l'appui, qu'il était possible de réaliser l'unité de normalisation à l'aide de composants aussi courants que les ROMs (de taille très raisonnable) ou que les Réseaux logiques programmables FPGA ou EPLD. Ces derniers réalisent alors uniquement de sommaires additions tandis que les précédents ont en charge les opérations unaires. Même la complexité des multiplications a pu être réduite, la notation S./L.N.S. permettant éventuellement de les supprimer.

Nous montrons ainsi que la réalisation de l'unité de normalisation ne pose aucun problème si l'on prend soin d'ajuster convenablement le rapport entre le coût et les performances. Nous nous sommes aussi intéressés aux technologies VLSI qui peuvent également convenir. La taille d'une telle unité VLSI a été estimée, à titre d'exemple, en utilisant le compilateur de silicium SOLO1400.

• Afin de continuer notre conception d'une unité d'éclairement utilisant des sources placées à l'infini, nous devions établir l'architecture du processeur d'éclairement. Nous avons distingué principalement deux blocs distincts : d'une part celui de l'élévation à la puissance pour le calcul du spéculaire, d'autre part celui regroupant les deux produits scalaires et le calcul du diffus.

Pour la réalisation du premier bloc, plusieurs solutions ont, successivement, été proposées au chapitre III. Nous dégageons de cette étude quelques solutions parmi les plus intéressantes. Le point commun de chacune d'entre elles est l'utilisation d'une RAM dans laquelle, pendant le retour trame, sont mémorisées les 15 fonctions élévations à la puissance relatives à chacune des 15 matières.

Les différences résident dans la conception de l'automate calculant les fonctions ou dans l'adjonction de l'UAB. Celui-ci permet d'accroître la précision des calculs sans modifier la taille de la RAM, montrant ainsi que les limites de l'architecture peuvent aisément être repoussées. La première version de l'automate de chargement, qui autorise directement l'ajout de l'UAB, utilise une ROM dans laquelle sont compressées les quelques 50 (voir même 250) fonctions admises d'élévations à la puissance. La seconde version utilise des méthodes d'approximations. Cela permet d'atteindre avec une précision conséquente des exposants spéculaires élevés sans devoir recourir à une ROM de taille démesurée. Couplée à l'UAB, cette architecture trouve, bien sûr, tout son intérêt.

Le second bloc se conçoit assez bien, tout le calcul de l'intensité diffuse ne requiert jamais qu'une précision très limitée. Toutes les multiplications le composant peuvent donc être remplacées par des RAMs contenant les tables de multiplication utilisées. Par contre, la réduction de la complexité du second produit scalaire est plus incertaine. Tant que l'exposant spéculaire reste faible (inférieur à 40 ou 45), ce produit scalaire peut être identique au précédent. Cela nous permet de présenter une proposition complète d'une architecture spécialisée d'entrée de gamme pour l'éclairement de Phong en temps réel dont le coût sera très réduit [35].

- Par contre, si l'on veut utiliser des exposants spéculaires élevés, il est nécessaire d'augmenter la précision pendant la normalisation du vecteur de la normale et le calcul de ce dernier produit scalaire. Dans ce cas, il conviendra donc d'établir une architecture plus adéquate pour l'unité de normalisation des vecteurs. L'étude, présentée au chapitre IV, continue d'indiquer dans quelle mesure la précision doit être accrue au cours de la normalisation. L'utilisation d'une ROM pour calculer l'inverse de l'extraction de la racine carrée devra être remise en cause. Il faudra trouver d'autres méthodes permettant d'obtenir le résultat.

La décomposition du calcul en deux opérateurs séparés pourrait s'avérer utile. Il est possible de calculer la racine carrée d'un nombre en utilisant un réseau qui peut facilement être constitué en pipeline [25] [32]. Il suffirait simplement de construire une unité de division ou d'inversion sous la forme d'un pipeline suffisamment rapide pour fonctionner en temps réel. Malheureusement, cela aura pour conséquence d'accroître le coût dans des proportions très importantes [31]. Il serait souhaitable de vérifier si des calculs locaux en notation redondante ne permettraient pas d'alléger la conception d'une telle opération en diminuant notamment le nombre des étages du pipeline.

L'utilisation de processeurs généraux disposant d'instructions rapides pour calculer l'extraction de la racine carrée, l'inversion ou la division, pourrait offrir une solution intéressante. Le DSP TMS320C40 arriverait en bonne place lors du choix d'un tel microprocesseur mais il conviendrait d'examiner également l'ALPHA, le POWER-PC, le R4000 et quelques autres. Une matrice de tels processeurs devraient développer la puissance requise.

- Par la suite, des processeurs d'éclairement opérant pour des sources d'un autre type (situées à distance finie dans une scène visualisée en projection perspective par exemple) devraient être étudiés. A cet effet, le chapitre II nous donne les indications utiles quant à la formule d'éclairement s'adaptant le mieux à la situation et quant au schéma synoptique du processeur. L'architecture calculant l'élévation à la puissance restera valable. Par contre, il conviendra de modifier le reste du processeur. Il faudra utiliser de véritables produits scalaires et ajouter un bloc pour le calcul du vecteur source.

Conclusion.

Après une brève introduction de l'infographie et des diverses techniques d'éclairement et d'ombrage, nous avons étudié différentes architectures en vue de la réalisation d'une unité calculant l'éclairement de Phong en temps réel. Nous avons rapidement montré que cela réclame une puissance de calcul importante. Les différentes formules d'éclairement ont successivement été étudiées afin de déterminer la meilleure optimisation. La complexité de chacune d'entre elles a été examinée selon la position de la source et de l'observateur, pour une ou plusieurs sources.

Par la suite, nous avons restreint notre étude à la conception d'une unité d'éclairement pour des sources de lumière situées à l'infini dans une scène visualisée en projection perspective. Lors de la conception, deux points ont plus particulièrement attiré notre attention du fait de leur complexité plus importante. Il s'agit de la normalisation et de l'élévation à la puissance. La nécessité de la première a tout d'abord été montrée quand on modélise la scène avec des facettes puis avec des quadriques. Ensuite, afin d'optimiser la complexité de ce module, nous avons déterminé les tailles utiles des chemins données grâce à une étude théorique sur la transmission de l'erreur. Cette étude nous a, de plus, permis d'introduire l'unité de pré-normalisation en indiquant son utilité.

Pour résoudre le second point, plusieurs solutions ont été proposées dont l'utilisation d'approximations. Les architectures correspondantes ont été présentées aboutissant ainsi à une solution finale permettant d'obtenir un rendu du spéculaire très proche de celui proposé par Phong pour un prix très abordable. Enfin nous avons présenté en détail le reste de la version restreinte du processeur d'éclairement.

Il faudrait maintenant, outre réaliser la présente proposition, étudier l'architecture du processeur d'éclairement disposant de sources à distance finie dans une scène visualisée en projection perspective. Une partie de ce travail a déjà été commencée au chapitre II, la formule d'éclairement la plus appropriée et l'algorithme correspondant ont été déterminés. Mais il faut encore présenter l'architecture et effectuer une étude théorique de la taille des chemins de données, semblable à celle effectuée au chapitre IV pour les sources à l'infini.

La réalisation requerra un nombre plus important de divisions et d'extractions de la racine carrée. Aussi la recherche de nouvelles techniques permettant de calculer rapidement ces opérations serait un point important pour la réalisation d'un tel processeur d'éclairement. Les calculs en notation redonnante, dont nous avons déjà su tirer avantage pour calculer les produits scalaires, devraient fournir des solutions attrayantes.

L'examen de l'élévation à la puissance a finalement abouti à plusieurs solutions technologiques intéressantes permettant d'obtenir pour un coût raisonnable des exposants spéculaires importants. Plus cet exposant croît, plus la taille de la tache spéculaire diminue et peut approcher celle d'un pixel. Afin de continuer à améliorer la qualité de l'image, il conviendrait donc d'étudier différentes techniques simples pour antialiasser l'éclairement spéculaire. Enfin, il faudrait étudier l'ajout d'un système de mappage de texture qui soit compatible avec les solutions proposées.

ANNEXE A: L'optimisation des algorithmes d'éclairement.

A•1 Légende.

A•1•1 Les fonctions.

Les fonctions utilisées dans cette annexe pour décrire les algorithmes sont les suivantes :

SQ (_valeur_)	donne $(\_valeur\_)^2$
SQRT (_valeur_)	donne $\sqrt{\_valeur\_}$
DIV (_valeur1_ , _valeur2_)	donne $\frac{\_valeur1\_}{\_valeur2\_}$
POWER (_valeur1_ , _valeur2_)	donne $(\_valeur1\_)^{-valeur2\_}$
SHIFT (_valeur1_ , _valeur2_)	donne $(\_valeur1\_) \times 2^{-valeur2\_}$

A•1•2 Les variables.

Les noms des variables utilisées comme données d'entrées dans les algorithmes de cette annexe sont les suivants :

I_rouge	est l'intensité de la composante rouge de la source.
I_vert	est l'intensité de la composante verte de la source.
I_bleu	est l'intensité de la composante bleue de la source.
Kd_rouge	est le coefficient diffus pour la composante rouge.
Kd_vert	est le coefficient diffus pour la composante verte.
Kd_bleu	est le coefficient diffus pour la composante bleue.
Ks_rouge	est le coefficient spéculaire pour la composante rouge.
Ks_vert	est le coefficient spéculaire pour la composante verte.
Ks_bleu	est le coefficient spéculaire pour la composante bleue.
E	est l'exposant spéculaire.
nx, ny, nz	sont les composantes du vecteur $\hat{N}$ non normé.

Quand la source est à distance finie, on a en plus :

Sx, Sy, Sz	les coordonnées de la position de la source.

Quand l'observateur est à distance finie, on a aussi en plus :

Xo, Yo, Zo	les coordonnées de la position de l'observateur.

Quand la source est à l'infini, les variables suivantes sont utilisées pour les données d'entrée. (Dans le cas contraire, ce sont des variables intermédiaires)

lx, ly, lz	les composantes du vecteur $\vec{L}$ non normé.
hx, hy, hz	les composantes du vecteur $\vec{H}$ non normé.

Les noms des variables utilisées pour des données intermédiaires dans les algorithmes de cette annexe sont les suivants :

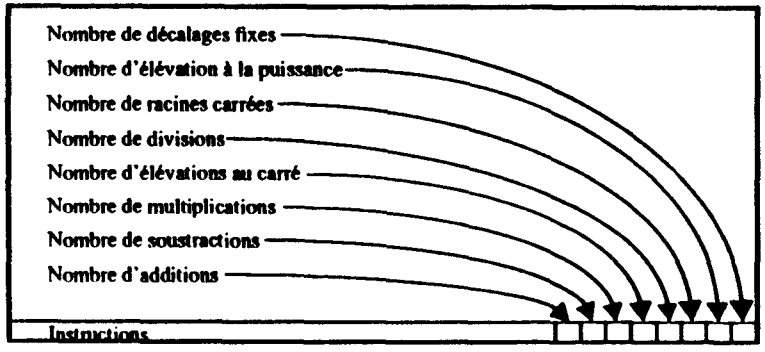
rx, ry, rz	les composantes du vecteur $\vec{R}$ non normé.
qx, qy, qz	les composantes du vecteur $\vec{Q}$ non normé.
ox, oy, oz	les composantes du vecteur $\vec{O}$ non normé.
Nx, Ny, Nz	les composantes du vecteur $\vec{N}$ normé.
Lx, Ly, Lz	les composantes du vecteur $\vec{L}$ normé.
Hx, Hy, Hz	les composantes du vecteur $\vec{H}$ normé.
Rx, Ry, Rz	les composantes du vecteur $\vec{R}$ normé
Qx, Qy, Qz	les composantes du vecteur $\vec{Q}$ normé.
Ox, Oy, Oz	les composantes du vecteur $\vec{O}$ normé.
NormeN2	le carré de la norme du vecteur $\vec{N}$.
NormeN	la norme du vecteur $\vec{N}$.
NormeL2	le carré de la norme du vecteur $\vec{L}$.
NormeL	la norme du vecteur $\vec{L}$.
NormeH2	le carré de la norme du vecteur $\vec{H}$.
NormeH	la norme du vecteur $\vec{H}$.
NormeO2	le carré de la norme du vecteur $\vec{O}$.
NormeH	la norme du vecteur $\vec{O}$.
MDR , MDV, MDB, MSR, MSV, MSB	les produits des coefficients et des intensités.
PS1, PS2	des résultats de produits scalaires.
P	le résulat du produit scalaire, pour l'illumination, spéculaire élévé à la puissance E.

Les noms des variables utilisées comme données de sorties dans les algorithmes de cette annexe sont les suivants :

IR	l'intensité résultante pour la composante rouge.
IV	l'intensité résultante pour la composante verte.
IB	l'intensité résultante pour la composante bleue.

A•1•3 Les colonnes.

Dans les tableaux décrivant les algorithmes, on trouve sur la gauche les instructions du programme et sur la droite, disposé dans 8 colonnes suivant leur nature, le nombre d'instructions.



A•2 Scène en projection orthogonale.

A•2•1 Une source placée à l'infini.

Par image :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
Lx ← DIV(lx , NormeL)					1			
Ly ← DIV(ly , NormeL)					1			
Lz ← DIV(lz , NormeL)					1			
hz ← Lz + 1	1							
NormeH2 ← SQ(Lx) + SQ(Ly) + SQ(hz)	2			3				
NormeH ← SQRT(NormeH2)						1		
Hx ← DIV(Lx , NormeH)					1			
Hy ← DIV(Ly , NormeH)					1			
Hz ← DIV(hz , NormeH)					1			
TOTAL	5		1	6	6	2		1

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
PS1 ← DIV((Lx*nx) +(Ly*ny) + (Lz*nz) , NormeN)	2		3		1			
PS2 ← DIV((Hx*nx) +(Hy*ny) + (Hz*nz)) , NormeN)	2		3		1			
P ← POWER(PS2 , E)							1	
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	9		12	3	2	1	1	

A•2•2 Plusieurs sources placées à l'infini.

Les modifications par rapport à l'algorithme précédent pour une source sont indiquées en italique.

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
Nx ← DIV(nx , NormeN)						1		
Ny ← DIV(ny , NormeN)						1		
Nz ← DIV(nz , NormeN)						1		
TOTAL	2			3	3	1		

Par image et par source :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
Lx ← DIV(lx , NormeL)						1		
Ly ← DIV(ly , NormeL)						1		
Lz ← DIV(lz , NormeL)						1		
hz ← Lz + 1	1							
NormeH2 ← SQ(Lx) + SQ(Ly) + SQ(hz)	2			3				
NormeH ← SQRT(NormeH2)						1		
Hx ← DIV(Lx , NormeH)						1		
Hy ← DIV(Ly , NormeH)						1		
Hz ← DIV(hz , NormeH)						1		
TOTAL	5			6	6	2		

Par image, par matière et par source :								
MDR ← L_rouge * Kd_rouge				1				
MDV ← L_vert * Kd_vert				1				
MDB ← L_bleu * Kd_bleu				1				
MSR ← L_rouge * Ks_rouge				1				
MSV ← L_vert * Ks_vert				1				
MSB ← L_bleu * Ks_bleu				1				
TOTAL				6				

Par pixel et par source :								
*PS1 ← (Lx*Nx) +(Ly*Ny) + (Lz*Nz)*	2			3				
*PS2 ← (Hx*Nx) +(Hy*Ny) + (Hz*Nz)*	2			3				
P ← POWER(PS2 , E)							1	
*IR ← IR + (MDR * PS1) + (MSR * P)*	2			2				
*IV ← IV + (MDV * PS1) + (MSV * P)*	2			2				
*IB ← IB + (MDB * PS1) + (MSB * P)*	2			2				
TOTAL	10			12			1	

A•2•3 Une source placée à distance finie.

A•2•3•1 Première méthode.

Par image et par matière :								
MDR ← L_rouge * Kd_rouge			1					
MDV ← L_vert * Kd_vert			1					
MDB ← L_bleu * Kd_bleu			1					
MSR ← L_rouge * Ks_rouge			1					
MSV ← L_vert * Ks_vert			1					
MSB ← L_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
lx ← Sx - x		1						
ly ← Sy - y		1						
lz ← Sz - z		1						
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
PS1 ← (lx*nx) + (ly*ny) + (lz*nz)	2		3					
rz ← SHIFT(DIV(nz*PS1 , NormeN2) , +1) - lz		1	1		1			
PS1 ← DIV(PS1 , NormeL*NormeN)			1		1			
PS2 ← DIV(rz , NormeL)					1			
P ← POWER(PS2 , E)						1		
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	9	4	11	6	3	2	1	1

A•2•3•2 Seconde méthode.

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
lx ← Sx - x		1						
ly ← Sy - y		1						
lz ← Sz - z		1						
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
Lx ← DIV(lx , NormeL)					1			
Ly ← DIV(ly , NormeL)					1			
Lz ← DIV(lz , NormeL)					1			
hz ← Lz + 1	1							
NormeH2 ← 2 + SHIFT (Lz , +1)	1							1
NormeH ← SQRT(NormeH2)						1		
PS2 ← (Lx*nx) +(Ly*ny)	1		2					
PS1 ← DIV(PS2 + (Lz*nz) , NormeN)	1		1		1			
PS2 ← DIV(PS2 + (hz*nz) , NormeN*NormeH)	1		2		1			
P ← POWER(PS2 , E)							1	
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	12	3	11	6	5	3	1	1

A•2•4 Plusieurs sources placées à distance finie.

A•2•4•1 Première méthode.

Les modifications par rapport à l'algorithme du paragraphe A•2•3•1 sont indiquées en italique.

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
Nx ← DIV(nx , NormeN)					1			
Ny ← DIV(ny , NormeN)					1			
Nz ← DIV(nz , NormeN)					1			
TOTAL	2			3	3	1		

Par image, par matière et par source :								
MDR ← I_rouge * Kd_rouge				1				
MDV ← I_vert * Kd_vert				1				
MDB ← I_bleu * Kd_bleu				1				
MSR ← I_rouge * Ks_rouge				1				
MSV ← I_vert * Ks_vert				1				
MSB ← I_bleu * Ks_bleu				1				
TOTAL				6				

Par pixel et par source :								
lx ← Sx - x		1						
ly ← Sy - y		1						
lz ← Sz - z		1						
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
*PS1 ← (lx*Nx) + (ly*Ny) + (lz*Nz)*	2		3					
*rz ← SHIFT((Nz*PS1) , +1) - lz*		1	1					1
PS1 ← DIV(PS1 , NormeL)				1				
PS2 ← DIV(rz , NormeL)				1				
P ← POWER(PS2 , E)							1	
*IR ← IR + (MDR * PS1) + (MSR * P)*	2		2					
*IV ← IV + (MDV * PS1) + (MSV * P)*	2		2					
*IB ← IB + (MDB * PS1) + (MSB * P)*	2		2					
TOTAL	10	4	10	3	2	1	1	1

A•2•4•2 Seconde méthode.

Les modifications par rapport à l'algorithme du paragraphe A•2•3•2 sont indiquées en italique.

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
Nx ← DIV(nx , NormeN)					1			
Ny ← DIV(ny , NormeN)					1			
Nz ← DIV(nz , NormeN)					1			
TOTAL	2			3	3	1		

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel et par source :									
Ix ← Sx - x		1							
Iy ← Sy - y		1							
Iz ← Sz - z		1							
NormeL2 ← SQ(Ix) + SQ(Iy) + SQ(Iz)	2			3					
NormeL ← SQRT(NormeL2)						1			
Lz ← DIV(Iz , NormeL)					1				
Iz ← Iz + NormeL	1								
NormeH2 ← 2 + SHIFT (Lz , +1)	1								1
NormeH ← SQRT(NormeH2)						1			
*PS2 ← (Ix*Nx) + (Iy*Ny)*	1	2							
*PS1 ← DIV(PS2 + (Iz*Nz) , NormeL)*	1	1			1				
*PS2 ← DIV(PS2 + (Iz*Nz) , NormeH*NormeL)*	1	2			1				
P ← POWER(PS2 , E)								1	
*IR ← IR + (MDR * PS1) + (MSR * P)*	2	2							
*IV ← IV + (MDV * PS1) + (MSV * P)*	2	2							
*IB ← IB + (MDB * PS1) + (MSB * P)*	2	2							
TOTAL	13	13	1	3	3	2	1	1	

A•2•4•3 Troisième méthode.

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
Nx ← DIV(nx , NormeN)					1			
Ny ← DIV(ny , NormeN)					1			
Nz ← DIV(nz , NormeN)					1			
Qx ← SHIFT((Nz*Nx) , +1)			1					1
Qy ← SHIFT((Nz*Ny) , +1)			1					1
Qz ← SHIFT(SQ(Nz) , +1) - 1		1		1				1
TOTAL	2	1	2	4	3	1	1	3

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel et par source :								
Ix ← Sx - x		1						
Iy ← Sy - y		1						
Iz ← Sz - z		1						
NormeL2 ← SQ(Ix) + SQ(Iy) + SQ(Iz)	2			3				
NormeL ← SQRT(NormeL2)						1		
PS1 ← DIV((Ix*Nx) + (Iy*Ny) + (Iz*Nz) , NormeL)	2		3		1			
PS2 ← DIV((Ix*Qx) + (Iy*Qy) + (Iz*Qz) , NormeL)	2		3		1			
P ← POWER(PS2 , E)							1	
IR ← IR + (MDR * PS1) + (MSR * P)	2		2					
IV ← IV + (MDV * PS1) + (MSV * P)	2		2					
IB ← IB + (MDB * PS1) + (MSB * P)	2		2					
TOTAL	12	3	12	3	3	1	1	

A•3 Scène en projection perspective.

A•3•1 Une source placée à l'infini.

A•3•1•1 Première méthode.

Par image :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)					1			
Lx ← DIV(lx , NormeL)					1			
Ly ← DIV(ly , NormeL)					1			
Lz ← DIV(lz , NormeL)					1			
TOTAL	2			3	3	1		

Par image et par matière :								
MDR ← l_rouge * Kd_rouge			1					
MDV ← l_vert * Kd_vert			1					
MDB ← l_bleu * Kd_bleu			1					
MSR ← l_rouge * Ks_rouge			1					
MSV ← l_vert * Ks_vert			1					
MSB ← l_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)					1			
PS1 ← (Lx*nx) + (Ly*ny) + (Lz*nz)	2		3					
PS2 ← SHIFT(DIV(PS1 , NormeN2) , +1)				1				1
PS1 ← DIV(PS1 , NormeN)				1				
Rx ← (PS2*nx) - Lx		1	1					
Ry ← (PS2*ny) - Ly		1	1					
Rz ← (PS2*nz) - Lz		1	1					
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)						1		
PS2 ← DIV((Rx*ox) + (Ry*oy) + (Rz*Zo) , NormeO)	2		3		1			
P ← POWER(PS2 , E)								1
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	11	5	15	3	3	2	1	1

A•3•1•2 Seconde méthode.

Par image :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)					1			
Lx ← DIV(lx , NormeL)					1			
Ly ← DIV(ly , NormeL)					1			
Lz ← DIV(lz , NormeL)					1			
TOTAL	2			3	3	1		

Par image et par matière :								
MDR ← l_rouge * Kd_rouge				1				
MDV ← l_vert * Kd_vert				1				
MDB ← l_bleu * Kd_bleu				1				
MSR ← l_rouge * Ks_rouge				1				
MSV ← l_vert * Ks_vert				1				
MSB ← l_bleu * Ks_bleu				1				
TOTAL				6				

Par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)					1			
PS1 ← (Lx*nx) + (Ly*ny) + (Lz*nz)	2		3					
PS1 ← DIV(PS1 , NormeN)				1				
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)						1		
hx ← (NormeO*Lx) + ox	1		1					
hy ← (NormeO*Ly) + oy	1		1					
hz ← (NormeO*Lz) + Zo	1		1					
NormeH2 ← SQ(hx) + SQ(hy) + SQ(hz)	2			3				
NormeH ← SQRT(NormeH2)					1			
PS2 ← DIV((hx*nx) +(hy*ny) + (hz*nz)) , (NormeN*NormeH))	2		4		1			
P ← POWER(PS2 , E)							1	
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	16	2	16	6	2	3	1	

A•3•1•3 Troisième méthode.

Par image :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3		1		
NormeL ← SQRT(NormeL2)								
Lx ← DIV(lx , NormeL)								
Ly ← DIV(ly , NormeL)								
Lz ← DIV(lz , NormeL)								
TOTAL	2			3	3	1		

Par image et par matière :								
MDR ← L_rouge * Kd_rouge			1					
MDV ← L_vert * Kd_vert			1					
MDB ← L_bleu * Kd_bleu			1					
MSR ← L_rouge * Ks_rouge			1					
MSV ← L_vert * Ks_vert			1					
MSB ← L_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3		1		
NormeN ← SQRT(NormeN2)								
PS1 ← (Lx*nx) + (Ly*ny) + (Lz*nz)	2		3					
PS1 ← DIV(PS1 , NormeN)						1		
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)						1		
hx ← (NormeO*Lx) + ox	1		1					
hy ← (NormeO*Ly) + oy	1		1					
hz ← (NormeO*Lz) + Zo	1		1					
NormeH ← NormeO + DDA(x,y)	2							
PS2 ← DIV((hx*nx) +(hy*ny) + (hz*nz)) , (NormeN*NormeH))	2		4			1		
P ← POWER(PS2 , E)								1
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	16	2	16	3	2	2	1	

A•3•1•4 Quatrième méthode.

Par image :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3		1		
NormeL ← SQRT(NormeL2)								
Lx ← DIV(lx , NormeL)								
Ly ← DIV(ly , NormeL)								
Lz ← DIV(lz , NormeL)								
TOTAL	2			3	3	1		

Par image et par matière :								
MDR ← L_rouge * Kd_rouge			1					
MDV ← L_vert * Kd_vert			1					
MDB ← L_bleu * Kd_bleu			1					
MSR ← L_rouge * Ks_rouge			1					
MSV ← L_vert * Ks_vert			1					
MSB ← L_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3		1		
NormeN ← SQRT(NormeN2)								
PS1 ← (Lx*nx) + (Ly*ny) + (Lz*nz)	2		3					
PS1 ← DIV(PS1 , NormeN)						1		
ox ← Xo - x		1						
oy ← Yo - y		1						
PS2 ← DIV((ox*nx) + (oy*ny) + (Zo*nz) , NormeN2)	2		3			1		
PS2 ← SHIFT(PS2 , +1)								1
qx ← (PS2 *nx) - ox		1	1					
qy ← (PS2*ny) - oy		1	1					
qz ← (PS2*nz) - Zo		1	1					
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)						1		
PS2 ← DIV((qx*Lx) + (qy*Ly) + (qz*Lz) , NormeO)	2		3			1		
P ← POWER(PS2 , E)								1
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	13	5	18	3	3	2	1	1

A•3•2 Plusieurs sources placées à l'infini.

A•3•2•1 Première méthode.

Les modifications par rapport à l'algorithme du paragraphe A•3•1•1 sont indiquées en italique.

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
Nx ← DIV(nx , NormeN)					1			
Ny ← DIV(ny , NormeN)					1			
Nz ← DIV(nz , NormeN)					1			
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)						1		
Ox ← DIV(ox , NormeO)					1			
Oy ← DIV(oy , NormeO)					1			
Oz ← DIV(oz , NormeO)					1			
TOTAL	4	2		3	6	2		

Par image et par source :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
Lx ← DIV(lx , NormeL)					1			
Ly ← DIV(ly , NormeL)					1			
Lz ← DIV(lz , NormeL)					1			
TOTAL	2			3	3	1		

Par image, par matière et par source :								
MDR ← l_rouge * Kd_rouge				1				
MDV ← l_vert * Kd_vert				1				
MDB ← l_bleu * Kd_bleu				1				
MSR ← l_rouge * Ks_rouge				1				
MSV ← l_vert * Ks_vert				1				
MSB ← l_bleu * Ks_bleu				1				
TOTAL				6				

Par pixel et par source :								
*PS1 ← (Lx*Nx) + (Ly*Ny) + (Lz*Nz)*	2			3				
PS2 ← SHIFT(PS1, +1)								1
*Rx ← (PS2*Nx) - Lx*			1	1				
*Ry ← (PS2*Ny) - Ly*			1	1				
*Rz ← (PS2*Nz) - Lz*			1	1				
*PS2 ← (Rx*Ox) + (Ry*Oy) + (Rz*Oz)*	2			3				
P ← POWER(PS2 , E)							1	
*IR ← IR + (MDR * PS1) + (MSR * P)*	2		2					
*IV ← IV + (MDV * PS1) + (MSV * P)*	2		2					
*IB ← IB + (MDB * PS1) + (MSB * P)*	2		2					
TOTAL	10	3	15				1	1

A•3•2•2 Seconde méthode.

Les modifications par rapport à l'algorithme du paragraphe A•3•1•2 sont indiquées en italique.

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
Nx ← DIV(nx , NormeN)					1			
Ny ← DIV(ny , NormeN)					1			
Nz ← DIV(nz , NormeN)					1			
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)						1		
Ox ← DIV(ox , NormeO)					1			
Oy ← DIV(oy , NormeO)					1			
Oz ← DIV(oz , NormeO)					1			
TOTAL	4	2		3	6	2		

Par image et par source :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
Lx ← DIV(lx , NormeL)					1			
Ly ← DIV(ly , NormeL)					1			
Lz ← DIV(lz , NormeL)					1			
TOTAL	2			3	3	1		

Par image, par matière et par source :								
MDR ← l_rouge * Kd_rouge				1				
MDV ← l_vert * Kd_vert				1				
MDB ← l_bleu * Kd_bleu				1				
MSR ← l_rouge * Ks_rouge				1				
MSV ← l_vert * Ks_vert				1				
MSB ← l_bleu * Ks_bleu				1				
TOTAL				6				

Par pixel et par source :								
*PS1 ← (Lx*Nx) + (Ly*Ny) + (Lz*Nz)*	2			3				
hx ← Lx + Ox		1						
hy ← Ly + Oy		1						
hz ← Lz + Oz		1						
NormeH2 ← SQ(hx) + SQ(hy) + SQ(hz)	2			3				
NormeH ← SQRT(NormeH2)						1		
*PS2 ← DIV((hx*Nx) + (hy*Ny) + (hz*Nz) , NormeH)*	2			3		1		
P ← POWER(PS2 , E)							1	
*IR ← IR + (MDR * PS1) + (MSR * P)*	2		2					
*IV ← IV + (MDV * PS1) + (MSV * P)*	2		2					
*IB ← IB + (MDB * PS1) + (MSB * P)*	2		2					
TOTAL	15		12	3	1	1	1	

A•3•2•3 Troisième méthode.

Les modifications par rapport à l'algorithme du paragraphe A•3•1•3 sont indiquées en italique.

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)								
Nx ← DIV(nx , NormeN)								
Ny ← DIV(ny , NormeN)								
Nz ← DIV(nz , NormeN)								
ox ← Xo - x								
oy ← Yo - y								
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)								
Ox ← DIV(ox , NormeO)								
Oy ← DIV(oy , NormeO)								
Oz ← DIV(oz , NormeO)								
TOTAL	4	2		3	6	2		

Par image et par source :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)								
Lx ← DIV(lx , NormeL)								
Ly ← DIV(ly , NormeL)								
Lz ← DIV(lz , NormeL)								
TOTAL	2			3	3			

Par image, par matière et par source :								
MDR ← l_rouge * Kd_rouge								
MDV ← l_vert * Kd_vert								
MDB ← l_bleu * Kd_bleu								
MSR ← l_rouge * Ks_rouge								
MSV ← l_vert * Ks_vert								
MSB ← l_bleu * Ks_bleu								
TOTAL				6				

Par pixel et par source :								
*PS1 ← (Lx*Nx) + (Ly*Ny) + (Lz*Nz)*	2			3				
hx ← Lx + Ox	1							
hy ← Ly + Oy	1							
hz ← Lz + Oz	1							
*NormeH ← SHIFT(1 + (Ox*Lx) + (Oy*Ly) + (Oz*Lz) , +1)*	3			3				
*PS2 ← DIV((hx*Nx) + (hy*Ny) + (hz*Nz) , NormeH)*	2			3				
P ← POWER(PS2 , E)								
*IR ← IR + (MDR * PS1) + (MSR * P)*	2			2				
*IV ← IV + (MDV * PS1) + (MSV * P)*	2			2				
*IB ← IB + (MDB * PS1) + (MSB * P)*	2			2				
TOTAL	16			19				

A•3•2•4 Quatrième méthode.

Les modifications par rapport à l'algorithme du paragraphe A•3•1•4 sont indiquées en italique.

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)								
Nx ← DIV(nx , NormeN)								
Ny ← DIV(ny , NormeN)								
Nz ← DIV(nz , NormeN)								
ox ← Xo - x								
oy ← Yo - y								
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)								
Ox ← DIV(ox , NormeO)								
Oy ← DIV(oy , NormeO)								
Oz ← DIV(oz , NormeO)								
*PS1 ← (Ox*Nx) + (Oy*Ny) + (Oz*Nz)*	2		3					
*Qx ← SHIFT((PS1*Nx) , +1) - Ox*								
*Qy ← SHIFT((PS1*Ny) , +1) - Oy*								
*Qz ← SHIFT((PS1*Nz) , +1) - Oz*								
TOTAL	6	5	6	3	6	2		3

Par image et par source :								
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)								
Lx ← DIV(lx , NormeL)								
Ly ← DIV(ly , NormeL)								
Lz ← DIV(lz , NormeL)								
TOTAL	2			3	3			

Par image, par matière et par source :								
MDR ← l_rouge * Kd_rouge								
MDV ← l_vert * Kd_vert								
MDB ← l_bleu * Kd_bleu								
MSR ← l_rouge * Ks_rouge								
MSV ← l_vert * Ks_vert								
MSB ← l_bleu * Ks_bleu								
TOTAL				6				

Par pixel et par source :								
*PS1 ← (Lx*Nx) + (Ly*Ny) + (Lz*Nz)*	2			3				
*PS2 ← (Lx*Qx) + (Ly*Qy) + (Lz*Qz)*	2			3				
P ← POWER(PS2 , E)								
*IR ← IR + (MDR * PS1) + (MSR * P)*	2			2				
*IV ← IV + (MDV * PS1) + (MSV * P)*	2			2				
*IB ← IB + (MDB * PS1) + (MSB * P)*	2			2				
TOTAL	10			12				

A•3•3 Une source placée à distance finie.

A•3•3•1 Première méthode.

Par image et par matière :							
MDR ← I_rouge * Kd_rouge			1				
MDV ← I_vert * Kd_vert			1				
MDB ← I_bleu * Kd_bleu			1				
MSR ← I_rouge * Ks_rouge			1				
MSV ← I_vert * Ks_vert			1				
MSB ← I_bleu * Ks_bleu			1				
TOTAL			6				

Par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)					1			
[lx , ly , lz] ← PSEUDO_DDA(x,y,z)	3	1	3					
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)					1			
PS1 ← (lx*nx) +(ly*ny) + (lz*nz)	2		3					
PS2 ← SHIFT(DIV(PS1 , NormeN2) , +1)					1			1
PS1 ← DIV(PS1 , NormeL*NormeN)			1		1			
rx ← (PS2*nx) - lx		1	1					
ry ← (PS2*ny) - ly		1	1					
rz ← (PS2*nz) - lz		1	1					
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)					1			
PS2 ← DIV((rx*ox) + (ry*oy) + (rz*Zo) , NormeL*NormeO)	2		4		1			1
P ← POWER(PS2 , E)								
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	16	6	20	6	3	3	1	1

A•3•3•2 Seconde méthode.

Par image et par matière :							
MDR ← I_rouge * Kd_rouge			1				
MDV ← I_vert * Kd_vert			1				
MDB ← I_bleu * Kd_bleu			1				
MSR ← I_rouge * Ks_rouge			1				
MSV ← I_vert * Ks_vert			1				
MSB ← I_bleu * Ks_bleu			1				
TOTAL			6				

Par pixel :							
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2		3				
NormeN ← SQRT(NormeN2)				1			
[lx , ly , lz] ← PSEUDO_DDA(x,y,z)	3	1	3				
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2		3				
NormeL ← SQRT(NormeL2)				1			
Lx ← DIV(lx , NormeL)				1			
Ly ← DIV(ly , NormeL)				1			
Lz ← DIV(lz , NormeL)				1			
PS1 ← DIV((Lx*nx) +(Ly*ny) + (Lz*nz) , NormeN)	2		3		1		
ox ← Xo - x		1					
oy ← Yo - y		1					
NormeO2 ← DDA(x,y)	2						
NormeO ← SQRT(NormeO)					1		
hx ← (NormeO*Lx) + ox	1		1				
hy ← (NormeO*Ly) + oy	1		1				
hz ← (NormeO*Lz) + Zo	1		1				
NormeH2 ← SQ(hx) + SQ(hy) + SQ(hz)	2		3				
NormeH ← SQRT(NormeH2)				1			
PS2 ← DIV((hx*nx) +(hy*ny) + (hz*nz)) , (NormeN*NormeH))	2		4		1		1
P ← POWER(PS2 , E)							
IR ← (MDR * PS1) + (MSR * P)	1		2				
IV ← (MDV * PS1) + (MSV * P)	1		2				
IB ← (MDB * PS1) + (MSB * P)	1		2				
TOTAL	21	3	18	9	5	4	1

A•3•3•3 Troisième méthode.

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
[lx , ly , lz] ← PSEUDO_DDA(x,y,z)	3	1	3					
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
Lx ← DIV(lx , NormeL)					1			
Ly ← DIV(ly , NormeL)					1			
Lz ← DIV(lz , NormeL)					1			
PS1 ← DIV((Lx*nx) +(Ly*ny) + (Lz*nz) , NormeN)	2			3		1		
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)						1		
hx ← (NormeO*Lx) + ox	1	1						
hy ← (NormeO*Ly) + oy	1	1						
hz ← (NormeO*Lz) + Zo	1	1						
NormeH ← NormeO + DDA(x,y)	2							
PS2 ← DIV((hx*nx) +(hy*ny) + (hz*nz) , (NormeN*NormeH))	2		4			1		
P ← POWER(PS2 , E)								1
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	21	6	16	6	3	3	1	1

A•3•3•4 Quatrième méthode.

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Par pixel :								
[lx , ly , lz] ← PSEUDO_DDA(x,y,z)	3	1	3					
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)					1			
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)					1			
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)					1			
PS1 ← (ox*nx) + (oy*ny) + (Zo*nz)	2		3					
PS2 ← SHIFT(DIV(PS1 , NormeN2) , +1)					1			1
PS1 ← DIV(lx*nx) + (ly*ny) + (lz*nz) , NormeL*NormeN)	2		4		1			
Qx ← (PS2*nx) - ox		1	1					
Qy ← (PS2*ny) - oy		1	1					
Qz ← (PS2*nz) - Zo		1	1					
PS2 ← DIV((lx*Qx) + (ly*Qy) + (lz*Qz) , NormeL*NormeO)	2		4		1			
P ← POWER(PS2 , E)							1	
IR ← (MDR * PS1) + (MSR * P)	1		2					
IV ← (MDV * PS1) + (MSV * P)	1		2					
IB ← (MDB * PS1) + (MSB * P)	1		2					
TOTAL	19	5	20	6	3	3	1	1

A•3•4 Plusieurs sources placées à distance finie.

A•3•4•1 Première méthode.

Les modifications par rapport à l'algorithme du paragraphe A•3•3•1 sont indiquées en italique.

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)					1			
Nx ← DIV(nx , NormeN)					1			
Ny ← DIV(ny , NormeN)					1			
Nz ← DIV(nz , NormeN)					1			
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)					1			
Ox ← DIV(ox , NormeO)					1			
Oy ← DIV(oy , NormeO)					1			
Oz ← DIV(Zo , NormeO)					1			
TOTAL	4	2		3	6	2		

Par pixel et par source :								
[Ix , Iy , Iz] ← PSEUDO_DDA(x,y,z)	3	1	3					
NormeL2 ← SQ(Ix) + SQ(Iy) + SQ(Iz)	2			3				
NormeL ← SQRT(NormeL2)					1			
*PS1 ← (Ix*Nx) +(Iy*Ny) + (Iz*Nz)*	2		3					
PS2 ← SHIFT(PS1 , +1)								1
PS1 ← DIV(PS1, NormeL)					1			
rx ← (PS2*Nx) - Ix		1	1					
ry ← (PS2*Ny) - Iy		1	1					
rz ← (PS2*Nz) - Iz		1	1					
*PS2 ← DIV((rx*Ox) + (ry*Oy) + (rz*Oz) , NormeL)*	2		3		1			
P ← POWER(PS2 , E)							1	
*IR ← IR + (MDR * PS1) + (MSR * P)*	2		2					
*IV ← IV + (MDV * PS1) + (MSV * P)*	2		2					
*IB ← IB + (MDB * PS1) + (MSB * P)*	2		2					
TOTAL	15	4	18	3	2	1	1	1

A•3•4•2 Seconde méthode.

Les modifications par à rapport l'algorithme du paragraphe A•3•3•2 sont indiquées en italique.

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)					1			
Nx ← DIV(nx , NormeN)					1			
Ny ← DIV(ny , NormeN)					1			
Nz ← DIV(nz , NormeN)					1			
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)					1			
Ox ← DIV(ox , NormeO)					1			
Oy ← DIV(oy , NormeO)					1			
Oz ← DIV(Zo , NormeO)					1			
TOTAL	4	2	1	3	6	2	1	

Par pixel et par source :								
[Ix , Iy , Iz] ← PSEUDO_DDA(x,y,z)	3	1	3					
NormeL2 ← SQ(Ix) + SQ(Iy) + SQ(Iz)	2			3				
NormeL ← SQRT(NormeL2)					1			
Lx ← DIV(Ix , NormeL)					1			
Ly ← DIV(Iy , NormeL)					1			
Lz ← DIV(Iz , NormeL)					1			
*PS1 ← (Lx*Nx) +(Ly*Ny) + (Lz*Nz)*	2		3					
hx ← Lx + Ox	1							
hy ← Ly + Oy	1							
hz ← Lz + Oz	1							
NormeH2 ← SQ(hx) + SQ(hy) + SQ(hz)	2			3				
NormeH ← SQRT(NormeH2)					1			
*PS2 ← DIV((hx*Nx) +(hy*Ny) + (hz*Nz) , NormeH)*	2		3		1			
P ← POWER(PS2 , E)							1	
IR ← IR + (MDR * PS1) + (MSR * P)	2		2					
IV ← IV + (MDV * PS1) + (MSV * P)	2		2					
IB ← IB + (MDB * PS1) + (MSB * P)	2		2					
TOTAL	20	1	15	6	4	2	1	

A•3•4•3 Troisième méthode.

Les modifications par rapport à l'algorithme du paragraphe A•3•3•3 sont indiquées en italique.

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
Nx ← DIV(nx , NormeN)					1			
Ny ← DIV(ny , NormeN)					1			
Nz ← DIV(nz , NormeN)					1			
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)						1		
Ox ← DIV(ox , NormeO)					1			
Oy ← DIV(oy , NormeO)					1			
Oz ← DIV(Zo , NormeO)					1			
TOTAL	4	2		3	6	2	1	

Par pixel et par source :								
[lx , ly , lz] ← PSEUDO_DDA(x,y,z)	3	1	3					
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
Lx ← DIV(lx , NormeL)					1			
Ly ← DIV(ly , NormeL)					1			
Lz ← DIV(lz , NormeL)					1			
*PS1 ← (Lx*Nx) +(Ly*Ny) + (Lz*Nz)*	2		3					
hx ← Lx + Ox	1							
hy ← Ly + Oy	1							
hz ← Lz + Oz	1							
*NormeH ← SHIFT(1 + (Ox*Lx) + (Oy*Ly) + (Oz*Lz) , +1)*	3		3					1
*PS2 ← DIV((hx*Nx) +(hy*Ny) + (hz*Nz)) , NormeH)*	2		3		1			
P ← POWER(PS2 , E)						1		
*IR ← IR + (MDR * PS1) + (MSR * P)*	2		2					
*IV ← IV + (MDV * PS1) + (MSV * P)*	2		2					
*IB ← IB + (MDB * PS1) + (MSB * P)*	2		2					
TOTAL	21	1	18	3	4	1	1	1

A•3•4•4 Quatrième méthode.

Les *mo⁴ifications* à par rapport l'algorithme du paragraphe A•3•3•4 sont indiquées en italique

Par image et par matière :								
MDR ← I_rouge * Kd_rouge			1					
MDV ← I_vert * Kd_vert			1					
MDB ← I_bleu * Kd_bleu			1					
MSR ← I_rouge * Ks_rouge			1					
MSV ← I_vert * Ks_vert			1					
MSB ← I_bleu * Ks_bleu			1					
TOTAL			6					

Global à toutes les sources, par pixel :								
NormeN2 ← SQ(nx) + SQ(ny) + SQ(nz)	2			3				
NormeN ← SQRT(NormeN2)						1		
Nx ← DIV(nx , NormeN)					1			
Ny ← DIV(ny , NormeN)					1			
Nz ← DIV(nz , NormeN)					1			
ox ← Xo - x		1						
oy ← Yo - y		1						
NormeO2 ← DDA(x,y)	2							
NormeO ← SQRT(NormeO)						1		
Ox ← DIV(ox , NormeO)					1			
Oy ← DIV(oy , NormeO)					1			
Oz ← DIV(oz , NormeO)					1			
*PS1 ← (Ox*Nx) + (Oy*Ny) + (Oz*Nz)*	2		3					
*Qx ← SHIFT((PS1*Nx) , +1) - Ox*			1		1			1
*Qy ← SHIFT((PS1*Ny) , +1) - Oy*			1		1			1
*Qz ← SHIFT((PS1*Nz) , +1) - Oz*			1		1			1
TOTAL	6	3	6	3	6	2	1	3

Par pixel :								
[lx , ly , lz] ← PSEUDO_DDA(x,y,z)	3	1	3					
NormeL2 ← SQ(lx) + SQ(ly) + SQ(lz)	2			3				
NormeL ← SQRT(NormeL2)						1		
*PS1 ← DIV (lx*Nx) + (ly*Ny) + (lz*Nz) , NormeL)*	2		3		1			
PS2 ← DIV((lx*Qx) + (ly*Qy) + (lz*Qz) , NormeL)	2		3		1			
P ← POWER(PS2 , E)						1		
*IR ← IR + (MDR * PS1) + (MSR * P)*	2		2					
*IV ← IV + (MDV * PS1) + (MSV * P)*	2		2					
*IB ← IB + (MDB * PS1) + (MSB * P)*	2		2					
TOTAL	15	1	12	3	2	1	1	

ANNEXE B: Les méthodes d'approximation de l'éclairement spéculaire.

B•1 Les approximations polynomiales.

Sont étudiées, ici, différentes fonctions de type polynomial pour établir une approximation de la fonction x^n. Les coefficients de ces fonctions sont établies par une étude théorique ayant pour but d'obtenir des courbes, respectant un certain nombre de propriétés de la fonction x^n.

B•1•1 Linéaire par morceaux.

On peut établir une approximation de la fonction x^n à l'aide de morceaux de droite. On utilise un minimum de droites afin de simplifier au maximum la méthode d'approximation. Pour chaque morceau de droite, il faudra, au minimum, garder en mémoire dans la ROM son coefficient directeur. Comme le but de l'approximation est de réduire la taille de la ROM, il ne faut pas qu'il y ait un grand nombre de morceaux. De plus chaque raccord de droite ajoutera une cassure ou discontinuité de la dérivé qui peut engendrer des effets indésirables sur l'éclairement.

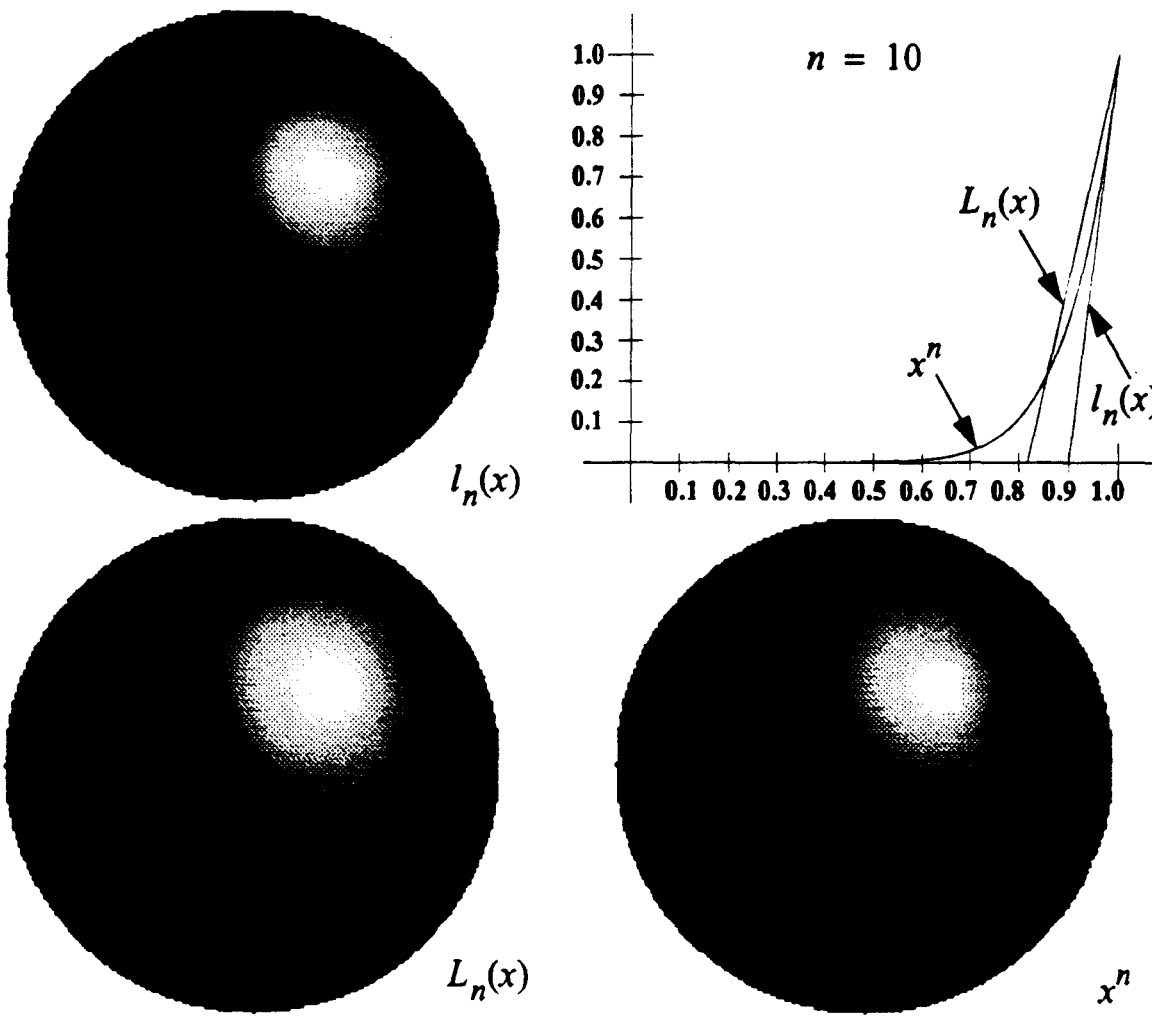


Figure B•1 L'approximation linéaire.

• Une approximation assez brutale est d'utiliser seulement deux morceaux. Le premier est constitué par un palier horizontal de valeur 0. Le second est un morceau de droite partant de l'axe des abscisses et ayant une pente égale à celle de la fonction x^n au voisinage de 1.

On a : $\frac{d}{dx}x^n(1) = n$ et $l_n(x) = \frac{x}{1-d} + \frac{d}{d-1}$. On en déduit : $d = 1 - \frac{1}{n}$ d'où : $l_n(x) = nx + (1-n)$.

Comme on peut l'observer sur la figure précédente, cette méthode d'approximation ne semble pas fournir un résultat satisfaisant. La tache lumineuse sera trop réduite.

• Une autre méthode d'approximation utilisant deux morceaux de droite consiste à déterminer l'équation du deuxième morceau de façon que l'aire de $L_n(x)$ soit la même que celle de x^n.

on a : $\int_0^1 x^n = \frac{1}{n+1}$ et on veut obtenir : $\int_0^1 L_n(x) = \int_d^1 (ax + 1 - a) = \frac{1}{2a} = \frac{1}{n+1}$.

On en déduit $a = \frac{2}{n+1}$ et $L_n(x) = \frac{2}{n+1}x + \frac{n-1}{n+1}$.

• Il est préférable de poser comme condition l'égalité des aires plutôt que celle de la dérivé en $x = 1$, on améliore l'approximation. On diminue l'erreur commise par rapport à l'originalmais son signe n'est plus toujours constant.

En prenant plus de droites, on améliorerait l'approximation, mais on obtiendrait une courbe avec des ruptures. Il faudrait utiliser des morceaux de courbes que l'on puisse raccorder entre eux plus facilement.

B•1•2 Quadratique par morceau.

• En utilisant des fonctions du second degré, on va pouvoir relier les morceaux en ajoutant une contrainte pour raccorder non seulement en continuité mais aussi en dérivabilité. Pour commencer, on va se contenter de n'utiliser que deux morceaux.

• Le premier est un palier horizontal de valeur 0 et le second est une portion de parabole joignant l'extrémité du palier au point (0,1). Pour calculer les paramètres, on impose que les courbes se raccordent de façon à être C_1 et qu'au voisinage de 1, la pente doit être égale à la pente de la fonction x^n.

Donc : $\frac{dq_n}{dx}(1) = \frac{d}{dx}x^n(x=1) = n$, $\frac{dq_n}{dx}(d) = 0$, $q_n(d) = 0$ et $q_n(1) = 1$, d'où on en déduit :

$d = \frac{n-2}{n}$ et $q_n(x) = \frac{n^2}{4}x^2 + \frac{n^2-2n}{2}x + \frac{(n-2)^2}{4}$.

L'approximation est certes meilleure, mais la courbe commence plus lentement que la fonction originale puis croît brutalement pour la rejoindre. Il faudrait regarder si on ne diminuerait pas l'erreur en préférant une fois encore la condition d'égalité des aires.

• On détermine les paramètres de la fonction $Q_n(x)$ pour qu'elle satisfasse à la contrainte sur l'égalité des aires.

on a : $\int_0^1 Q_n(x) = \int_d^1 \left(\frac{x-d}{1-d}\right)^2 = (1-d)\int_0^1 x^2 = \frac{1-d}{3}$ et on veut obtenir : $\int_0^1 Q_n(x) = \int_0^1 x^n = \frac{1}{n+1}$. On en déduit la valeur de $d = \frac{n-2}{n+1}$.

$$\text{D'ou : } Q_n(x) = \begin{cases} 0 & x \in [0, \frac{n-2}{n+1}] \\ \tilde{Q}_n(x) = \left(\dfrac{x - \frac{n-2}{n+1}}{1 - \frac{n-2}{n+1}} \right)^2 & x \in [\frac{n-2}{n+1}, 1] \end{cases} \quad \tilde{Q}_n(x) = \frac{(1+n)^2}{9}x^2 + \frac{2(n-2)(1+n)}{9}x + \frac{(n-2)^2}{9}$$

• Quoique la courbe croisse encore trop tardivement et trop brutalement, le résultat obtenu est assez satisfaisant sur une large portion de la courbe.

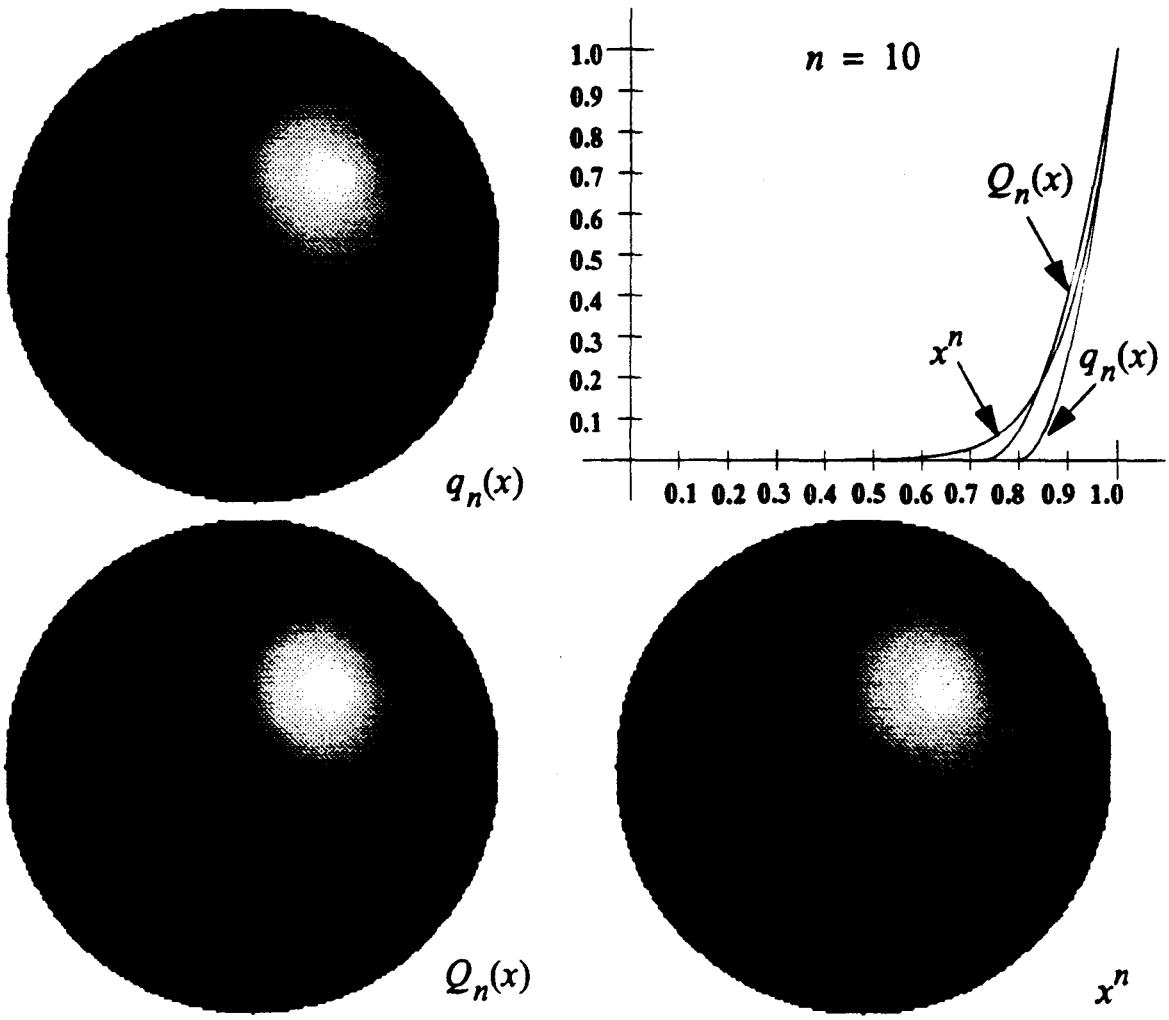


Figure B•2 L'approximation quadrique.

B•2 Les approximations rationnelles.

B•2•1 Fonction homographique.

Les approximations polynomiales ne nous satisfaisant pas complètement, il est nécessaire de chercher d'autres fonctions. Il faudrait choisir des fonctions qui puissent avoir une pente de plus en plus faible près de l'abscisse 0 et de plus en plus forte près de l'abscisse 1. Christophe Schlick [48] propose d'utiliser, dans les algorithmes de lancer de rayons, une fonction homographique pour établir une approximation de l'élévation à la puissance.

Soit : $H(x) = \dfrac{ax+b}{cx+d}$ la fonction homographique.

On détermine les coefficients (a, b, c, d) de manière à obtenir la meilleur approximation de $E(x) = x^n$

• Pour établir les formules de correspondance entre (a, b, c, d) et n, on choisit quatre propriétés de la fonction $E(x)$ qui devront être respectées par la fonction $H(x)$. Les propriétés sont les suivantes.

(1) $H(0) = 0$

(2) $H(1) = 1$

(3) $H'(1) = n$, $H'(x)$ étant la dérivée de la fonction $H(x)$.

(4) $H(x)$ continue sur $[0, 1]$

De l'équation (1), on déduit que b doit être nul. L'équation (2) nous indique que $a = c+d$. Ayant trois équations pour quatre inconnues, on peut fixer $a = 1 = c+d$. Avant de calculer la dérivée, on remplace a et b par leur valeur. On obtient ainsi :

$H(x) = \frac{x}{cx+d}$ et $H'(x) = \frac{d}{(cx+d)^2}$ et en appliquant la formule (3), on obtient :

$$H'(1) = \frac{d}{(c+d)^2} = d = n \ .$$

La fonction homographique est finalement la suivante : $H_n(x) = \frac{x}{(1-n)x+n}$

• Reste à vérifier la propriété de continuité, en calculant le domaine de définition. La seule discontinuité de la fonction $H_n(x)$ est en $x_d = 1 + [1/(n-1)] \Rightarrow x_d > 1$. Comme la discontinuité ne se trouve pas dans $[0, 1]$, la propriété est vérifiée pour $n > 1$.

• De même que pour la fonction quadratique, l'approximation sera peut-être améliorée en remplaçant la condition sur la dérivée par une condition sur l'aire. La relation (3) devient :

(3) $\int_0^1 H_n(x) = \int_0^1 x^n$.

Des relations (1), (2), et (4) on en déduit comme précédemment, en introduisant la variable m, la forme de l'approximation homographique : $H_n(x) = \frac{x}{(1-m)x+m}$. Reste à établir une relation entre m et n à partir de la relation (3).

En calculant la primitive de $H_n(x)$ on obtient l'équation suivante :

$$\int_0^1 H_n(x) = \int_0^1 \frac{x}{(1-m)x+m} = \frac{1}{1-m} + \frac{m \times \log(m)}{(1-m)^2} = \int_0^1 x^n = \frac{1}{n+1}$$

n	m	n	m	n	m	n	m	n	m	n	m
5	10.68	11	31.90	17	57.11	26	99.46	38	161.55	65	316.11
6	13.82	12	35.88	18	61.58	28	109.42	40	172.37	70	346.38
7	17.14	13	39.96	19	66.12	30	119.55	45	199.94	75	377.07
8	20.63	14	44.13	20	70.72	32	129.84	50	228.15		
9	24.26	15	48.38	22	80.09	34	140.27	55	256.96		
10	28.02	16	52.71	24	89.68	36	150.84	60	286.29		

Figure B•3 Tableau de correspondance avec égalité de la surface.

• Il n'y a pas de solution analytique connue à cette équation, et les logiciels de calcul formel tels que "Mathematica" et "Maple" ne donnent pas non plus de solution. Il faut donc utiliser le calcul numérique pour établir pour chaque valeur de n la valeur de m correspondante. Les résultats peuvent être précalculés et mis en mémoire dans un tableau pour être utilisés dans le calcul de l'éclairement.

Il est facile de parcourir chacune des valeurs de m pour calculer l'aire de la courbe, pour déduire la valeur de n correspondante et enfin remplir le tableau ci-dessous. Le calcul a été effectué avec une précision de un centième sur la valeur m.

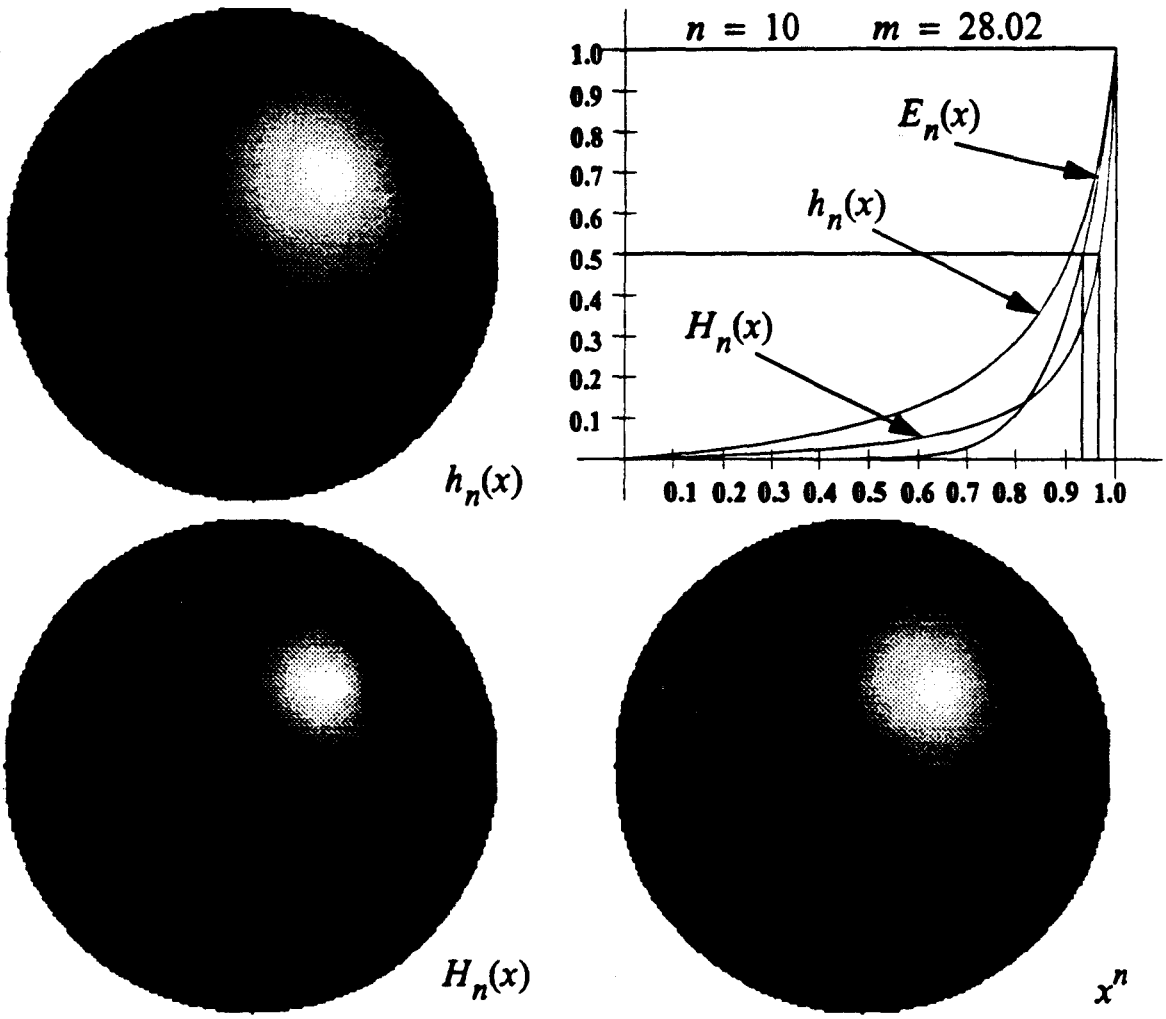


Figure B•4 L'approximation homographique.

• Malgré l'emploi de la condition sur la surface, on obtient une approximation qui paraît assez éloignée de la courbe $E_n(x) = x^n$. On remarque que la courbe utilisée pour l'approximation est environ deux fois plus proche de $x = 1$ pour $y = 0.5$. Voir Figure B•4. Cela signifie que la partie la plus lumineuse de la tache spéculaire paraîtra environ deux fois plus petite, puisqu'il faudra une variation deux fois moins importante du cosinus de l'angle entre $\hat{N}$ et $\hat{H}$ pour obtenir la même intensité lumineuse.

B•2•2 Fonction rationnelle de degré deux.

• Il faudrait accroître le nombre de paramètres indépendants dans la fonction. Pour cela on va étudier maintenant des fonctions rationnelles de degré supérieur.

Soit $H(x) = \frac{ax^2+bx+c}{ex+f}$, la fonction rationnelle qui doit approximer $E(x)$

Pour établir les formules de correspondance entre (a, b, c, e, f) et n, on choisit cinq propriétés de la fonction $E(x)$ qui devront être respectées par la fonction $H(x)$. Les propriétés sont les suivantes.

(1) $H(0) = 0$

(2) $H(1) = 1$

(3) $H'(0) = 0$, $H'(x)$ étant la dérivée de la fonction $H(x)$.

(4) $H'(1) = n$

(5) $H(x)$ continue sur $[0, 1]$

• De l'équation (1), on déduit que c doit être nul et f non nul. De l'équation (2) nous déduisons que $a+b = e+f$. En remplaçant c par sa valeur, on calcule la dérivée :

$H(x) = \frac{ax^2+bx}{ex+f}$ et $H'(x) = \frac{(2ax+b)(ex+f) - (ax^2+bx)e}{(ex+f)^2}$ et en appliquant la formule (3), on trouve : $b = 0$, ce qui permet de simplifier la fonction et sa dérivée :

$H(x) = \frac{ax^2}{ex+f}$, $H'(x) = \frac{(2ax)(ex+f) - (ax^2)e}{(ex+f)^2}$ et (4) implique $H'(1) = a\frac{2(e+f)-d}{(e+f)^2} = a\frac{e+2f}{(e+f)^2} = n$

• On peut fixer un paramètre $a = 1$, et ainsi en déduire la valeur des deux autres : $\{e = 1-f\} \Rightarrow \{(f = n-1), (e = 2-n)\}$.
La fonction rationnelle est finalement la suivante : $H(x) = \frac{x^2}{(2-n)x+(n-1)}$

• Vérifions enfin la propriété de continuité(5) en calculant le domaine de définition. La seule discontinuité de la fonction $H_n(x)$ est en $x_d = 1 + [1/(n-2)]$. Dès que l'exposant spéculaire est suffisamment grand : $n > 2 \Leftrightarrow x_d > 1$, la discontinuité ne se trouve pas dans $[0, 1]$, ainsi la propriété est vérifiée. Cependant pour $n = 1$ et $n = 2$, on peut malgré tout, trouver des qunituplés (a, b, c, e, f) tel que $H(x) = E(x)$. La fonction $H(x)$ associée est alors dégénérée. Les solutions respectives sont :
$((a = 0), (b = 1), (c = 0), (e = 0), (f = 1))$ et
$((a = 1), (b = 0), (c = 0), (e = 0), (f = 1))$.

• Bien que cela n'ait pas été particulièrement probant pour la fonction homographique, l'approximation sera peut-être améliorée en remplaçant la condition sur la dérivée par une condition sur l'aire. La relation (4) devient :

(4) $\int_0^1 H_n(x) = \int_0^1 x^n$.

En introduisant la variable m, des relation (1), (2), (3), et (5) on en déduit comme précédemment la forme de l'approximation homographique : $H(x) = \frac{x^2}{(2-m)x+(m-1)}$. Reste à établir une relation entre m et n à partir de la relation (4).

En calculant la primitive de $H_n(x)$ on obtient l'équation suivante :

$$\int_0^1 H_n(x) = \int_0^1 \frac{x^2}{(2-m)x+(m-1)} = \frac{4-3m}{2(2-m)^2} - \frac{(1-m)^2 \times \log(m-1)}{(2-m)^3} = \int_0^1 x^n = \frac{1}{n+1}$$

Une fois encore, il n'y a pas pas de solution analytique connue de cette équation, et les logiciels de calcul formel tels que "Mathematica" et "Maple" ne donnent pas de solution. Il faut donc utiliser le calcul numérique pour établir pour chaque valeur de n la valeur de m qui lui correspond. Les résultats peuvent être précalculés et mis en mémoire dans un tableau pour être utilisés dans le calcul de l'éclairement.

n	m	n	m	n	m	n	m	n	m	n	m
5	7,74	11	25,01	17	46,41	26	83,15	38	137,90	65	276.24
6	10,21	12	28,35	18	50,25	28	91,89	40	147,51	70	303.53
7	12,87	13	31,79	19	54,16	30	100,79	45	172,05	75	331.24
8	15,69	14	35,32	20	58,14	32	109,85	50	197,26		
9	18,67	15	38,94	22	66,26	34	119,06	55	223,06		
10	21,78	16	42,64	24	74,61	36	128,42	60	249,40		

Figure B•5 Tableau de correspondance pour l'approximation rationnelle.

• Il est facile de parcourir chacune des différentes valeurs de m pour calculer l'aire de la courbe, pour déduire la valeur de n qui lui correspond et enfin remplir le tableau ci-dessous. Le calcul a été effectué avec une précision de un centième sur la valeur m.

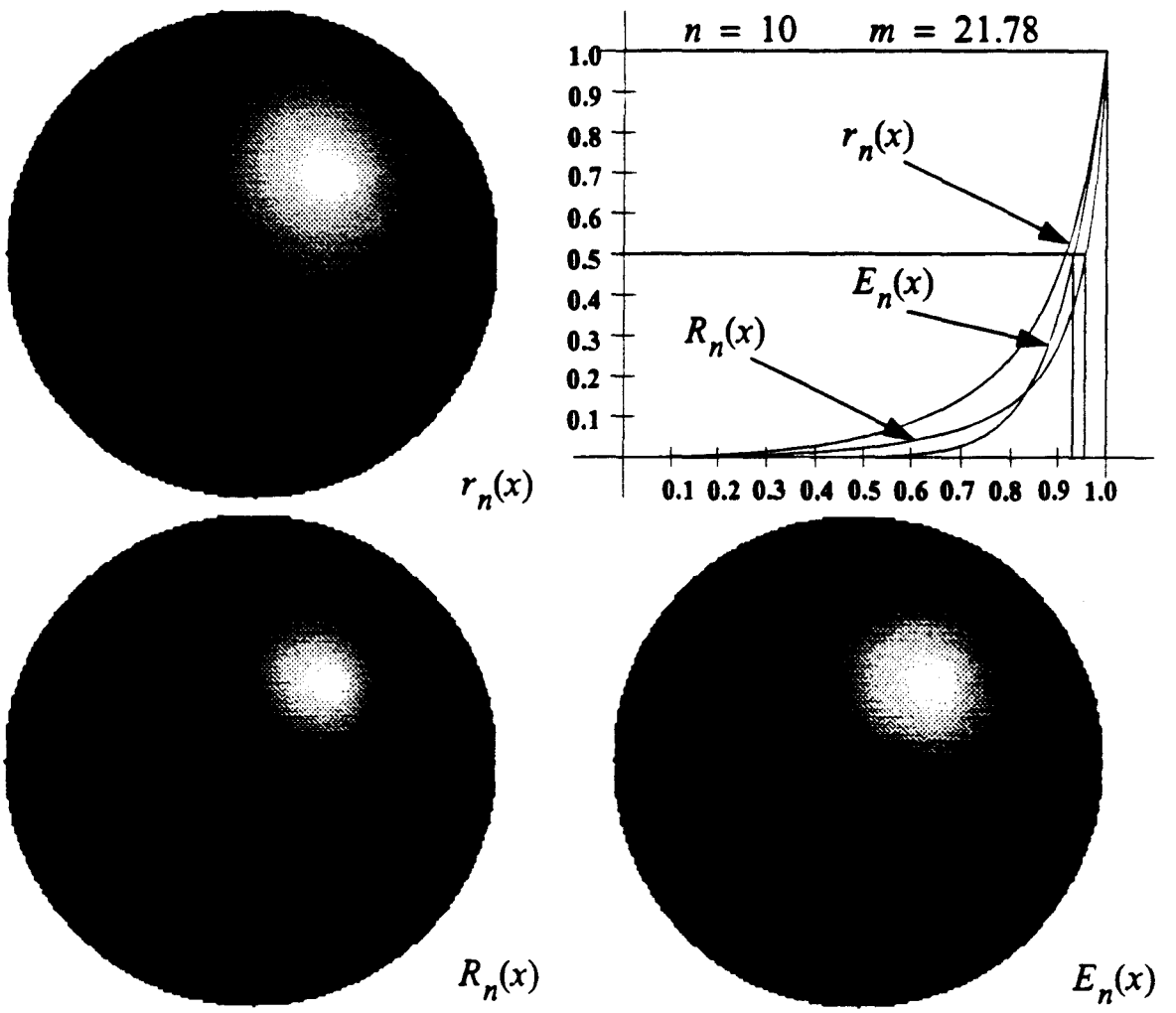


Figure B•6 Approximation rationnelle.

• Grâce à l'emploi de la condition sur la surface, on obtient une approximation qui paraît moins éloignée de la courbe $E_n(x) = x^n$. On remarque que la courbe s'est rapprochée de $x = 1$ dans un rapport d'environ deux sur trois pour $y = 0.5$ (cf Figure B•3). Cela signifie que la partie la plus lumineuse de la tache spéculaire paraîtra plus petite, sans que la différence soit aussi importante que celle obtenue par l'approximation homographique.

• En regardant la courbe de la fonction rationnelle, on remarque aisément que le problème se situe dans l'intervalle pendant lequel la fonction commence à croître de façon importante alors que la fonction $E_n(x)$ reste presque nulle. Ce problème ne nous est pas apparu lors de l'étude des fonctions quadratiques par morceaux, puisque l'on se définissait une plage pour laquelle la fonction servant à l'approximation est nulle. Il semble donc que la solution serait d'utiliser un palier nul, couplé à une fonction homographique ou rationnelle de degré supérieur.

B•3 Utilisation de fonction rationnelle par morceaux.

B•3•1 Fonction homographique.

• Comme nous l'avons déjà démontré, la complexité de l'architecture pour calculer une fonction homographique est la même que pour calculer une fonction rationnelle de degré deux au numérateur et de degré un au dénominateur. Passons donc directement à l'étude de la fonction rationnelle de degré deux.

B•3•2 Fonction rationnelle de degré deux.

• On définit la nouvelle fonction $H_n(x)$, servant à établir une approximation de $E_n(x) = x^n$, par un palier nul qui s'étend de 0 à d et par un morceau de fonction rationelle de degré deux, au numérateur et de degré un, au démoninateur. Pour fixer les paramètres de la fonction rationnelle, on se fixe les conditions suivantes :

(1) $H_n(d) = 0$

(2) $H_n(1) = 1$

(3) $H'_n(d) = 0$, $H'_n(x)$ étant la dérivée de la fonction $H_n(x)$.

(4) $H_n(x)$ continue sur $[d, 1]$

Nous avons déjà démontré, pour le cas où $d = 0$, que la solution est : $\frac{x^2}{(2-m)x+(m-1)}$. On peut ramener létude à celle du cas général par un changement de variable linéaire.

On utilise le changement de variable $t = \frac{x-d}{1-d}$, ainsi $x = d \rightarrow t = 0$ et $x = 1 \rightarrow t = 1$. Il s'agit, en fait, d'une mise à l'échelle de la courbe suivant l'axe des abscisses, comme cela a déjà été présenté dans la partie traitant de l'approximation par une fonction quadratique. La formule complète de la fonction est : $H_n(x) = \begin{bmatrix} 0 & \text{si } x \in [0,d] \\ f_m(\frac{x-d}{1-d}) & \text{si } x \in [d,1] \end{bmatrix}$ avec $f_m(t) = \frac{t^2}{(2-m)t+(m-1)}$.

Il ne reste plus qu'à déterminer les relations entre (m,d) et n en utilisant les deux conditions suivantes :

(5) $H'(1) = n.$

(6) $\int_0^1 H_n(x) = \int_0^1 x^n.$

De la relation (5) on en déduit : $H'(1) = f(1) \times \frac{1}{1-d} = \frac{m}{1-d} = n$ et à partir de la relation (6), en effectuant un changement de variable dans l'intégrale, on obtient :

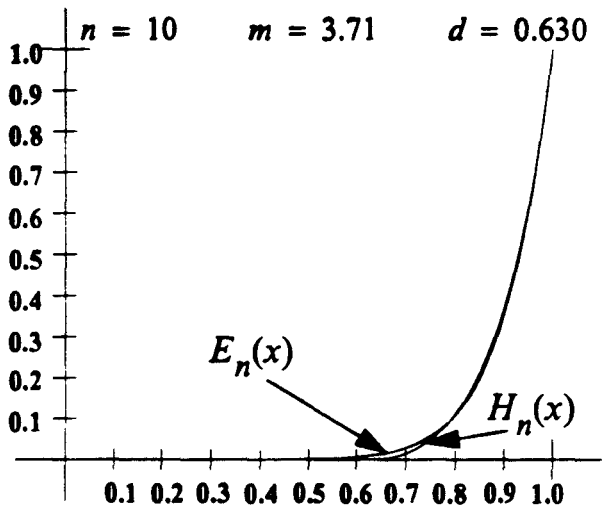
$\int_0^1 H_n(x) = \int_d^1 H_n(x) = \int_0^1 [f_m(t) \times (1-d)] = (1-d)\int_0^1 f_m(t)$, puis en calculant l'intégrale, on a :

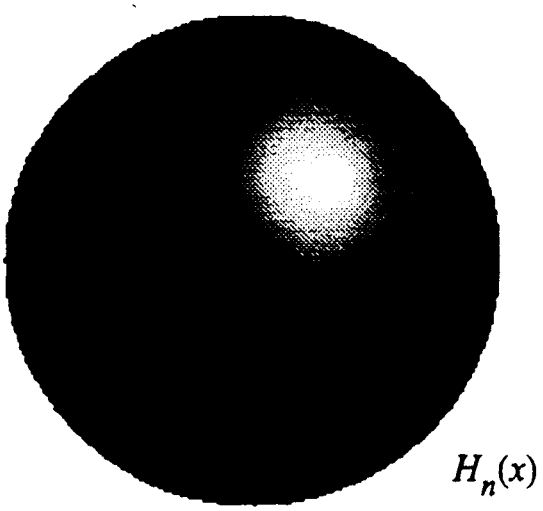
$\int_0^1 H_n(x) = (1-d)\left(\frac{4-3m}{2(2-m)^2} - \frac{(1-m)^2 \times \log(m-1)}{(2-m)^3}\right) = \int_0^1 x^n = \frac{1}{1+n}$. On peut utiliser la première relation entre n, m et d pour substituer l'une des variables dans la seconde relation.

Malheureusement, Il n'y a pas de solution analytique à l'équation ci-dessus et les logiciels "Mathematica" et "Maple" ne fournissent pas de solution. Il faut donc résoudre numériquement les deux équations pour trouver les valeurs de m et d pour chaque valeur de n.

Il n'est pas trop difficile d'écrire un programme qui calcule numériquement la table de correspondance. Il suffit pour cela de parcourir les différentes valeurs de m et de calculer la surface de $f_m(t)$ entre 0 et 1 puis de trouver à partir des deux relations les valeurs de d et de n.

n	m	d	n	m	d
5	3.12	0.379	16	3.98	0.752
6	3.30	0.453	17	4.01	0.766
7	3.43	0.508	18	4.03	0.775
8	3.54	0.555	19	4.05	0.785
9	3.63	0.595	20	4.07	0.789
10	3.71	0.630	22	4.11	0.813
11	3.78	0.661	24	4.14	0.828
12	3.82	0.679	26	4.16	0.839
13	3.87	0.702	28	4.19	0 850
14	3.91	0.720	30	4.21	0.860
15	3.95	0.734	35	4.25	0.879





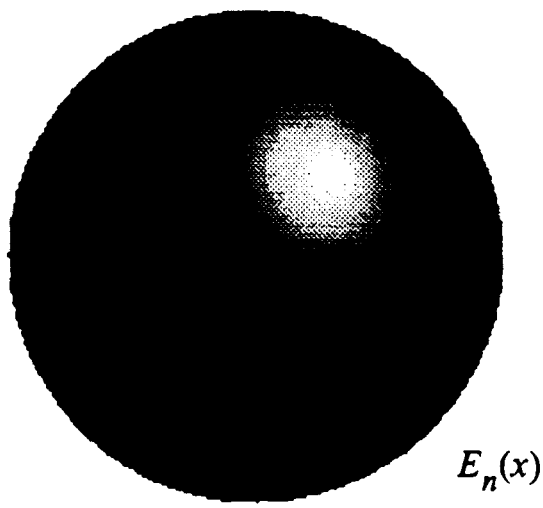


Figure B•7 Tableau de correspondance et exemple pour l'approximation rationnelle avec un palier nul.

• Le tableau ainsi établi, on peut se demander s'il ne serait pas possible de trouver une formule empirique pour obtenir d et m. En calculant, pour toutes les valeurs de n du tableau, la formule d^n, on trouve à peu près toujours le même nombre. Les fluctuations sont très faibles et la valeur moyenne est 0.011021 soit environ 0.01.

Pour n compris entre 8 et 200, d et m codés sur 12 bits, la plus grande erreur obtenue par rapport au calcul utilisant la fonction x^n est de 0.0218. La moyenne de l'erreur reste inférieure à 0.00508 (Valeur obtenue pour $n = 8$). Quand on augmente n, l'erreur maximum augmente très faiblement et très lentement, par contre l'erreur moyenne diminue. C'est la méthode qui donne les meilleurs résultats.

• Pour une valeur de n très grande la valeur de m s'approche de 4.50234476 sans que ce nombre ne soit jamais dépassé.

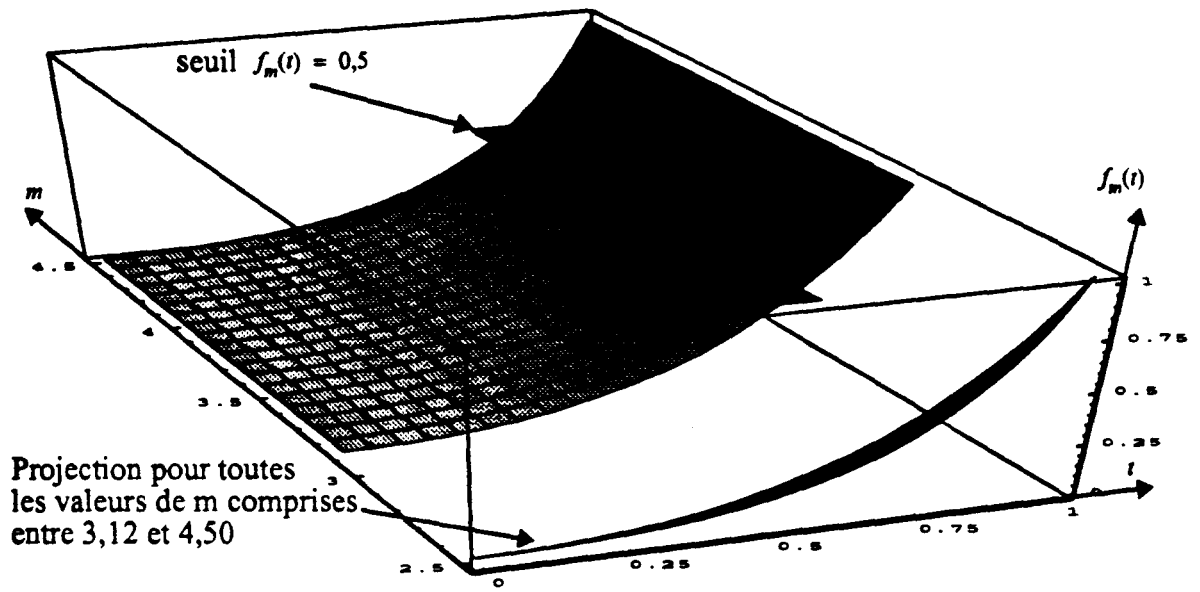


Figure B•8 Représentation 3D de l'ensemble des fonctions $f_m(t)$.

ANNEXE C: Algorithmes de chargement pour l'éclairement spéculaire .

C•1 Elaboration du premier algorithme de chargement.

Il s'agit de synthétiser un algorithme incrémental, pour le chargement des tables contenant la fonction élévation à la puissance.

• La formule du binôme de Newton nous donne l'expression mathématique qu'il faudrait utiliser pour calculer de façon incrémentale l'élévation à la puissance :

$(x+p_x)^n = x^n + \left[\sum_{i=0}^{n-1} (C_i^n \times x^i \times p_x^{(n-i)})\right] = x^n + \delta_n(x)$ avec C_i^n les coefficients du triangle de Pascal.

Cela nécessiterait de garder les n x^i précédents et donc de calculer les n $(x+p_x)^i$ pour l'étape suivante. On pourrait envisager de se limiter dans l'utilisation des termes du binôme de Newton, la valeur $p_x^{(n-i)}$ étant négligeable pour certaines valeurs de i. Malheureusement, cela ne permet pas de limiter le nombre de x^i devant être conservé. Ceux qui ne seraient pas utilisés pour le calcul de x^n le seraient dans le calcul d'autres x^i servant directement ou indirectement au calcul de x^n. Il ne semble donc pas utile d'étudier précisément les effets d'une approximation de la formule du binôme de Newton. Ni la formule directe ni aucune de ces approximations ne seront employées. Il faudrait limiter le nombre d'étapes de calcul utilisées pour chaque itération de l'algorithme.

On peut, bien sûr, stocker dans une ROM les valeurs de $\delta_n(x)$. Mais, pour le moment, on ne voit pas quel est l'intérêt de garder $\delta_n(x)$ plutôt que x^n. Cela requiert une unité d'addition supplémentaire et cela ne permet pas de réduire la taille de la ROM.

• Gardons, pour l'instant, l'idée que l'élévation à la puissance utilisera une ROM et voyons si l'on ne pourrait pas trouver un moyen de supprimer la multiplication. Une table de multiplication peut aisément être calculée de manière incrémentale. Une simple unité d'addition suffit. L'expression mathématique $[K \times (x+p_x)] = [K \times x] + (K \times p_x)$ nous montre comment la table se remplit par additions successives de $(K \times p_x)$.

Pour l'élévation à la puissance cela n'est pas aussi simple car l'entrée de l'opérateur ne croît plus de manière régulière. On obtient, cette fois, $K_s \times (x+p_x)^n = K_s \times (x)^n + K_s \times \delta_n(x)$.

Cela n'améliore pas vraiment la complexité, car il y a toujours une multiplication, pour calculer cette fois : $K_s \times \delta_n(x)$. Il y a cependant un espoir. En utilisant une approximation brutale de $\delta_n(x)$, on devrait pouvoir simplifier l'unité de multiplication.

Posons : $(x+p_x)^n \approx (x)^n + \Delta_n(x)$ avec $\Delta_n(x) \in \{0, p_y\}$. Soit $e_n(i)$ les valeurs successives qui approximent $(i \times p_x)^n$. Alors, on a : $e_n(i) = \sum_1^n \Delta(i \times p_x)$ et, les $\Delta_n(x)$ sont choisis tel que $(i \times p_x)^n \le e_n(i) < (i \times p_x)^n + p_y$

En procédant ainsi, les erreurs ne s'accumulent pas mais au contraire sont rattrapées dès que possible lors du tracé de la fonction $(x)^n$. L'erreur induite est la même que celle produite par la quantification des valeurs de la sortie. Cela n'ajoute donc aucune erreur supplémentaire lors du tracé de la fonction $(x)^n$. Reste à effectuer la multiplication par K_s qui est grandement simplifiée, P_x étant une puissance de 2. Une rangée d'opérateurs logiques "et" synthétise efficacement l'unité de multiplication.

La figure ci-dessous montre le tracé de la fonction $f(x) = x^3$ à partir de $\Delta_n(x)$. On observe que $\Delta_n(x) = 0$ provoque, en fait, un déplacement horizontal et $\Delta_n(x) = 1$ un déplacement diagonal.

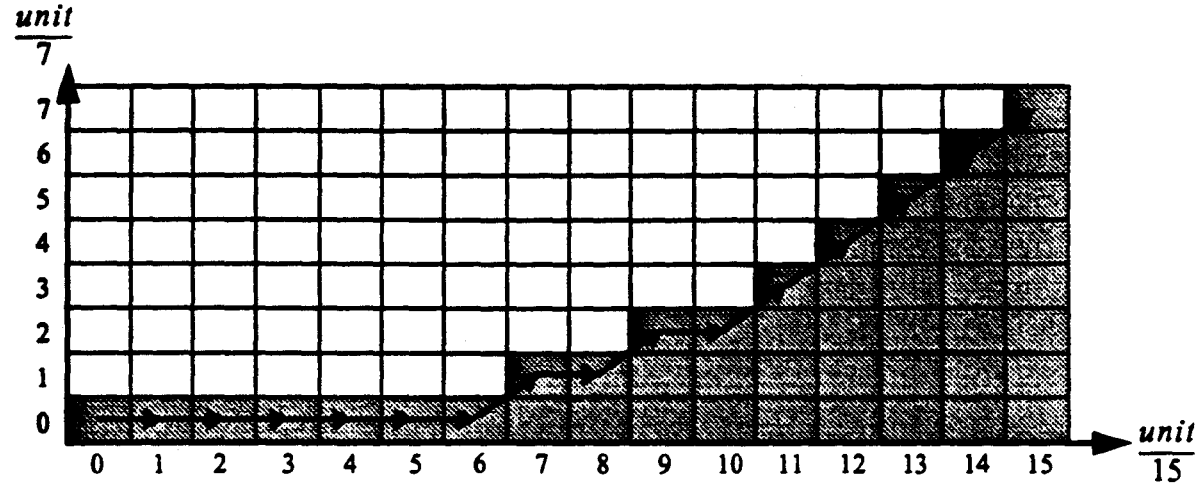


Figure 3•1 Exemple de tracé incrémental de la fonction $f(x) = x^3$.

• Montrons par un contre-exemple que cela ne fonctionne pas. Il n'est pas toujours possible de satisfaire les conditions posées pour calculer la suite des $\Delta_n(x)$ avec $x = i \times p_x$ et $x \in]0,1]$.

Prenons $n = 4$, $p_x = 1/15$ et $p_y = 1/7$,on obtient pour $x = 13$, $x = 14$ et $x = 15$:

$\left(\frac{13}{15}\right)^4 = \frac{199927}{354375} \approx \frac{3.95}{7}$ $\left(\frac{14}{15}\right)^4 = \frac{268911}{354375} \approx \frac{5.31}{7}$ $\left(\frac{15}{15}\right)^4 = \frac{7}{7}$. Des deux sous-suites de $e_n(i)$ satisfaisant la propriété $(i \times p_x)^n \le e_n(i) < (i \times p_x)^n + p_y$: $\left\{\frac{4}{7}, \frac{5}{7}, \frac{7}{7}\right\}$ $\left\{\frac{4}{7}, \frac{6}{7}, \frac{7}{7}\right\}$

aucune ne vérifie la condition de base : $\Delta_n(x) \in \{0, \frac{1}{7}\}$

• Cela montre qu'il est nécessaire que $\Delta_n(x)$ puisse prendre des valeurs entières supérieures à 1. Les valeurs exigées sont d'autant plus grandes que n est grand. Au niveau du calcul de la multiplication, autoriser le chiffre 2 pour $\Delta_n(x)$ augmenterait déjà de façon appréciable la complexité et nous serions en bonne voie pour réintroduire la multiplication que l'on désirait supprimer.

• Pour résoudre la difficulté, quand $\Delta_n(x) > 1$ on ajoute autant de fois qu'il le faut K_x de manière itérative, ainsi il n'y a pas besoin de multiplication.

Remarque : toutes les fonctions x^n sont croissantes et leurs courbes passent par les points $(0,0)$ et $(1,1)$. Pour passer de $(0,0)$ à $((p_x/p_x), (p_y/p_y))$, on doit faire $P_x - 1$ pas horizontaux et $P_y - 1$ pas verticaux, peu importe le chemin choisi à condition de toujours monter ou d'aller vers la droite. Donc, en supprimant les déplacements diagonaux, c'est-à-dire en les remplaçant par un déplacement horizontal suivi d'un déplacement vertical, le nombre d'étapes de calcul sera constant, indépendamment de la puissance spéculaire. Cela permet de résoudre le problème rencontré pour $n = 4$.(cf Figure 3•14)

C•2 Etude du chargement d'une fonction homographique.

• Avant d'aller plus avant, une question se pose. Existe-t-il un moyen qui ne soit pas trop coûteux pour calculer les fonctions $H(x)$ et charger les résultats dans les tables d'indirection du spéculaire (les RAMs)? En d'autres termes, quel est le coût matériel de cette solution?

Il est clair qu'il n'est pas souhaitable de mettre un opérateur de division dans l'unité arithmétique qui calcule les valeurs à charger dans les tables. Il serait avantageux de trouver un algorithme incrémental.

On va essayer d'utiliser, une fois encore, le principe des déplacements horizontaux et verticaux, pour le tracé de la fonction. Dans la première version, la nature des déplacements était connue de l'automate de chargement en regardant dans la ROM. L'implémentation matérielle va donc scrupuleusement se substituer à la ROM.

• Sur $[0,1]$, l'intervalle où la fonction homographique est utilisée, elle est continue et croissante. Par rapport à cette courbe, un point est donc, soit au dessus et à gauche, soit en dessous et à droite.

La logique de décision entre un pas horizontal et un pas vertical doit être inspirée par la volonté de rester le plus près possible de la courbe réelle. Si on se trouve au dessus de la courbe, pour s'en rapprocher, il ne faut surtout pas monter. On va donc avancer horizontalement. Si on se trouve à droite, il ne faut pas se déplacer horizontalement; on s'éloignerait. On va donc se déplacer verticalement.

Dans un premier temps, l'algorithme se traduit ainsi :

```
Faire
        X <-- 0
        Y <-- 0
        Répéter
        H <-- (ax + b)/(cx + d)
                Si      (y + 1 > H)
                Alors
                        Déplacement Horizontal
                Sinon
                        Déplacement Vertical
                FinSi
        Jusqu'à ( x = 1 )
        FinRépéter
Fait
```

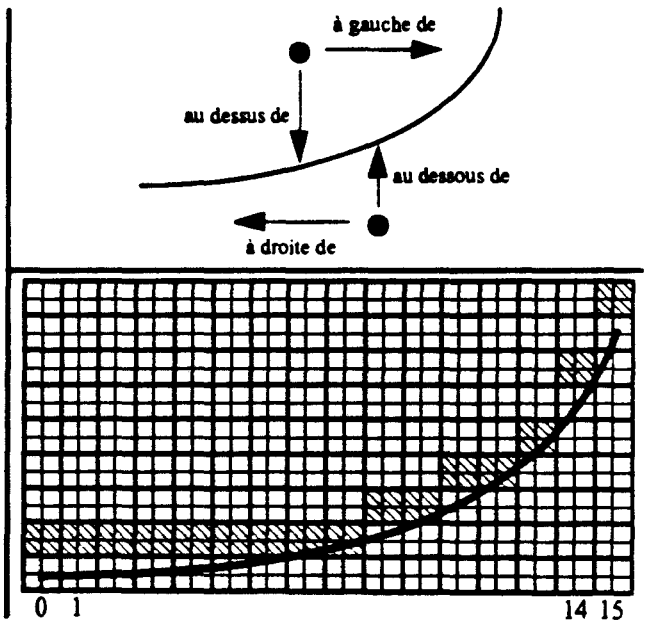


Figure C•2 Algorithme initial de tracé d'une fonction homographique.

• De par l'algorithme qui est écrit, pour chaque abscisse, l'ordonnée choisie sera toujours la plus haute située au dessous de la courbe. On vérifie l'exactitude du test en posant :
$y_{inf} = E(H) \Rightarrow y_{inf} \leq H < y_{inf} + 1 \Rightarrow (H-1) < y_{inf} \leq H \Rightarrow (y_{inf} + 1) > H.$
Cela correspond aux carreaux faiblement grisés dans le dessin ci-dessus. En remplaçant la condition par *{Si (y > H) Alors}*, on choisirait toujours l'ordonnée située le plus bas au dessus de la courbe. Dans le dessin, cela correspond aux carreaux hachurés. On obtiendrait un meilleur tracé en choisissant un compromis, le test devenant ainsi *{Si (y + 0.5 > H) Alors}*. On peut y trouver une certaine correspondance avec l'algorithme de tracé de droite de Bresenham [5] dans lequel il y a une erreur initiale de 0.5 pour obtenir la "meilleure" droite. ("Meilleure", signifiant "la plus proche de la droite réelle".)

• L'algorithme étant établi, il faut remplacer la comparaison et le calcul réel de la fonction homographique, par un test sur une expression que l'on peut calculer de manière incrémentale.
En transformant l'expression, on supprime la division.

$$test(y + 0.5 > \frac{ax+b}{cx+d}) \Leftrightarrow test((y+0.5)(cx+d) > ax+b) \Leftrightarrow test(cx(y+0.5) - ax + d(y+0.5) - b > 0)$$

à condition que $\forall x \in I \quad cx + d > 0$. Posons $\delta(x,y) = cxy - ax + dy - b$. Le test devient ainsi *{Si ($\delta(x, y+0.5) > 0$) Alors}*. Le déplacement dépend du signe de $\delta(x,y)$ qui peut être, de tout évidence, calculé de façon incrémentale.

On peut assimiler $\delta(x,y)$ à une distance algébrique. C'est une distance entre la position (x,y) et la courbe de la fonction homographique.

• Le déplacement horizontal consiste à accroître la valeur de l'abscisse d'une graduation. Posons $P_x = 1/N_x$, la largeur d'une graduation de l'axe des abscisses, N_x étant le nombre de graduations présentes dans $[0,1]$. L'opération $x \leftarrow x + P_x$ correspond au déplacement horizontal. De même, posons $P_y = 1/N_y$, la largeur d'une graduation de l'axe des ordonnées, N_y étant le nombre de graduations présentes dans $[0,1]$.

Pour obtenir l'algorithme incrémental, on calcule les accroissements de $\delta(x,y)$ pour les déplacements horizontaux et verticaux.

$$\begin{array}{llll}
\delta(x+P_x,y) = \delta(x,y) + \delta_x(y) & \Rightarrow \delta_x(y) = \delta(x+P_x,y) - \delta(x,y) & \Rightarrow \delta_x(y) = (cy-a)P_x \text{ et } \delta_x(0) = -aP_x \\
\delta(x,y+P_y) = \delta(x,y) + \delta_y(x) & \Rightarrow \delta_y(x) = \delta(x,y+P_y) - \delta(x,y) & \Rightarrow \delta_y(x) = (cy+d)P_y \text{ et } \delta_x(0) = dP_y \\
\delta_x(y+P_y) = \delta_x(y) + \delta_{xy}(y) & \Rightarrow \delta_{xy}(y) = \delta_x(y+P_y) - \delta_x(y) & \Rightarrow \delta_x(y) = cP_xP_y \\
\delta_y(x+P_x) = \delta_y(x) + \delta_{yx}(x) & \Rightarrow \delta_{xy}(y) = \delta_x(x+P_x) - \delta_y(x) & \Rightarrow \delta_x(y) = cP_xP_y
\end{array}$$

• Comme seul le signe de $\delta(x,y)$ est utilisé, on peut le multiplier par le nombre entier positif $2N_xN_y$ et cela ne l'altérera pas. Cela permet de travailler uniquement avec des nombres entiers. Cela simplifiera les calculs ainsi que l'architecture. Les formules deviennent :

$$\begin{array}{ll}
\Delta(x,y+0.5P_y) = 2(cxy - ax + dy - b)N_xN_y + (cx+d)N_x & \Delta_x(x,y) = 2(cy-a)N_y \\
\Delta_y(x,y) = 2(cx+d)N_x \qquad \Delta_{xy}(x,y) = 2c = \Delta_{yx}(x,y) & \Delta_{xx}(x,y) = \Delta_{yy}(x,y) = 0
\end{array}$$

On suppose que la courbe de $H(x)$ passe par (X_début, Y_début) et donc Δ_{xy}(X_début, Y_début) = 0

- L'algorithme de tracé incrémental de la fonction $H(x) = \frac{ax+b}{cx+d}$ est

Faire

$X \leftarrow$ X_début
$Y \leftarrow$ Y_début
$\Delta_x \leftarrow 2(c \times (\text{Y\_début} + 0.5P_y) - a)N_y$
$\Delta_y \leftarrow 2(c \times \text{X\_début} + d)N_x$
$\Delta \leftarrow (c \times \text{X\_début} + d)N_x$
Répéter
  Si $(\Delta > 0)$
  Alors
    $\Delta \leftarrow \Delta + \Delta_x$
    $\Delta_x \leftarrow \Delta_x + 0$ $\{\Delta_x \leftarrow \Delta_x + \Delta_{xx}\}$
    $\Delta_y \leftarrow \Delta_y + 2c$ $\{\Delta_y \leftarrow \Delta_y + \Delta_{xy}\}$
    Déplacement Horizontal
  Sinon
    $\Delta \leftarrow \Delta + \Delta_y$
    $\Delta_x \leftarrow \Delta_x + 2c$ $\{\Delta_x \leftarrow \Delta_x + \Delta_{yx}\}$
    $\Delta_y \leftarrow \Delta_y + 0$ $\{\Delta_y \leftarrow \Delta_y + \Delta_{yy}\}$
    Déplacement Vertical
  FinSi
  Jusqu'à $(X = X\_fin)$
FinRépéter
Fait

Figure C•3 Algorithme incrémental de tracé de fonction homographique.

C•3 Etude du chargement d'une fonction rationnelle.

Soit : $H(x) = \frac{a \cdot x^2 + b \cdot x + c}{d \cdot x^2 + e \cdot x + f}$ la fonction rationnelle de degré deux.

On va utiliser le même algorithme que précédemment. (cf Figure C•2) On change juste l'expression pour le calcul de H et on utilise le test amélioré *{Si (y + 0.5 > H) Alors}*. En transformant l'expression, on supprime la division puis on passe en calcul incrémental.

$$test(y + 0.5 > \frac{a \cdot x^2 + b \cdot x + c}{d \cdot x^2 + e \cdot x + f}) \Leftrightarrow test(d \cdot x^2 (y + 0.5) - (a \cdot x^2) + e \cdot x (y + 0.5) - (b \cdot x) + f(y + 0.5) > 0)$$

à condition que $\forall x \in I \quad dx^2 + ex + f > 0$. Posons $\delta(x,y) = dx^2y - ax^2 + exy - bx + fy - c$. Le test devient, comme précédemment, *{Si ($\delta(x, y + 0.5) > 0$) Alors}*. L'expression $\delta(x,y)$ peut, de tout évidence, être calculée de façon incrémentale.

- L'expression $\delta(x,y) = dx^2y - ax^2 + exy - bx + fy - c$ est une forme bilinéaire de degré 3 au lieu d'en être une de degré 2 comme dans le cas des fonctions homographiques. Cet accroissement de degré va augmenter la complexité de l'algorithme et de l'automate câblé. Pour que $\delta(x,y)$ soit une forme bilinéaire de degré 2, il faudrait que : $d = 0$.

Dans un premier temps on va donc se restreindre à $H(x) = \frac{a \cdot x^2 + b \cdot x + c}{e \cdot x + f}$

et simplifier l'expression de l'erreur $\delta(x,y) = -ax^2 + exy - bx + fy - c$.

• Posons $P_x = 1/N_x$ la largeur d'une graduation de l'axe des abscisses, N_x étant le nombre de graduations présentes dans $[0,1]$, et $P_y = 1/N_y$, la largeur d'une graduation de l'axe des ordonnés, N_y étant le nombre de graduations présentes dans $[0,1]$. Pour obtenir l'algorithme incrémental, on calcule les accroissements du premier et second ordre de $\delta(x,y)$ concernant les déplacements horizontaux et verticaux.

$$\delta_x(x,y) = \delta(x+P_x,y) - \delta(x,y) \quad \Rightarrow \quad \delta_x(x,y) = (e\cdot y - (2a\cdot x) - aP_x - b)P_x$$
$$\delta_y(x,y) = \delta(x,y+P_y) - \delta(x,y) \quad \Rightarrow \quad \delta_y(x,y) = (ex+f)P_y$$
$$\delta_{xx}(x,y) = \delta_x(x+P_x,y) - \delta_x(x,y) \quad \Rightarrow \quad \delta_{xx}(x,y) = -2a\cdot P_x\cdot P_x$$
$$\delta_{xy}(x,y) = \delta_x(x,y+P_y) - \delta_x(x,y) \quad \Rightarrow \quad \delta_{xy}(x,y) = e\cdot P_x\cdot P_y$$
$$\delta_{yx}(x,y) = \delta_y(x+P_x,y) - \delta_y(x,y) \quad \Rightarrow \quad \delta_{yx}(x,y) = e\cdot P_x\cdot P_y$$

• Comme seul le signe de $\delta(x,y)$ est utilisé, on peut procéder de la manière déjà utilisée pour les fonctions homographiques. On va multiplier l'expression $\delta(x,y)$ par le nombre entier positif $2\cdot N_x^2\cdot N_y$. Cela permet de travailler uniquement avec des nombres entiers et cela simplifiera les calculs ainsi que l'architecture. Les formules deviennent :

$$\Delta(x,y+0.5P_y) = 2[-(a\cdot x^2) + e\cdot x\cdot y - (b\cdot x) + f\cdot y - c]\cdot N_x^2\cdot N_y + (e\cdot x+f)\cdot N_x^2$$
$$\Delta(x,y+0.5P_y) = \Delta(x,y) + (e\cdot x+f)\cdot N_x^2$$
$$\Delta_x(x,y) = 2(e\cdot y - (2a\cdot x) - b)\cdot N_x\cdot N_y - (2a\cdot N_y)$$
$$\Delta_y(x,y) = 2(e\cdot x+f)\cdot N_x^2$$
$$\Delta_{xx}(x,y) = -4a\cdot N_y$$
$$\Delta_{xy}(x,y) = 2e\cdot N_x = \Delta_{yx}(x,y)$$
$$\Delta_{yy}(x,y) = 0$$

On suppose que la courbe de $H(x)$ passe par (X_début, Y_début) et donc Δ_{xy}(X_début, Y_début) = 0

• L'algorithme de tracé incrémental de la fonction $H(x) = \dfrac{a\cdot x^2+b\cdot x+c}{e\cdot x+f}$ est

Faire
  $X \leftarrow$ X_début
  $Y \leftarrow$ Y_début
  $\Delta_x \leftarrow 2(e\times(\text{Y\_début} + 0.5P_y) - (2a\times\text{X\_début}) - b)\times N_x\times N_y - 2a\times N_y$
  $\Delta_y \leftarrow 2(e\times\text{X\_début} + f)N_x^2$
  $\Delta \leftarrow (e\times\text{X\_début} + f)\times N_x^2$
  Répéter
    Si ($\Delta > 0$)
    Alors
      $\Delta \leftarrow \Delta + \Delta_x$
      $\Delta_x \leftarrow \Delta_x + -4a\times N_y$ {$\Delta_x \leftarrow \Delta_x + \Delta_{xx}$}
      $\Delta_y \leftarrow \Delta_y + 2e\times N_x$ {$\Delta_y \leftarrow \Delta_y + \Delta_{xy}$}
      Déplacement Horizontal
    Sinon
      $\Delta \leftarrow \Delta + \Delta_y$
      $\Delta_x \leftarrow \Delta_x + 2e\times N_x$ {$\Delta_x \leftarrow \Delta_x + \Delta_{yx}$}
      $\Delta_y \leftarrow \Delta_y + 0$ {$\Delta_y \leftarrow \Delta_y + \Delta_{yy}$}
    FinSi
    Jusqu'à ($X = X\_fin$)
  FinRépéter
Fait

Figure C•4 Algorithme incrémental de tracé de la fonction rationnelle (2/1).

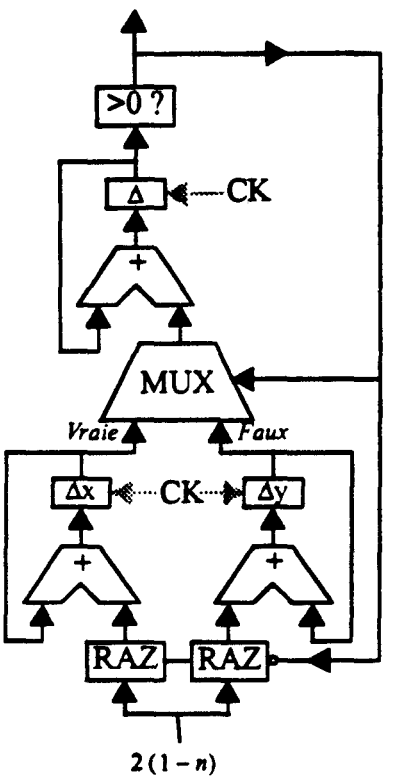
C•4 Application à l'approximation homographique.

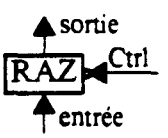
• A présent, oublions le cas général du tracé des fonctions homographiques pour nous intéresser seulement aux fonctions qui vérifient les quatre propriétés servant à l'approximation.

Dès lors la condition de validité est toujours vérifiée, du fait de la propriété (4) sur les fonctions $H_n(x)$. En remplaçant, d'autre part, X_début et Y_début par leur valeur : 0 et 0, on obtient le programme suivant :

Faire
- $X \leftarrow 0$
- $Y \leftarrow 0$
- $\Delta_x \leftarrow 1 - 2N_x - n$
- $\Delta_y \leftarrow 2nN_x$
- $\Delta \leftarrow nN_x$
- Répéter
  - Si $(\Delta > 0)$
  - Alors
    - $\Delta_x \leftarrow \Delta_x + 0$
    - $\Delta_y \leftarrow \Delta_y + 2(1-n)$
    - $\Delta \leftarrow \Delta + \Delta_x$
    - Déplacement Horizontal
  - Sinon
    - $\Delta_x \leftarrow \Delta_x + 2(1-n)$
    - $\Delta_y \leftarrow \Delta_y + 0$
    - $\Delta \leftarrow \Delta + \Delta_y$
    - Déplacement Vertical
  - FinSi
  - Jusqu'à ($X = X\_fin$)
- FinRépéter

Fait





Si Ctrl est vrai, l'entrée est recopiée sur la sortie, sinon la sortie est nulle.

Si Ctrl est faux, l'entrée est recopiée sur la sortie, sinon la sortie est nulle.

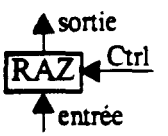


Figure C•5 Algorithme et automate câblé pour tracer l'approximation homographique

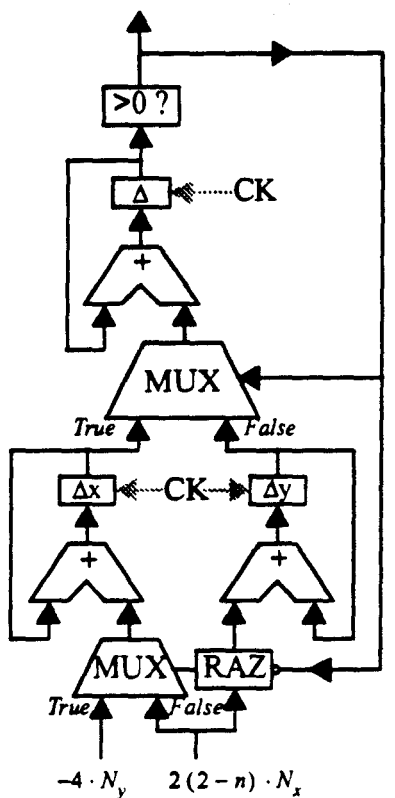
• On peut facilement synthétiser un automate câblé qui exécutera cet algorithme. Quel que soit le résultat du test, on ajoute des données à δ, δ_x et δ_y. L'automate dispose donc de trois unités d'accumulation composées chacune d'un additionneur et d'un registre pour garder en mémoire les valeurs successives. Les données ajoutées dépendent du signe de δ. Un multiplexeur, faisant le dispatching des valeurs, doit donc être ajouté en entrée des unités d'accumulation. Par souci d'économie, on peut remplacer deux des multiplexeurs par des unités de mise à zéro (RAZ) car, pour ces deux-là, l'une des deux valeurs d'entrée est nulle.

• En observant la courbe de la fonction homographique, on remarque aisément que le problème se situe dans le voisinage de l'abscisse 0. La dérivée en 0 est : $H'(0) = 1/n$, alors que la dérivée de la fonction $E(x)$ en 0, est nulle. Si on pouvait ajouter, comme condition supplémentaire, $H'(0) = 0$ cela devrait permettre d'aplatir la courbe dans le voisinage de 0 et ainsi améliorer l'approximation. Mais, il n'est pas possible, de demander aux fonctions homographiques de satisfaire une condition de plus. Le degré de liberté des fonctions homographiques est déjà saturé. Dorénavant, oublions les fonctions $H(x)$ et $H_n(x)$ telles qu'elles avaient été définies.

C•5 Application à l'approximation rationnelle.

• En remplaçant, X_début et Y_début par leur valeur : 0 et 1 dans l'algorithme général, on obtient le programme suivant :

Faire
  $X \leftarrow 0$
  $Y \leftarrow 0$
  $\Delta_x \leftarrow -n \cdot N_x + 2(N_x - N_y)$
  $\Delta_y \leftarrow 2(n-1) \cdot N_x$
  $\Delta \leftarrow (n-1) \cdot N_x^2$
  Répéter
    Si $(\Delta > 0)$
    Alors
      $\Delta_x \leftarrow \Delta_x + -4 \cdot N_y$
      $\Delta_y \leftarrow \Delta_y + 2(2-n) \cdot N_x$
      $\Delta \leftarrow \Delta + \Delta_x$
      Déplacement Horizontal
    Sinon
      $\Delta_x \leftarrow \Delta_x + 2(2-n) \cdot N_x$
      $\Delta_y \leftarrow \Delta_y + 0$
      $\Delta \leftarrow \Delta + \Delta_y$
      Déplacement Vertical
    FinSi
    Jusqu'à ($X = X\_fin$)
  FinRépéter
Fait



Si Ctrl est vrai, l'entrée est recopiée sur la sortie sinon la sortie est nulle.

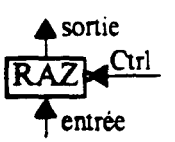


Figure C•6 Algorithme et automate câblé pour tracer l'approximation rationnelle

ANNEXE D: Les configurations de l'unité de normalisation.

D•1 Architecture avec unités séparées.

m1 m2 m3 m4	Valeur de l'erreur après:		m1 m2 m3 m4	Valeur de l'erreur après:	
	normalisation	produit scalaire		normalisation	produit scalaire
$m\ m$ $m\ m$	$\delta N < \frac{10,79}{2^m}$	$\delta PS < \frac{2,34}{2^{(m-3)}}$	$m+1\ m+1$ $m+1\ m+1$	$\delta N < \frac{6,54}{2^m}$	$\delta PS < \frac{1,42}{2^{(m-3)}}$
$m+2\ m+2$ $m+2\ m+2$	$\delta N < \frac{4,42}{2^m}$	$\delta PS < \frac{0,96}{2^{(m-3)}}$	$m+3\ m+3$ $m+3\ m+3$	$\delta N < \frac{3,36}{2^m}$	$\delta PS < \frac{0,73}{2^{(m-3)}}$
$m+4\ m+4$ $m+4\ m+4$	$\delta N < \frac{2,83}{2^m}$	$\delta PS < \frac{0,62}{2^{(m-3)}}$	$m+9\ m+9$ $m+9\ m+9$	$\delta N < \frac{2,31}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
$m+2\ m+2$ $m+2\ \ m$	$\delta N < \frac{4,79}{2^m}$	$\delta PS < \frac{1,04}{2^{(m-3)}}$	$m+3\ m+3$ $m+3\ m+1$	$\delta N < \frac{3,54}{2^m}$	$\delta PS < \frac{0,77}{2^{(m-3)}}$
$m+4\ m+4$ $m+4\ m+2$	$\delta N < \frac{2,92}{2^m}$	$\delta PS < \frac{0,64}{2^{(m-3)}}$	$m+9\ m+9$ $m+9\ m+7$	$\delta N < \frac{2,31}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
$m+3\ m+3$ $m+2\ \ m$	$\delta N < \frac{4,04}{2^m}$	$\delta PS < \frac{0,88}{2^{(m-3)}}$	$m+4\ m+4$ $m+3\ m+1$	$\delta N < \frac{3,17}{2^m}$	$\delta PS < \frac{0,69}{2^{(m-3)}}$
$m+5\ m+5$ $m+2\ \ m$	$\delta N < \frac{2,73}{2^m}$	$\delta PS < \frac{0,59}{2^{(m-3)}}$	$m+9\ m+9$ $m+8\ m+6$	$\delta N < \frac{2,32}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
$m+5\ m+2$ $m+2\ \ m$	$\delta N < \frac{3,98}{2^m}$	$\delta PS < \frac{0,87}{2^{(m-3)}}$	$m+6\ m+6$ $m+3\ m+1$	$\delta N < \frac{3,14}{2^m}$	$\delta PS < \frac{0,68}{2^{(m-3)}}$
$m+7\ m+4$ $m+4\ m+2$	$\delta N < \frac{2,72}{2^m}$	$\delta PS < \frac{0,59}{2^{(m-3)}}$	$m+11\ m+8$ $m+8\ m+6$	$\delta N < \frac{2,32}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
$m+9\ m+9$ $m+9\ m+9$	$\delta N < \frac{2,32}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$	$m+10\ m10$ $m+10\ m+4$	$\delta N < \frac{2,34}{2^m}$	$\delta PS < \frac{0,51}{2^{(m-3)}}$

Figure D•1 L'erreur pour différentes configurations.

m1 m2 m3 m4	Valeur de l'erreur après:		m1 m2 m3 m4	Valeur de l'erreur après:	
	normalisation	produit scalaire		normalisation	produit scalaire
$m\ m$ $m\ m$	$\delta N < \frac{9,90}{2^m}$	$\delta PS < \frac{2,15}{2^{(m-3)}}$	$m+1\ m+1$ $m+1\ m+1$	$\delta N < \frac{5,65}{2^m}$	$\delta PS < \frac{1,23}{2^{(m-3)}}$
$m+2\ m+2$ $m+2\ m+2$	$\delta N < \frac{3,53}{2^m}$	$\delta PS < \frac{0,77}{2^{(m-3)}}$	$m+3\ m+3$ $m+3\ m+3$	$\delta N < \frac{2,47}{2^m}$	$\delta PS < \frac{0,54}{2^{(m-3)}}$
$m+4\ m+4$ $m+4\ m+4$	$\delta N < \frac{1,94}{2^m}$	$\delta PS < \frac{0,42}{2^{(m-3)}}$	$m+3\ m+3$ $m+2\ \ m$	$\delta N < \frac{3,15}{2^m}$	$\delta PS < \frac{0,68}{2^{(m-3)}}$
$m+4\ m+4$ $m+3\ m+1$	$\delta N < \frac{2,28}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$	$m+7\ m+4$ $m+4\ \ m$	$\delta N < \frac{2,25}{2^m}$	$\delta PS < \frac{0,49}{2^{(m-3)}}$

Figure D•2 L'erreur avec extension de la dynamique d'entrée de l'unité de normalisation.

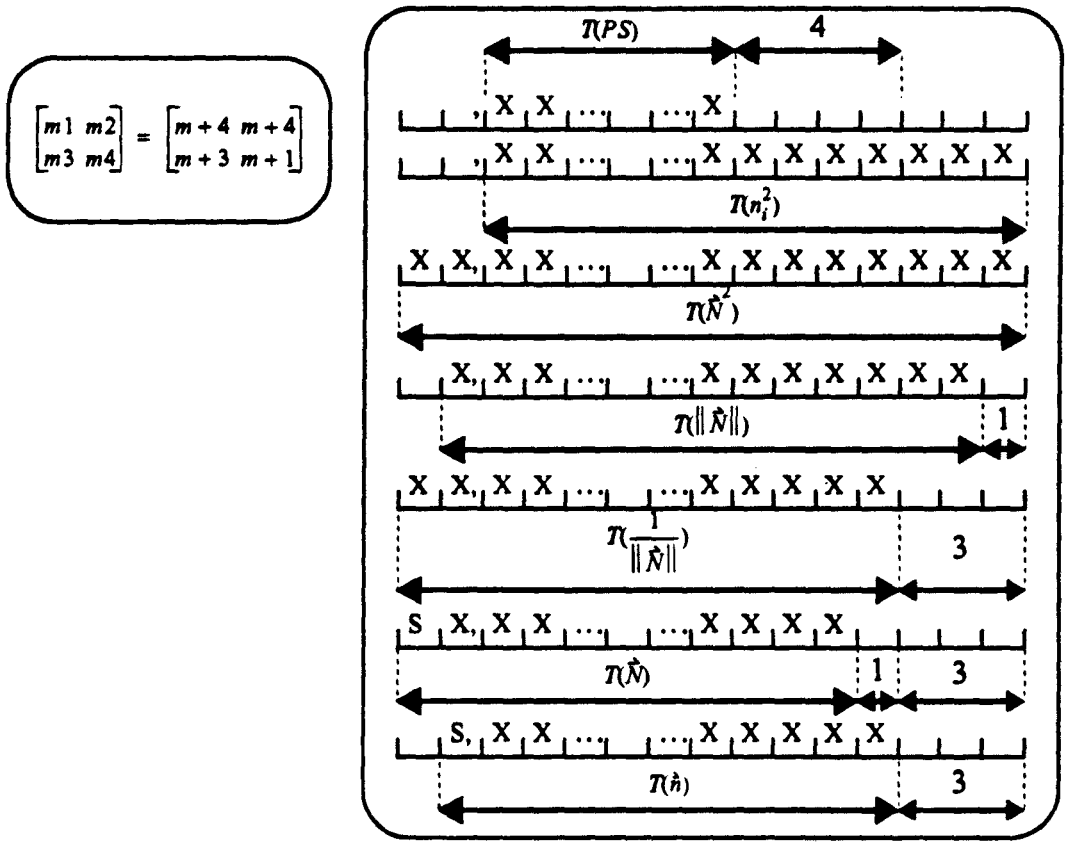


Figure D•3 Formats de données optimum.

D•2 Architecture avec unité d'inversion de la racine carrée.

m1 m2 / m4	Valeur de l'erreur après: normalisation	produit scalaire	m1 m2 / m4	Valeur de l'erreur après: normalisation	produit scalaire
m m / m	$\delta N < \frac{8,79}{2^m}$	$\delta PS < \frac{1,91}{2^{(m-3)}}$	m+1 m+1 / m+1	$\delta N < \frac{5,54}{2^m}$	$\delta PS < \frac{1,20}{2^{(m-3)}}$
m+2 m+2 / m+2	$\delta N < \frac{3,92}{2^m}$	$\delta PS < \frac{0,85}{2^{(m-3)}}$	m+3 m+3 / m+3	$\delta N < \frac{3,11}{2^m}$	$\delta PS < \frac{0,67}{2^{(m-3)}}$
m+4 m+4 / m+4	$\delta N < \frac{2,70}{2^m}$	$\delta PS < \frac{0,59}{2^{(m-3)}}$	m+8 m+8 / m+8	$\delta N < \frac{2,32}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
m+2 m+2 / m	$\delta N < \frac{4,29}{2^m}$	$\delta PS < \frac{0,93}{2^{(m-3)}}$	m+3 m+3 / m+1	$\delta N < \frac{3,29}{2^m}$	$\delta PS < \frac{0,72}{2^{(m-3)}}$
m+4 m+4 / m+4 m+2	$\delta N < \frac{2,79}{2^m}$	$\delta PS < \frac{0,61}{2^{(m-3)}}$	m+9 m+9 / m+7	$\delta N < \frac{2,31}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
m+3 m+3 / m	$\delta N < \frac{3,54}{2^m}$	$\delta PS < \frac{0,77}{2^{(m-3)}}$	m+4 m+4 / m+1	$\delta N < \frac{2,92}{2^m}$	$\delta PS < \frac{0,64}{2^{(m-3)}}$
m+5 m+5 / m+2	$\delta N < \frac{2,61}{2^m}$	$\delta PS < \frac{0,57}{2^{(m-3)}}$	m+9 m+9 / m+6	$\delta N < \frac{2,31}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$

Figure D•4 L'erreur pour différentes configurations.

m1 m2 m4	Valeur de l'erreur après: normalisation	produit scalaire	m1 m2 m4	Valeur de l'erreur après: normalisation	produit scalaire
m m m	$\delta N < \frac{7,90}{2^m}$	$\delta PS < \frac{1,72}{2^{(m-3)}}$	m+1 m+1 m+1	$\delta N < \frac{4,65}{2^m}$	$\delta PS < \frac{1,01}{2^{(m-3)}}$
m+3 m+3 m+3	$\delta N < \frac{2,22}{2^m}$	$\delta PS < \frac{0,48}{2^{(m-3)}}$	m+3 m+3 m+2	$\delta N < \frac{2,28}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
m+5 m+2 m+2	$\delta N < \frac{2,22}{2^m}$	$\delta PS < \frac{0,48}{2^{(m-3)}}$	m+6 m+2 m+1	$\delta N < \frac{2,25}{2^m}$	$\delta PS < \frac{0,49}{2^{(m-3)}}$
m+4 m+4 m	$\delta N < \frac{2,28}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$	m+6 m+3 m	$\delta N < \frac{2,25}{2^m}$	$\delta PS < \frac{0,49}{2^{(m-3)}}$

Figure D•5 L'erreur avec extension de la dynamique d'entrée de l'unité de normalisation.

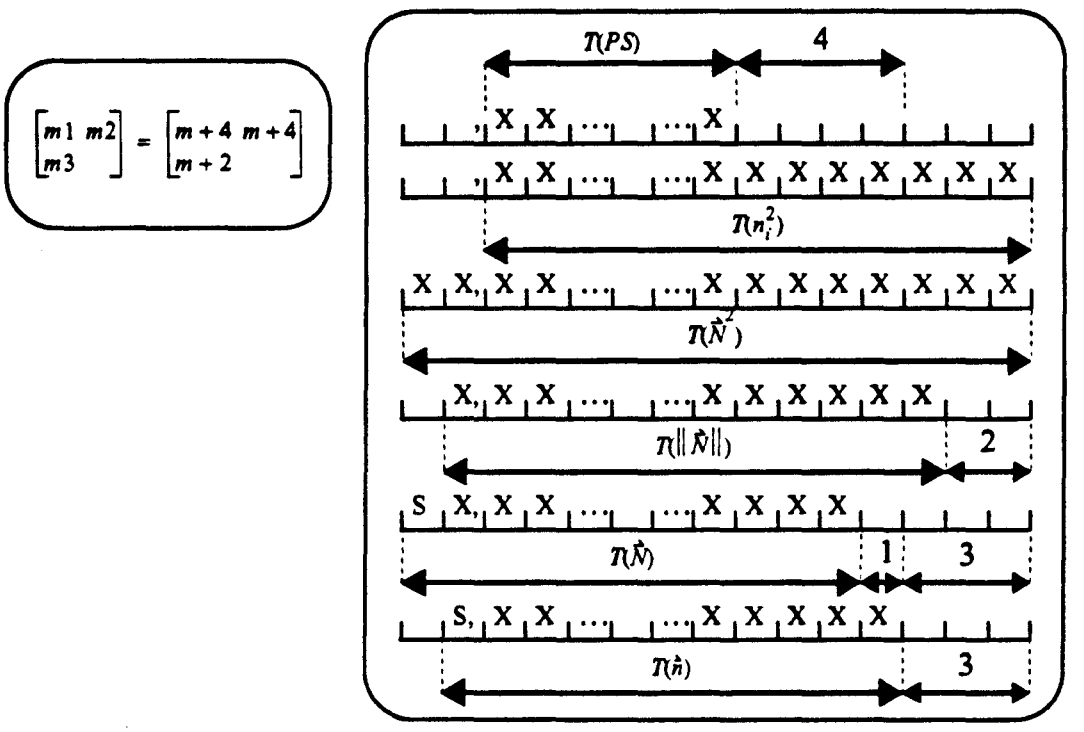


Figure D•6 Formats de données optimum.

D•3 Architecture avec unités de division.

m1 m2 m3	Valeur de l'erreur après: normalisation	produit scalaire	m1 m2 m3	Valeur de l'erreur après: normalisation	produit scalaire
m m m	$\delta N < \frac{10,29}{2^m}$	$\delta PS < \frac{2,23}{2^{(m-3)}}$	m+1 m+1 m+1	$\delta N < \frac{6,29}{2^m}$	$\delta PS < \frac{1,36}{2^{(m-3)}}$
m+2 m+2 m+2	$\delta N < \frac{4,29}{2^m}$	$\delta PS < \frac{0,93}{2^{(m-3)}}$	m+3 m+3 m+3	$\delta N < \frac{3,29}{2^m}$	$\delta PS < \frac{0,72}{2^{(m-3)}}$
m+4 m+4 m+4	$\delta N < \frac{2,79}{2^m}$	$\delta PS < \frac{0,61}{2^{(m-3)}}$	m+9 m+9 m+9	$\delta N < \frac{2,31}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
m+3 m+3 m	$\delta N < \frac{5,04}{2^m}$	$\delta PS < \frac{1,10}{2^{(m-3)}}$	m+4 m+4 m+1	$\delta N < \frac{3,67}{2^m}$	$\delta PS < \frac{0,80}{2^{(m-3)}}$
m+5 m+5 m+2	$\delta N < \frac{2,98}{2^m}$	$\delta PS < \frac{0.65}{2^{(m-3)}}$	m+11 m+1 m+8	$\delta N < \frac{2,30}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
m+5 m+2 m	$\delta N < \frac{4,98}{2^m}$	$\delta PS < \frac{1,08}{2^{(m-3)}}$	m+6 m+3 m+1	$\delta N < \frac{3,64}{2^m}$	$\delta PS < \frac{0,78}{2^{(m-3)}}$
m+7 m+4 m+2	$\delta N < \frac{2,97}{2^m}$	$\delta PS < \frac{0,65}{2^{(m-3)}}$	m+13 m+10 m+8	$\delta N < \frac{2,31}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$

Figure D•7 L'erreur pour différentes configurations.

m1 m2 m3	Valeur de l'erreur après: normalisation	produit scalaire	m1 m2 m3	Valeur de l'erreur après: normalisation	produit scalaire
m m m	$\delta N < \frac{9,40}{2^m}$	$\delta PS < \frac{2,04}{2^{(m-3)}}$	m+2 m+2 m+2	$\delta N < \frac{3,40}{2^m}$	$\delta PS < \frac{0,74}{2^{(m-3)}}$
m+3 m+3 m+3	$\delta N < \frac{2,40}{2^m}$	$\delta PS < \frac{0,52}{2^{(m-3)}}$	m+4 m+4 m+4	$\delta N < \frac{1,90}{2^m}$	$\delta PS < \frac{0,42}{2^{(m-3)}}$
m+4 m+4 m+3	$\delta N < \frac{2,03}{2^m}$	$\delta PS < \frac{0,44}{2^{(m-3)}}$	m+4 m+4 m+2	$\delta N < \frac{2,28}{2^m}$	$\delta PS < \frac{0,50}{2^{(m-3)}}$
m+6 m+3 m+2	$\delta N < \frac{2,25}{2^m}$	$\delta PS < \frac{0,49}{2^{(m-3)}}$	m+6 m+2 m+3	$\delta N < \frac{2,25}{2^m}$	$\delta PS < \frac{0,49}{2^{(m-3)}}$

Figure D•8 L'erreur pour différentes configurations avec extension de la dynamique d'entrée de l'unité de normalisation.

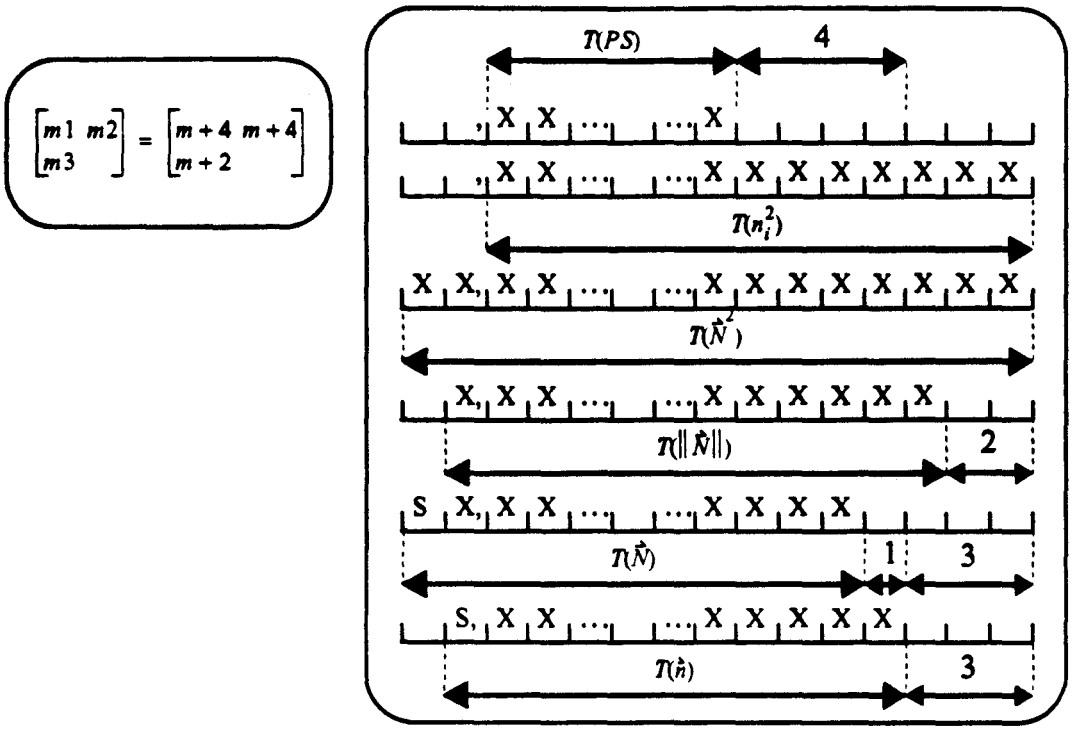


Figure D•9 Formats de données optimum.

Références

[1] ANDRES E.
"Cercles discrets et Rotations Discrète"
Thèse doctorat, Université Louis Pasteur, Strasbourg, Novembre 1992.

[2] BISHOP G. and WEIMER D.M.
"Fast Phong Shading"
ACM Computer Graphics, août 1986, pp 103-106.

[3] BLINN J.F., NEWELL M. E.
"Texture and reflection in Computer Generated Images"
Communications of ACM, vol. 19 n° 10, octobre 1976, pp 542-547.

[4] BLINN J. F.
"Computer Display of Curved Surfaces"
PhD thesis, University of Utah, Department of Computer Science, december 1978.

[5] BRESENHAM J.E.
"Algorithm for Computer Control of a Digital Plotter"
IBM Systems Journal, 4(1), 1965 , pp 25-30.

[6] CATMULL E.
"Computer Display of Curved Surfaces"
Proc. of IEEE Conference Computer Graphics Pattern Recognition Data Structure, mai 1975, p 11.

[7] CHAILLOU C.
"Etude d'un processeur de Visualisation d'Images de Synthèse en Temps Réel Expotant un Parallélisme Massif Objet: le Projet I.M.O.G.E.N.E."
Thèse de Doctorat, Université de Lille, Janvier 1991.

[8] CLAUSSEN U.
"On Reducing the Phong Shading Method"
Proc. Eurographics'89, Elsevier Science Publishers B.V., 1989, pp 333-344.

[9] CLAUSSEN U.
"Verfahren zur schnellen Beleuchtungs und Schattierungsberechnung"
Thèse de doctorat, Université deTubingen,1990 .

[10] CLAUSSEN U.
"Real Time Phong Shading"
Advances in Computer Graphics V, Springer Verlag, to appear in 1991.

[11] COHEN M. and GREENBERG D.
"The Hemi-Cube, a Radiosity Solution for Complex Environment"
ACM Computer Graphics vol. 19 n° 3, Juillet 1985, pp 31-40.

[12] CYPRESS SEMICONDUCTOR
"SPARC RISC USER'S GUIDE"
Data Book, Ross Technology Subsidiary, Second Edition, février 1990.

[13] DEERING M., WINNER S., SCHEDIWY B. et al.
"The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics."
ACM Computer Graphics, vol. 22 n° 4, août 1988, pp 21-30.

[14] DELFOSSE J.
"Discrétisation d'Objets Graphiques"
Mémoire de DEA, Université de Lille, juillet 1993.

[15] FUCHS H. et POULTON J.
"A VLSI Oriented Design for a Raster Graphics Engine"
VLSI Design, n° 3, 1981, pp 20-28.

[16] FUCHS H., GOLDFEATHER J., HULTQUIST J. et al.
"Fast Spheres,Shadows, Testures, Transparencies and Image Enhancements in Pixels Planes"
ACM Computer Graphics, vol. 16, n° 13, Juillet 1985, pp 11-120.

[17] FUCHS H., POULTON J., EYLES J. et GREER T.
"Coarse-grain and fine-grain parallelisme in the next generation Pixel-planes graphics system"
rapport TR88-014, Université de Caroline du Nord, mars 1988.

[18] FUCHS H., GOLDFEATHER J., HULTQUIST J. et al.
"Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories."
ACM Computer Graphics, vol. 23 n° 3, Juillet 1989, pp 79-88.

[19] FOLEY J., VAN DAM A., FEINER S. et HUGHES J.
"Computer Graphics: Principes and Practice (second edition)"
The systems Programming Series,Addison Wesley 12110,1990.

[20] FROUMENTIN M. et CHAILLOU C.
"Déformation interactive de patchs quadriques"
Les journées de l'AFIG 1993, Bordeaux, France, décembre 1993.

[21] GLASSNER A.S.
"An introduction to Ray Tracing"
Academic Press 1989.

[22] GOLDFEATHER J. et FUCHS H.
"Quadratic Surface Rendering on a Logic-Enhanced Frame-Buffer Memory"
IEEE Computer Graphics and Applications, vol. 6, n° 1, janvier 1986, pp 48-59.

[23] GORAL C., TORRANCE K., GREENBERG D., BATTAILE B.
"Modeling the Interaction of Light Between Diffuse Surfaces"
ACM Computer Graphics, vol. 18 n° 3, Juillet 1984, pp 213-222.

[24] GOURAUD H.
"Continous Shading of Curved Surfaces"
IEEE Transaction on Computers, vol. C-20 n° 6, juin 1971, pp 623-629.

[25] HASHEMIAN R.
"Square Rooting Algorithms for integer and floating Point Numers"
IEEE Transaction on Computer, vol. 39 n° 8, août 1990, pp 1025-1029.

[26] HE X.D., HEYNEN R.L., PHILLIPS R.L., TORRANCE K.E, and al.
"A Fast and Accurate Light Reflection Model"
Proc. SIGGRAPH'92 on Computer Graphics, pp 253-254.

[27] INTEGRATED DEVICE TECHNOLOGIE, INC
"Specilized Memories 1990/91"
Data Book, Integrated Device Technologie Inc, ∫dt, 1990.

[28] KARPF S.
"Architectures Massivement Parallèles pour la Synthèse d'Images Temps Réel"
Thèse de Doctorat,Université de Lille, janvier 1993.

[29] KARPF S., CHAILLOU C., NYIRI E. and MERIAUX M.
"Real-time Display of Quadrics Objects in the I.M.O.G.E.N.E. Machine"
ACM Symposium on Solid Modeling Foundations and CAD/CAM Application, Austin, 5-7 juin 1991.

[30] KAUFMAN A.
"Volume Visualisation"
IEEE Computer Society Press Tutorial, 1990.

[31] KNITTEL G.
"A VLSI-Design for fast Vector Normalization"
Proc. 8th Eurographics Workshop on Graphics Hardware, Barcelone, 6-7 septembre 1993.

[32] LAPORTE H.
"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.

[33] LEFER W.
"Etude de la parallélisation de l'algorithme du lancer de rayon en synthèse d'images."
Thèse de Doctorat, Université de Caen, septembre 1992.

[34] LEFEVERE V., KARPF S., CHAILLOU C. and MERIAUX M.
"The I.M.O.G.E.N.E. Machine: Some Hardware Elements."
Proc. 6th Eurographics Workshop on Graphics Hardware, Vienne, 1-2 septembre 1991.

[35] LEFEVERE V., CHAILLOU C. and MERIAUX M.
"Low-cost hardware for real time Phong Lighting"
Proc. Graphics Interface 92 Workshop on Local Illumination, Vancouver B.C., mai 1992, pp 23-30.

[36] LUCAS M.
"Parallélisme et synthèse d'image"
TSI, vol. 10 n° 3, 1991, pp 171-202.

[37] MERIAUX M.
"Contribution à l'Imagerie Informatique: Aspects Algorithmiques et Architecturaux"
Thèse d'Etat, Université de Lille I,1984.

[38] NEHLIG P.
"Applications affines discrètes et antialiassage"
Thèse de doctorat, Université Louis Pasteur, Strasbourg, Novembre 1992.

[39] NICODEMUS F.E., RICHMOND J.C. and HSIA J.J.
"Géometrical Considerations and Nomemclature for reflectance"
U.S. Departement of Commerce, National Bureau of Standard, octobre 1977.

[40] NYIRI E.
"Modélisation et Simulation d'Objets 3D à l'Aide d'Expressions du Second Degré"
Mémoire de DEA, Université de Lille, septembre 1990.

[41] NYIRI E.
"Etude d'une nouvelle primitive d'affichage pour les machines de rendu classique à parallélisme objet: la quadrique"
Thèse de Doctorat, Université de Lille, à soutenir en février 1994.

[42] NYIRI E. and CHAILLOU C.
"Aspect logiciel du project I.M.O.G.E.N.E"
MICAD 92, Paris, 11-14 février 1992, pp 201-218.

[43] PITOT P.
"The Voxar Project."
IEEE Computer Graphics and Applications 13, 1 janvier 1993, pp 27-33.

[44] PHONG B. T.
"Illumination for Computer Generated Pictures"
Communications ACM, vol. 18 n° 18, juin 1975, pp 311-317.

[45] POULIN P. and FOURNIER A.
"A model for anisortopic reflection"
Proc. SIGGRAPH'90 on Computer Graphics 24, n° 4, pp 273-282.

[46] RENAUD C.
"Approches paralléles pour la radiosité"
Thèse de doctorat, Université de Lille, octobre 1993

[47] ROGERS D. F.
"Algorithmes pour l'infographie"
"Procedural elements for computer graphics"
McGRAW-HILL, 1988.

[48] SCHLICK C.
"Diverses optimisations pour le calcul du facteur de réflectance"
Journées Graphiques GROS PLAN 91, Lille, France, décembre 1991, pp 39-46.

[49] SCNEIDER B.O.
"A Processor for an Object-Oriented Rendering System"
Computer Graphics Forum,n° 7, 1988, pp 301-310.

[50] SCNEIDER B.O. et CLAUSSEN U.
"PROOF: An Architecture for Rendering in Object Space."
Advances in Computer Graphics Hardware III,Springer Verlag, 1988, pp 121-140.

[51] SILLION F. et PUECH C.
"A general two-pass method integrating specular and diffuse reflection"
SIGGRAPH'89, vol 23, n° 4, août, pp 335-344.

[52] SWARTZLANDER E.E. and ALEXOPOULOS A.G.
"The Sign/Logarithm Number System"
IEEE Transactions on Computer, vol. 24 n° 12, décembre 1975, pp 1238-1242.

[53] TEXAS INSTRUMENTS
"TMS 320C4X User's Guide"
Data Book, revision A, mai 1991.

[54] VIDAL B.
"Vers un lancer de rayons discret"
Thèse de Doctorat,Université de Lille, février 1992.

[55] WARD G.J.
"Towards More Pratical Reflectance Measurements and Models"
Proc. Graphics Interface 92 Workshop on Local Illumination, mai 1992, pp 15-22.

[56] WESTIN S.H., ARVO J.R. and TORRANCE K.E.
"Prédicting Reflectance Functions from Complex Surfaces"
Proc. SIGGRAPH'92 on Computer Graphics, pp 255-264.

[57] WHITTED T.
"An Improved Illumination Model for Sheded Display"
Communication ACM, vol 23, 1980, pp 343-349.