CHAPITRE VIII

Logiciels annexes au lancer de rayon

Mots clés

Modélisation, textures, parallélisme.

Résumé

Conjointement au développement du programme de lancer de rayon, un certain nombre de modules ou de logiciels ont été développés. Ce chapitre explique rapidement les fonctionnalités de ces logiciels.

Summary

In this chapter, we describe the functionality's of different softwares that have been developed with our ray-tracer.

Introduction

Dans le cadre de la réalisation de notre programme de lancer de rayon, plusieurs aspects connexes ont dû être abordés pour aboutir à un environnement complet permettant la génération d'images. En effet, un tel programme ne peut être pleinement utilisé sans logiciels permettant de créer des scènes variées et complexes et les calculs ne peuvent être distribués sur un réseau hétérogène sans outils de parallélisation. Enfin, dans le but d'enrichir le rendu de nos scènes, des procédures de texture ont été développées.

I) La modélisation

I.1) Ébauche d'un modeleur graphique

Antérieurement à cette thèse, un petit modeleur graphique avait été développé. Il permettait de placer des objets de base dans une scène en construction, de saisir leurs paramètres photométriques et de visualiser le résultat en fil de fer. À partir de cette expérience, différentes conclusions ont été tirées :

- les modeleurs simples de ce type sont très faciles à prendre en main mais ne permettent pas de générer des scènes très évoluées,

- a contrario, ces logiciels peuvent être très enrichis mais dans ce cas, leur utilisation devient complexe et d'un point de vue de la programmation, il n'est pas raisonnable dans le cadre d'une thèse sur le rendu de vouloir développer un tel programme,

- enfin, point qui nous semble très important, un grand nombre de scènes sont très facilement générées par un petit programme (par exemple le flocon de sphères), alors qu'il est extrêmement difficile de les produire "à la main".

I.2) Fichier d'échange de données

Les modeleurs produisent des résultats sous la forme d'un fichier standard qui sera relu par le logiciel de rendu. Ce fichier peut être modifié directement, mais cette opération n'est pas aisée car la description de la scène se présente généralement sous la forme d'une longue liste de chiffres. La structure du fichier que notre lancer de rayon lit est la suivante :

```
# tous ce qui commence par # est un commentaire
# entête générale du fichier
nombre d'objets
couleur ambiante de la scène
#description de chaque objet
type de l'opération (ajouter, modifier, détruire)
numéro de l'objets lors de la sauvegarde
type de l'objet (cube, …)
numéro d'identification de l'objet
paramètres géométriques pour la radiosité
# Si l'objet n'est pas une facette, on donne les matrices :
matrice 4x4 directe
matrice 4x4 inverse
```

```
émittance
réflectance
couleur intrinsèque de l'objet
diffusion lambertienne
transparence
indice de réfraction
diffusion spéculaire
indice n de Phong
booléen indiquant si l'objet est une lumière
numéro de la texture
nom du fichier bitmap de la texture
# si l'objet est une facette on donne :
3 points décrivant une facette
tableau de points d'une courbe réglée
#fin de la description de l'objet
marque de fin du fichier
```

Certains champs ont des valeurs par défaut lorsqu'ils ne sont pas renseignés. On notera la présence de deux matrices inverses et directes. Elles sont stockées pour réduire les arrondis qui pourraient apparaître lors de l'inversion à partir d'une seule matrice.

I.3) Réalisation d'un compilateur pour la modélisation

Suite à la réalisation du modeleur graphique, une approche par la compilation a été testée. Une grammaire simple a été définie et le compilateur correspondant réalisé en lex et yacc. Ce petit langage (SMILE1.0) était purement déclaratif, il ne permettait pas d'utiliser des variables. Par contre, il était possible (et relativement facile) de générer des scènes variées pouvant contenir jusqu'à une centaine d'objets, le seul problème étant le temps qu'il fallait pour écrire un tel fichier et sa longueur. À partir de cette version, des spécifications ont été produites pour palier certains défauts de cette première version : absence de variables, de structures de contrôle, de boucles et de procédures. Ces spécifications ont abouti à la réalisation d'un second langage [AA94]. De ces expériences, nous pouvons tirer les conclusions suivantes :

- L'avantage du premier langage était d'être intégré au logiciel de rendu. La grammaire analysait donc le programme de description de la scène et instanciait directement les données du lancer de rayon. Cependant, cela augmentait la taille finale du programme.

- La seconde version séparait nettement les deux aspects ce qui permettait de développer séparément la modélisation du rendu. Or il nous semble que dans l'optique d'une recherche de la meilleure chaîne de l'image possible, cet aspect est essentiel.

- Ces deux langages souffraient d'un certain nombre de manques :

* absence de certaines structures de données,

* pas de réelle approche de programmation par objet,

* lenteur intrinsèque de l'analyse LEX/YACC,

* absence d'analyse du code de la scène permettant de l'optimiser,

* absence de librairies classiques permettant d'accéder facilement au système, périphériques, …

Tous ces points peuvent être résolus. Cependant, le travail à faire est alors un travail long et complexe de compilation et il n'était pas possible d'envisager de passer plusieurs mois voire plusieurs années à développer une autre version du langage. C'est pourquoi, une troisième approche a finalement été adoptée : l'extension du langage C ANSI par une bibliothèque.

I.4) Réalisation d'une bibliothèque C

Cette approche résout en fait la grande majorité des problèmes pré-cités. Seule la demande d'une programmation par objet n'a pas été totalement satisfaite par souci de portabilité maximale du code. Le programme réalisé (bien qu'à ce jour non totalement abouti) s'appelle EMILE et a été développé en coopération avec M. S. Michelin.

I.4.a) Caractéristiques du programme

- Le langage manipule des entités abstraites : les objets.

- Ces objets sont, informatiquement, soit une structure simple, soit un ensemble d'objets.

- La scène qui est un objet, peut-être soit un arbre CSG soit une liste.

- Chaque objet élémentaire, une fois déclaré, reçoit un type (cube, sphère, facette, courbe réglée,…), une géométrie, une photométrie.

- Une fois déclaré et instancié, un objet peut être manipulé comme une entité indépendante ou rattaché à la scène.

Les objets peuvent être définis de différentes façons. Cependant, il nous a paru important de respecter les conventions suivantes :

- la majorité des objets sont définissables par un centre, des dimensions en x,y,z et une matrice de rotation ou encore par une unique matrice regroupant ces trois caractéristiques et appliquée à un objet dit canonique,

- tous les objets canoniques ont un centre qui doit appartenir à l'ensemble des points tels que le nombre de symétries est maximal (dans le cas des objets simples pavés, ellipsoïdes,… ce point est unique) et ce centre canonique est confondu avec l'origine du repère,

- les dimensions (mise à l'échelle) des objets canoniques se définissent à partir de rayon d'une unité.

- la caméra est un objet particulier géométriquement défini par une pyramide à base carrée, dont le sommet est l'œil d'un observateur fictif, et dont le centre de la base carrée est l'origine. L'observateur regarde donc dans une direction fixée par défaut (ici l'axe des y, vers les positifs). Cette direction, ainsi que la taille de l'écran peuvent être changées grâce aux opérateurs géométriques définis pour tous les autres objets (échelle, rotation, translation).

Ces règles simples garantissent qu'une matrice appliquée à un cube aura la "même signification" qu'appliquée à un autre objet, c'est à dire que seuls les éléments géométriques voulus changeront, pas la géométrie générale et encore moins la topologie (par une matrice de déformation, il n'y a pas nécessairement conservation des angles, les homotéties ne donnent pas les mêmes résultats selon le point pris comme origine, etc…).

I.4.b) Un serveur interactif de base de données

EMILE n'est pas seulement un modeleur, il a aussi pour vocation d'être un serveur (simple) de base de données et une interface de conversion de formats de fichier. Six possibilités d'action avec les logiciels de rendu ont été définies. Le logiciel est capable de donner l'ordre à ces logiciels de :

- DÉTRUIRE un objet,
- d'AJOUTER un objet
- de MODIFIER un objet.

Ces actions apparaissent dans la description du fichier d'échange de données présenté plus haut.

Parallèlement, le logiciel de rendu est autorisé à :

-DEMANDER UN OBJET donné à EMILE,

- DEMANDER UNE SCÈNE précise.

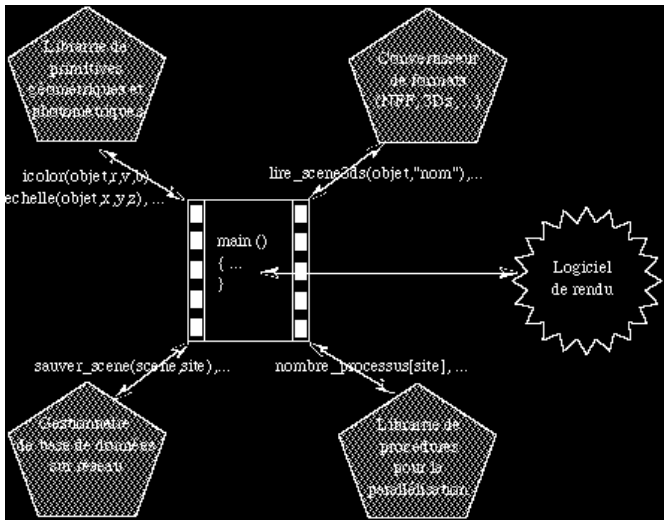

Figure VIII-1

Enfin, la possibilité de lire une action générée par un acteur extérieur a été envisagée et est relativement simple à mettre en œuvre mais n'a pas été testée (Cf. Figure VIII-1).

Ces caractéristiques permettent de créer des images statiques sans difficultés, mais également de créer des séquences d'images en gardant la cohérence des informations (le $n^{ième}$ objet de l'image 1 sera toujours le $n^{ième}$ à l'image 2).

II) La parallélisation

II.1) Information du parallélisme dans le serveur de base de données

Le programme décrivant la scène donne également différentes informations indispensables pour la parallélisation :

- nombre de processus de l'écran,
- affectation d'un groupe de processus à une machine donnée,
- nom de la machine sur le réseau,
- nom de l'image à sauvegarder,
- répertoire d'arrivée du fichier de description de la scène.

Lorsqu'EMILE a généré son fichier de sortie contenant les objets et les informations du parallélisme pour une machine donnée, il se charge d'envoyer cette description dans le répertoire précisé pour la machine destinataire.

II.2) Programme de distribution des calculs

Un programme général de distribution des tâches à effectuer englobe les différents aspects de l'exécution finale (Cf. Figure VIII-2) :

- gestion des droits d'exécution, de lecture et de copie,
- lancement du modeleur,
- éventuelle compilation sur le réseau du programme de rendu,
- envoi des fichiers nécessaires aux différents sites,
- exécution du programme de rendu sur les différentes machines,
- affichage de l'image finale,
- terminaison du programme parallèle.

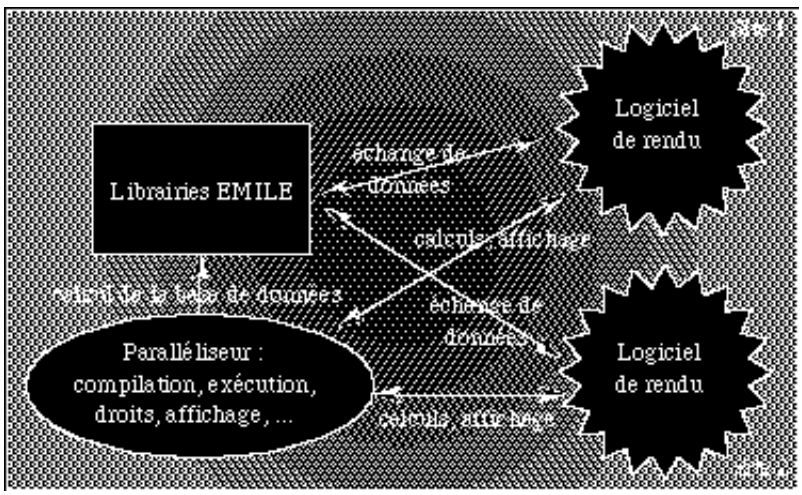

Figure VIII-2 : schéma de distribution des calculs

III) Les textures

Les textures sont un élément essentiel pour la variété des images de synthèse. Cette thèse n'était pas consacrée aux textures mais nous avons dû y travailler un peu et un certain nombre de conclusions ont été tirées.

III.1) Définition

On appelle plaquage de texture toute technique visant à dessiner un objet de manière à ce que les surfaces apparaissant sur cet objet semblent être revêtues d'une image. Ce problème revient à attribuer une teinte à tout pixel appartenant à un objet non plus en fonction du modèle général d'illumination mais en fonction de ce modèle augmenté d'une fonction de plaquage de la texture considérée. On peut distinguer les textures dites "solides" qui varient selon les trois dimensions de l'espace des autres textures qui sont implicitement surfacique.

III.2) Propriétés générales des textures

P_1) Les textures sont des applications à plusieurs dimensions. À un ensemble de paramètres, elles font correspondre une couleur.

P_2) L'application de texture doit respecter les couleurs originelles de l'objet, c'est à dire qu'un objet texturé rouge doit apparaître plus rouge qu'un objet identique mais blanc).

P_3) L'application de texture doit respecter les ombres.

P_4) L'application de texture doit respecter les effets spéculaires.

P_5) L'application de texture doit avoir un résultat reproductible. Ce point est indispensable pour rendre les réflexions ou les transmissions réalistes.

III.3) Textures matrice de pixels (bitmap)

On considère un espace 2D discret de n*m éléments. Chacun de ces éléments correspond à un pixel (une couleur C) d'une image numérisée I (la matrice de pixels). On considère de plus la surface S d'un objet O.

Définition : une texture matrice de pixels est une application T_b qui à tout point de S fait correspondre un pixel de I de couleur C.

$$T_b : S \rightarrow I$$
$$x,y \text{ a } C = T_b(x,y)$$

T_b n'est ni injective ni surjective dans le cas général (deux points de S très proches peuvent correspondre à un même pixel de I et inversement, un pixel de I peut être l'image de plusieurs points de S - effet de zoom -).

Notre programme prend en compte toutes les surfaces des objets que nous avons définis (bien que la correspondance 3D vers 2D soit parfois très simplifiée).

NB : Le fait que plusieurs éléments de texture (ou texels) peuvent s'appliquer sur le même pixel est un phénomène appelé "compression de la texture" : cette compression est l'un des facteurs importants qui expliquent les phénomènes d'alliassage. Des techniques de filtrage issues de la théorie du signal sont une autre approche pour résoudre les problèmes d'alliassage qui apparaissent lors du placage de texture (ils ne sont pas développés ici).

III.4) Textures mathématiques

Le nombre de paramètres des textures mathématiques T_m peut être très variable. En général, T_m utilise en entrée une position x,y,z dans l'espace (le point d'impact) et une couleur (r,v,b) de départ. Cependant, rien n'interdit d'y ajouter le vecteur normal ou tout autre paramètre.

$T_m : R^n \rightarrow$ Ensemble des couleurs

$$x,y,z,r,v,b,\ldots \text{ a } T_m(x,y,z,r,v,b,\ldots)$$

T_m n'est ni injective ni surjective dans le cas général.

Différentes familles de textures mathématiques ont été testées :

- les textures fractales : Julia et Mandelbrot,
- les textures chaotiques : itération sur l'équation logistique $y_{n+1} = r\ (y_n - 1)$ et sur l'équation de Hénon,
- les combinaisons de fonctions mathématiques de base (trigonométriques, modulo, …),

- des fonctions de bruit selon Perlin [Pe85]. Schématiquement, les fonctions de bruits définies par Perlin ont la propriété de générer des images qui semblent aléatoires tout en gardant une cohérence (on ne génèrent pas des points chaotiques). Cette technique utilise deux fonctions : une fonction appelée turbulence qui est la sommation d'une seconde fonction appelée bruit qui elle-même est une sommation pondérée de termes d'un tableau initialisé aléatoirement.

L'intérêt des textures mathématique est qu'elles permettent des zooms sur les objets sans avoir d'effet de pixellisation si on se rapproche à une distance telle qu'un pixel de l'image source corresponde à plus d'un pixel à l'écran. En revanche, toute image n'est pas stockable sous la forme d'une texture mathématique. Ces applications donnent des résultats très variés mais les plus jolis résultats sont souvent le fruit de l'empirisme ou du hasard.

III.5) Textures par variation de la normale ("bump mapping")

Ces applications sont des moyens simples de générer des effets de pseudo-relief par une simple perturbation de la normale et par conséquent va modifier le calcul d'illumination en un point donné. Cette perturbation peut être une fonction sinus par exemple, mais n'importe quelle fonction peut être utilisée.

$$T_v : R^3 \rightarrow \text{Ensemble des couleurs}$$

$$u,v,w \quad a \quad T_v(u,v,w)$$

Des études ont été menées pour évaluer de vraies textures 3D mais celles-ci demandent de modifier le programme de lancer de rayon lui-même (introduction de méthodes incrémentales par exemple). Ces textures sont donc d'une nature très différente.

III.6) Textures spéciales

Enfin, nous avons implanté deux textures qui ne semblent pas totalement classiques :

- texture de calcul récursif de la scène : l'idée est celle du camescope qui filme son écran. Lorsqu'un rayon heurte un objet avec cette texture récursive, on relance un calcul de lancer de rayon comme si cet objet était un écran.

- textures appliquées à tous les paramètres photométriques : ici, plutôt que de considérer qu'un objet a une couleur diffuse R,V,B constante par exemple, on considère que cette couleur diffuse est donnée par un calcul de texture sur ce paramétres photométrique précis. L'idée est relativement simple.

Conclusion

Les trois éléments librairies EMILE, programme de distribution des calculs, et textures, sans être fondamentaux, représentent trois outils qui améliorent grandement la qualité technique du logiciel de rendu présenté dans ce mémoire.

Bibliographie

[AA94] : ARQUES (D.), AUBERT (S.). - "Modélisation de scènes tri-dimensionnelles pour l'animation en synthèse d'images : notion d'arbres ECSG et compilation", Journées AFIG-GROPLAN, Toulouse, 1994, p. 273-284.

[Pe89] : PERLIN (K.) - "Hypertexture", SIGGRAPH'89 conference procedings, 23(3), 1989, p. 253-262.

Chapitre IX

Calcul de séquences d'images

Mots clés

Animation, temps réel.

Résumé

L'animation n'est pas le sujet de cette thèse mais c'est un prolongement logique. En suivant la même ligne directrice de réutilisation des connaissances acquises, nous exposons une dernière série de règles qui optimisent les calculs d'une séquence d'images. Des tests illustrent l'intérêt de cette approche.

Summary

Animation is not the subject of this Ph.D. but it is a logical extension. By using the same idea of improvements deduced from knowledge acquisition, different new rules are exposed : they optimize calculation of a sequence of images. Some tests show the interest of this approach.

Introduction

S'il est très important de pouvoir calculer des images d'un grand réalisme en un temps court, beaucoup d'applications demandent d'aller plus loin, c'est à dire vers le temps réel. L'animation est un prolongement naturel du calcul d'images statiques mais elle exige la mise en œuvre de techniques supplémentaires. Parmi toutes ces techniques, il est intéressant de savoir si certaines ne pourraient pas réutiliser l'approche de l'optimisation exposée dans ce mémoire, à savoir la réutilisation de connaissances déjà acquises.

I) Extension des règles à l'animation

I.1) Les boîtes englobantes

Dans une image statique, le programme pré-calcule une boîte englobante pour chaque objet puis, par les règles de déduction, ces boîtes sont dynamiquement affinées en cours de calcul.

Règle C_{1a} : si un objet ne subit pas de modifications géométriques entre deux images, sa boîte englobante reste inchangée (cette règle est évidente).

C_{1a} permet donc d'optimiser les pré-calculs ainsi que les déductions dynamiques (on ne cherche plus à les faire pour cet objet).

Règle C_{1b} : si un objet subit une modification géométrique dans une direction identifiée, la boîte englobante ne pourra être modifiée que dans cette direction.

Cette seconde règle permet de tenir compte des affinements dynamiques de l'étape précédente. En effet, si un objet se déplace vers l'OUEST, la boîte englobante aura au pire, la même extrémité EST qu'à la fin de l'étape précédente (Cf. Figure IX-1).

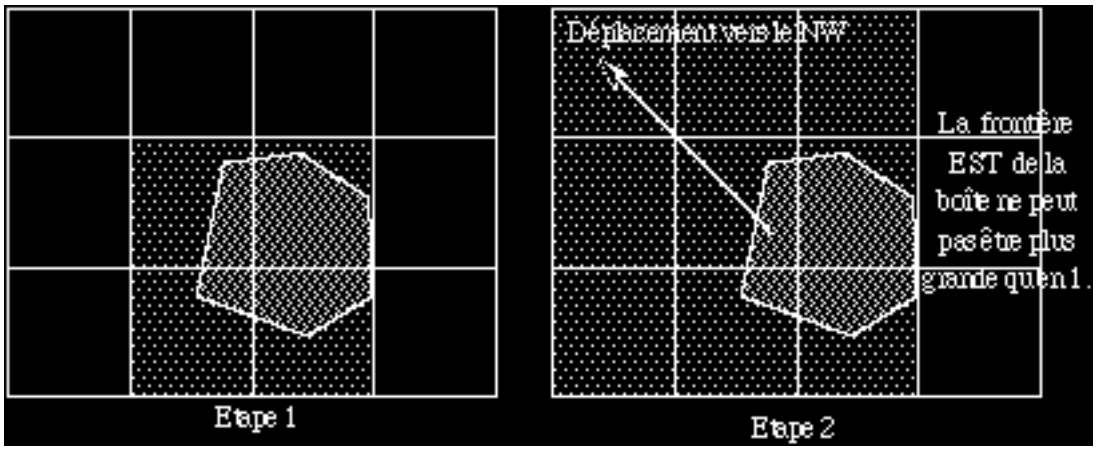

Figure IX-1 : règle C_2

I.2) Les ombres

Règle C_{2a} : pour un couple (source de lumière, objet), si aucun des deux éléments n'est modifié entre deux images, les cônes d'ombre inscrits et circonscrits restent inchangés.

Règle C_{2b} : si un objet a fait de l'ombre à un autre lors du calcul de l'image n, cet objet a une forte probabilité de faire à nouveau de l'ombre lors du calcul de l'image n+1 (on ne réinitialise pas les variables concernant cette optimisation).

I.3) Diffusion des messages

Règle C_3 : sous l'hypothèse que le déplacement d'un objet apparaît continu à l'écran, si on ne ré-initialise les messages finis que sur les zones recouvertes par les nouvelles boîtes ainsi que pour les voisins immédiats on n'aura à diffuser qu'un très petit nombre de nouveaux messages (Cf. Figure IX-2).

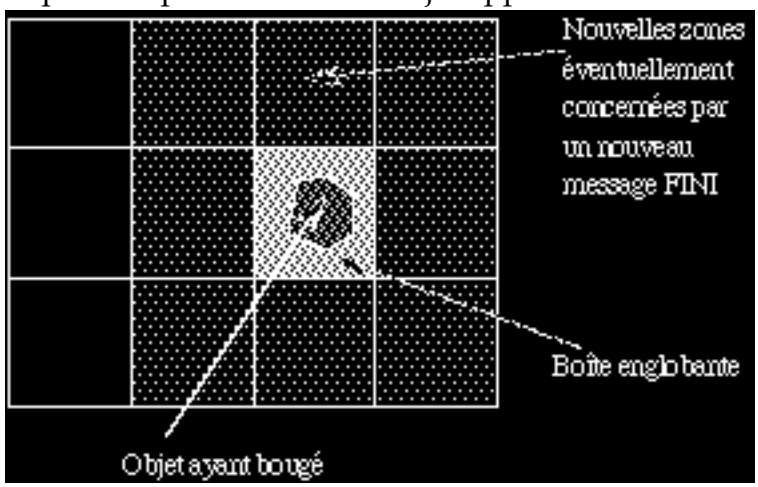

Figure IX-2 : règle C_3

Cette règle nous permet lors du calcul d'une séquence d'images de faire disparaître l'un des points négatifs de notre algorithme : le nombre important de messages qui sont générés. Une fois la première image calculée, la gestion des déductions sera très fortement optimisée.

I.4) Optimisations annexes

L'algorithme de pré-tri est plus performant car il travaille sur une liste déjà fortement triée.

La phase de lecture de la scène et d'allocations mémoire est grandement améliorée car on ne lit que les informations des objets qui ont été modifiés, et on n'alloue de la mémoire que pour les objets qui apparaissent éventuellement (on ne désalloue de même que la mémoire de ceux qui disparaissent).

I.5) Bénéfices issus de ces optimisations

On constate donc, que pour l'animation :

1) la phase de pré-calcul est simplifiée et bénéficie en partie des résultats déjà acquis,

2) tous les rayons vont bénéficier du fait qu'une grande partie des boîtes englobantes sont optimales au moment où l'on relance l'algorithme,

3) les calculs d'ombrage bénéficient en plus de la connaissance des anciens objets faisant de l'ombre ce qui permet de re-tester certains objets en priorité,

4) la gestion des optimisations est très fortement améliorée,

5) la liste des objets primaires est presque optimale pour une grande partie des processus.

6) Le nombre de messages échangés diminuent fortement.

II) Test

II.1) Configuration

L'animation test est celle de la Figure IX-3 : une image est constituée de 4 objets plus 1 lumière (le paraboloïde est texturé, la sphère est réfléchissante).

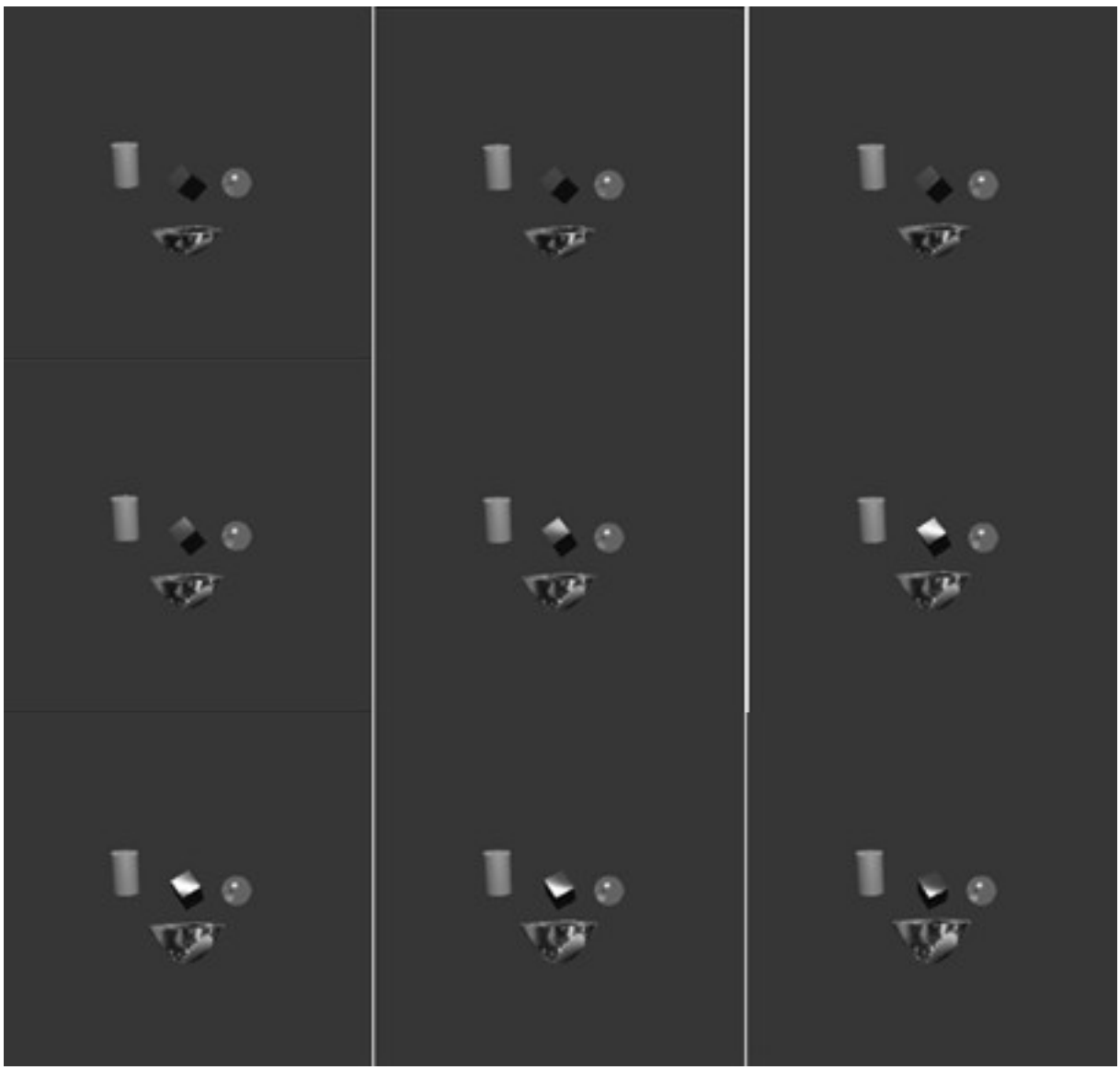

Figure IX-3 : Calcul d'une séquence d'images

On calcule une séquence de 25 images en 512*512 avec 1024 processus en parallélisme simulé sur la RS6000-320 en changeant les paramètres géométriques et/ou photométriques de deux des objets (dans une animation, nous avons la possibilité de faire changer tous les paramètres y compris le type de l'objet à condition de respecter l'hypothèse que l'objet modifié soit plus petit ou qu'il ne grossisse pas en augmentant de plus d'une rangée de processus frontières ce qui serait de toute façon visuellement inacceptable - Cf. Figure IX-4).

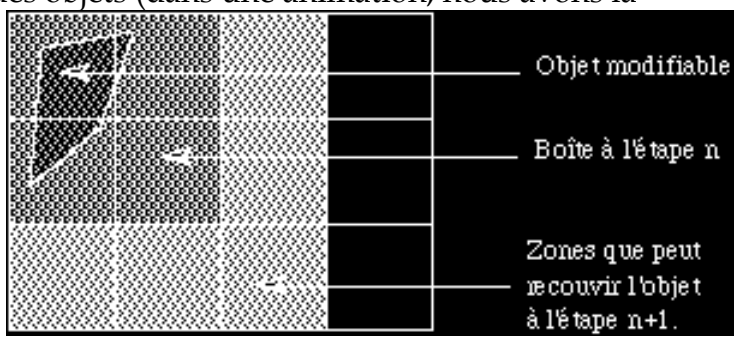

Figure IX-4 : modifications géométriques possibles

II.2) Résultats

	Sans optimisation C_n	Avec optimisation C_n
Nombre d'objets testés par rayon	1,65	1,64
Pourcentage d'objets testés par rayon par rapport à la base de données	33,00 %	32,86 %
Nombre de messages internes	219098	11374
Temps de calculs pour l'ensemble de l'animation.	70,9 s	70,0 s

A priori, il n'était pas évident que l'on puisse mettre en évidence une amélioration franche des performances car il y avait très peu d'objets dans cette scène, seulement 25 images calculées et les résultats sans optimisation déjà performants. Or on constate bien des gains :

- le nombre d'objets testés tous rayons confondus diminue légèrement (gain engendré ici par les règles C_1),

- ce gain plus les pré-calculs qui ne sont pas refait et la gestion simplifiée des données produisent une légère amélioration de temps pour arriver à une moyenne

d'une image toutes les 2,8 secondes (il faut noter tout de même qu'une seconde est la limite de précision de nos prises de temps),

- enfin, l'amélioration la plus spectaculaire se situe au niveau de la gestion des messages : diminution de 94,8 % du nombre de messages. On imagine facilement qu'avec une scène composée de beaucoup plus d'objets, les gains induits par la règle C_3 peuvent complètement résoudre le problème de la lourdeur de la gestion des messages.

II.3) Autre test

Le test précédent permet de constater des gains effectifs dans un cas où on avait effectivement une animation mais sur un nombre faible d'objets. Un second test avec plus d'objets a été effectué (100 boules opaques tombant sur un sol également opaque, une lumière projetant de l'ombre sur le sol). L'enchaînement des 25 images est beaucoup plus lent (on ne peut plus considérer avoir une animation). Cependant, les gains issus des règles C_n apparaît plus fortement.

	Sans optimisation C_n	Avec optimisation C_n
Nombre d'objets testés par rayon	1,06	1,06
Nombre de messages internes	325887	240775
Temps de gestion des messages	30,1 s	20,8 s
Temps d'initialisation	9,9 s	8,1 s
Temps de calcul total	1124,2 s	921,7

Cette fois les gains sur les temps de calculs sont très nets. Cependant, nos mesures ne sont pas assez détaillées et si l'on constate bien que ces gains touchent la gestion des messages ainsi que la préparation des données, d'autres parties de l'algorithme sont également optimisées (probablement la gestion des déductions logiques).

Conclusion

Bien que relativement simple, la troisième série d'optimisations introduite dans le cas de l'animation améliore encore les résultats obtenus en statique. En fait, ces règles optimisent les optimisations statiques en rendant presque négligeable les surcoûts qu'elles engendraient.

Nos calculs d'animation souffrent cependant d'un certain nombre de défauts :

- la lecture des données se fait dans un fichier ASCII, elle est donc lente,
- l'affichage de l'image peut également être améliorée,

- le parallélisme vrai n'a pas été testé en animation (il aurait fallut concevoir un protocole de communication plus performant pour faire du temps réel mais la partie lancer de rayon ne pose aucun problème nouveau).

En fait, le rendu réaliste en temps réel implique la maîtrise de tout le pipeline (modélisation, transfert, rendu, transfert, affichage). À ce jour, nous ne sommes performants que sur le rendu. Cependant, les résultats nous paraissent honorables au regard de la taille de l'image calculée (512*512), de la puissance de la machine utilisée (nous disposons de machines quatre fois plus puissantes) et même du type de scène car si la première scène test comportait peu d'objets, l'un d'eux était réfléchissant.

De plus, même si le sujet de cette thèse n'était pas à l'origine de produire des séquences animées d'images, les déductions introduites ici étendent et renforcent les optimisations mises au point pour le calcul statique d'images et il est sans doute possible de trouver d'autres règles plus fines (éventuellement en introduisant un nouveau mécanisme d'acquisition de la connaissance comme nous l'avions fait pour les règles B_n).