CHAPITRE V

Tester les performances en lancer de rayon

Mots clés

Lancer de rayon, tests de performance

Résumé

Ce chapitre a pour objet de donner des bases de réflexion au difficile problème de la comparaison de performance des logiciels en image de synthèse. Nous proposons un certain nombre de critères permettant de vérifier en partie la justesse des algorithmes de lancer de rayon, d'évaluer leurs performances sur des cas simples et de comparer certains aspects de deux algorithmes écrits et utilisés dans des conditions semblables aussi bien que totalement différentes. Bien qu'encore imparfaite, une partie de l'analyse présentée peut servir de base à une réflexion identique sur des sujets connexes comme la radiosité par exemple.

Summary

This chapter is a survey of the difficult problem of benchmarks and comparisons for image synthesis. We propose different criteria that allow evaluation of a ray-tracer and comparisons between two of them. They make possible the proof of correctness of optimizations too. This approach may be extended.

Introduction

Tester la justesse et les performances d'un algorithme, et au-delà, comparer deux algorithmes de lancer de rayon entre eux est un exercice reconnu comme très difficile. Peu d'articles sont consacrés directement au sujet [MCD93][BB94] et la majorité d'entre eux décrivent des aspects de comparaison assez généraux (ils

cherchent plus à énoncer les qualités générales d'un algorithme de lancer de rayon que de donner une méthodologie permettant d'effectuer des comparaisons qualitatives et quantitatives directes) [GP90]. Or une telle approche, et une telle réflexion sur la métrologie, est absolument nécessaire si l'on souhaite donner une dimension expérimentale à l'informatique, ce qui est généralement le cas de l'infographie.

I) État de l'art

I.1) Quelques tests existants

En informatique, il existe des critères de performances généraux concernant :

a) La puissance intrinsèque du processeur qui indique le nombre d'instructions élémentaires par cycle d'horloge que peut traiter un processeur : c'est par exemple le traditionnel coefficient MIPS (ou Million d'Instructions Par Seconde). Ce coefficient est capable de prendre en compte des caractéristiques architecturales comme le pipelinage des instructions, leur chaînage ou encore le parallélisme intrinsèque d'un processeur [He86].

b) La puissance de calcul en situation "réelle". On cherche ici essentiellement à tester le nombre d'opérations que l'on peut effectuer sur des flottants grâce à une batterie de programmes tests. Les résultats s'expriment la plupart du temps en terme de MFlops ou d'indice SPECfp92.

c) La bande passante réelle des systèmes de stockage qui est une composante essentielle lorsque l'on désire tester des programmes avec de très grandes bases de données ne tenant pas en mémoire vive. Des tests comme SAXER [Af91] cherchent à l'évaluer.

Ces tests observent essentiellement le matériel mais il est nécessaire d'évaluer l'influence des différentes couches de logiciel présentes sur la machine. On peut alors évaluer :

d) La puissance du système d'exploitation de la machine avec un programme comme BYTE ou MUSBUS [Af91]. Il est à noter que l'on fait généralement abstraction de la présence du système d'exploitation lorsque l'on procède à des tests alors qu'un même programme exécuté sur le même ordinateur mais avec des systèmes d'exploitation différents peut donner lieu à de grandes différences de performance.

e) La puissance relative des compilateurs est malheureusement rarement connue. En effet, l'analyse du code (C par exemple) peut conduire à supprimer des instructions inutiles, à réussir à en pipeliner d'autres, etc… or ces optimisations sont très inégales selon les machines, les langages et les sociétés qui les commercialisent.

De plus, les compilateurs ont une durée de vie effective très courte (quelques mois entre deux versions) ce qui rend très difficile la mise au point et surtout la réalisation les tests comparatifs entre ces compilateurs. Enfin, chaque compilateur dispose d'un jeu d'options assez grand et il n'est pas toujours évident de réaliser la bonne combinaison de ces options dans toutes les situations. La qualité du code généré doit donc être considérée comme une inconnue dans le cas général.

Enfin des tests plus spécifiques aux applications graphiques ont été élaborés :

f) La performance de l'environnement graphique est évaluée avec Xmark93 [Ro94] par exemple (tests de X11).

g) La qualité des processeurs graphiques (ou de leur émulation logicielle) est réalisée avec des tests comme PLBwire93 et PLBsurf93 [Ro94] qui donnent des résultats en nombre de vecteurs 3D/s ou de polygones 3D/s.

I.2) Des tests spécifiques au lancer de rayon

Nous avons vu au paragraphe précédent qu'il existait une assez large variété de programmes d'évaluation des performances allant du matériel aux plus hautes couches logicielles (Cf. Figure V-1). Cependant, ce chapitre a pour objet de mieux évaluer les performances du lancer de rayon en faisant abstraction (autant que possible) de son environnement. Or il n'existe pas à notre connaissance de tests spécifiques complets et normalisés sur le sujet. De plus, comme nous le verrons, les critères quantitatifs généralement utilisés ne semblent pas satisfaisants pour mettre en évidence les qualités et les défauts d'un programme de lancer de rayon.

Nous nous proposons de faire une analyse de ces problèmes et de donner un début de solution.

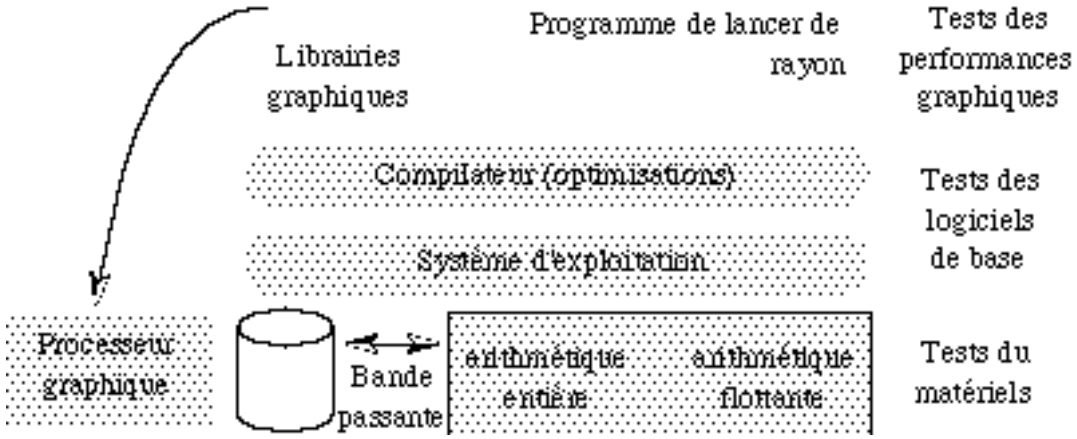


Figure V-1: Les différents niveaux d'application des tests de performance

II) Buts de tels tests en lancer de rayon

II.1) Vérifier la validité d'un programme de lancer de rayon

II.1.a) Points à vérifier

D'un point de vue pratique, il n'est pas évident de vérifier qu'un programme d'image de synthèse a donné le "bon" résultat. En effet, ce résultat étant avant tout une image, dans la majorité des cas on pensera que celle-ci est juste si :

- les objets ont la forme attendue (pas de trous, pas de déformations),
- les objets (non texturés) ont une couleur régulière (dégradés continus, couleurs homogènes),
- il n'y a pas apparition de motifs géométriques plus ou moins répétitifs sur des objets non texturés (mise en évidence d'effets d'aliassage),
- les ombres portées sont des tâches régulières dans l'alignement de l'objet faisant de l'ombre et de la source qui l'éclaire,
- les réflexions donnent une image symétrique de la scène,
- les transmissions semblent plausibles.

On remarquera que ces critères sont très intuitifs, qu'ils ne sont clairement vérifiables que dans les cas simples et qu'ils peuvent être perturbés par des phénomènes en marge de l'algorithme. En effet :

- l'image des objets, bien qu'ayant été produite par un calcul juste peut être entachée de problèmes d'aliassage dus aux arrondis de calcul ou à une discrétisation de l'écran insuffisante,
- la couleur des objets peut varier très rapidement si l'on utilise certains modèles d'illumination. Il sera par exemple difficile de dire si une tache due a la réflexion spéculaire de Phong est placée au bon endroit sur l'objet. De plus, les taches spéculaire ou de diffusion peuvent se chevaucher lorsque l'on a plusieurs sources de lumière.
- les ombres se recouvrent souvent, elles peuvent être ténues ou apparaître sur un groupe d'objets dispersés et les approximations sur la modélisation de ces ombres ne sont pas nécessairement les mêmes (ombre et pénombre, comportement en cas de transparence…),
- les réflexions font souvent apparaître une image partielle, ou l'image d'un morceau de scène qui n'est pas directement vu par l'œil ce qui rend la majorité des vérifications visuelles directes très aléatoires,
- les transmissions enfin sont les plus difficiles à vérifier car elles font appel à l'intuition ou à l'expérience de celui qui regarde l'image. Or ces images sont

généralement plausibles sans être assurément justes ce qui nuit à l'objectivité du jugement.

II.1.b) Méthodologie de vérification

II.1.b.1) Comparaison directe

La meilleure façon de vérifier qu'une image est juste serait de physiquement construire la scène, d'en faire une photographie et de numériser cette photographie (ou encore d'utiliser une caméra vidéo stockant une image numérique) puis d'effectuer une comparaison automatique pixel par pixel entre la réalité numérisée et l'image calculée. Cependant, cette façon de procéder est impossible pour plusieurs raisons :

- Même pour des scènes simples, la réalisation d'un modèle physique d'une scène est techniquement difficile (où trouver un ellipsoïde parfaitement transparent d'indice de réfraction 1,35 ?).

- Photographier et numériser la réalité induit nécessairement des modifications (sous-exposition ou saturation de la pellicule photographique ou des capteurs CCD, déformations dues à l'objectif de l'appareil, perte et déformation d'informations lors de la numérisation).

- La définition de la notion de couleur est technologiquement et physiologiquement difficile à appréhender : d'une part la vision des couleurs est propre à chaque individu (même si les variations sont généralement très faibles) et d'autre part le calibrage électronique des éléments d'affichage est sujet à de grandes variations.

- Vouloir calculer une scène qui existe réellement demande une connaissance très précise des différents matériaux utilisés, des conditions d'éclairage, des caractéristiques géométriques des différents éléments de la scène. Cela exige d'avoir un modèle de lumière extrêmement complet pour le calcul, or tous les modèles utilisés en image de synthèse ne sont que des simplifications des modèles physiques théoriques. Il faut de plus ajouter que les modèles physiques eux-mêmes ne sont pas parfaits et s'appliquent généralement à un ensemble de cas définis (i.e. l'optique géométrique ne traite pas les mêmes cas que l'optique ondulatoire).

- Enfin, même si tous ces problèmes pouvaient être résolus, il faudrait encore disposer d'une précision de calcul et d'un échantillonnage de l'écran qui seront la plupart du temps inaccessibles à la technologie disponible (par exemple il sera impossible d'utiliser un modèle ondulatoire de la lumière pour une scène complexe ou de décomposer la lumière en un grand nombre de longueur d'onde pour obtenir une décomposition spectrale issue d'un prisme [Mo81]).

On le voit, dans le meilleur des cas, une comparaison directe ne pourrait que fournir la conclusion que l'image calculée est plausible ou à défaut pas trop fausse.

II.1.b.2) Comparaison entre images

En fait, la solution la plus utilisée est la comparaison entre des scènes et des images tests (comme la célèbre théière ou encore les "sphere flakes" de Haines). Cette comparaison étant principalement visuelle, l'image sera considérée comme juste si elle ne diffère pas trop selon les critères énoncés en II.1.a). Cependant, cette approche pose elle aussi quelques problèmes :

- il faut être capable de lire les fichiers de description de ces scènes ou de reconstruire une scène équivalente (ce qui peut par exemple poser des problèmes entre des programmes utilisant des différences CSG par exemple et ceux utilisant simplement des listes d'objets),

- il faut disposer de modèles d'illumination semblables (certains modèles tentent d'obtenir des effets de flou, de pénombres, etc...),

- il faut également éliminer les problèmes d'aliassage,

- ces scènes sont très générales alors que l'on souhaiterait parfois ne tester que certains points particuliers (complexité selon le nombre d'objets, complexité en fonction du nombre de sources lumineuses, etc...).

On le voit, même si cette approche est plus réalisable que la comparaison directe, elle permet principalement de savoir si une image est plausible ou pas (sans oublier le fait qu'il faut être certain que l'image avec laquelle on effectue la comparaison est intrinsèquement juste).

II.1.c) Conclusion

Ne serait-ce qu'à cause des approximations dues aux modèles d'illumination, il n'est pas possible d'affirmer qu'une image calculée est juste par simple comparaison. De plus, l'analyse visuelle est trop suggestive et ignore trop de détails pour fournir un verdict fiable et définitif. Par conséquent, il est souhaitable de rechercher des critères numériques de comparaison, d'évaluer l'effet de l'aliassage et d'obtenir des références théoriques. La comparaison visuelle ne servant plus alors qu'à faire un pré-traitement éliminant les images manifestement fausses.

Nous proposons un début de solution concernant la justesse de l'image dans la suite, par le biais de la notion de rayon utile.

II.2) Évaluer les performances

Pendant longtemps, les recherches en synthèse d'images statiques se focalisèrent sur différents problèmes :

- chercher une méthode générale permettant d'avoir un rendu réaliste d'une scène,

- chercher des modèles d'illuminations calculables par ordinateur,

- améliorer les temps de calcul.

On peut considérer que les deux premiers points ont été résolus avec un certain succès pour des cas généraux. Aujourd'hui, et principalement en lancer de rayon, la question la plus difficile qu'il reste à résoudre est de diminuer le temps de calcul, et par voie de conséquence, d'évaluer les performances d'un algorithme.

II.2.a) Complexité théorique

L'étude de complexité d'un algorithme donne généralement le premier indice de performance. Cependant, cet indice ne fournit pas ici une aide précieuse. En effet, si l'on ne considère que le nombre d'objets, tous les algorithmes de lancer de rayon ont une complexité du même ordre (au pire O(nombre d'objets * nombre de sources) - Cf. Figure V-2), et si l'on cherche à avoir une formule plus détaillée sur un algorithme basique :

Si : P le nombre de pixels, N le nombre d'objets (ici sans boîte englobante), R le niveau de profondeur de la récursivité (imposé), S le nombre de sources lumineuses et C le nombre d'intersections calculées, alors :

$$C = P * \left[\begin{array}{l} N \text{ intersections testees pour les rayons primaires} \\ + S * N * \sum_{i=0}^{R-1} 2^{i-1} : \text{intersections testees pour les ombres} \\ N * \sum_{i=0}^{R-1} 2^{i} : \text{intersections testees pour les rayons secondaires} \end{array} \right] = PN(1+S)(2^R - 1)$$

Mais surtout, la complexité réelle est dans la majorité des cas bien inférieure à la complexité théorique et doit tenir compte du fait que la scène n'est pas une simple liste d'objets car ces objets se masquent les uns les autres et donnent lieu à des calculs de nature différente. Un calcul de complexité théorique doit donc tenir compte de ces aspects pour être effectivement exploitable.

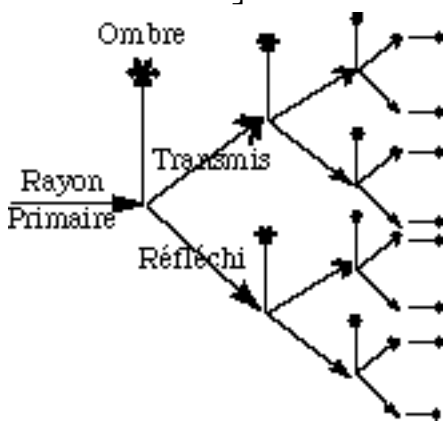


Figure V-2 : Arbre de calcul pour R=4 niveaux de récursivité

Dans le cas général, il sera bon de considérer les calculs d'intersection liés aux boîtes englobantes et ceux liés aux autres objets réels de façon séparée, car le coût des calculs est a priori très différents. Une analyse très fine devrait même être

capable de pondérer chaque type de calcul selon leur coût, mais arriver à une telle précision n'est pas, dans un premier temps, notre but.

II.2.b) Prise de temps de calculs

L'autre approche qui est incontestablement la plus utilisée, est de prendre le temps d'exécution du programme. Cependant, cette méthode présente de graves lacunes :

- Le protocole de prise de temps est rarement spécifié dans les articles ce qui ne permet pas de fournir des bases de comparaison fiables. Généralement, si les tests ont été effectués sous Unix, le temps donné est le temps utilisateur+système. Mais d'autres approches sont peut-être utilisées.

- Cette mesure a une précision limitée qui peut être source d'erreurs.

- Prendre un temps et le stocker est une opération qui n'a pas un coût nul. Si l'on souhaite ne faire qu'une prise globale, ce coût pourra être considéré comme négligeable, mais si l'on souhaite avoir un détail fin des différentes phases de calcul, ce n'est plus le cas. Par conséquent, le temps donné est toujours une surestimation du temps de calcul réel.

- Il n'est pas aisé de comparer des temps pris sur des systèmes très différents. Et que dire lorsqu'il s'agit de comparer les temps de programmes parallèles ?

- Il n'est pas toujours indiqué si les mesures fournies ne concernent que le temps de calculs de la scène, ou s'ils tiennent également compte du pré-traitement de la base de données par exemple (tri, découpage, calcul des boîtes englobantes...).

- Le même programme compilé sur des processeurs différents, avec des compilateurs différents ou des options de compilation différentes donnera bien évidemment un temps d'exécution différent. Or, il serait absurde de considérer que ce programme est plus ou moins performant que lui-même. En faisant ces manipulations, on teste plus le compilateur et la machine que le programme d'image de synthèse, ce qui n'est pas le but.

- Il est possible de réaliser plusieurs implantations d'un même algorithme théorique. Par conséquent, une prise de temps va également mesurer la qualité d'écriture du code (en terme de performance). Or on souhaite souvent tester une idée d'algorithme indépendamment des qualités du programmeur.

- Un programme développé spécifiquement pour une machine sera normalement plus rapide qu'un programme généraliste exécuté dans les mêmes conditions. Mais doit-on le qualifier de plus performant en sachant qu'avec un autre ordinateur les résultats seraient sans doute inversés ? Le champ d'application de tels

programmes doit être considéré comme restreint ainsi que leur durée de vie (qui est celle de la machine utilisée).

- Enfin, il est quasiment impossible, avec une prise de temps, de comparer objectivement deux programmes utilisés dans des conditions très différentes alors que le but essentiel des recherches actuelles est justement d'améliorer les performances des algorithmes et que ceux-ci ne sont quasiment jamais utilisés, donc comparés, dans les mêmes conditions.

Nous pouvons en conclure que la prise de temps :

- permet bien d'évaluer l'évolution d'un même logiciel pris dans les mêmes conditions de compilation et d'exécution sur le même ordinateur,
- doit être donnée pour un programme lisant au départ une base de données brutes d'objets aléatoires et doit inclure tous les pré-traitements (éventuellement en faisant apparaître les temps des différentes phases),
- permet de comparer deux logiciels généralistes pris dans les mêmes conditions sur la même machine.

Cependant, il serait trompeur de tirer une quelconque conclusion de la comparaison de deux logiciels différents, exécutés dans des conditions différentes sur des machines différentes, ce qui est hélas le cas général.

D'autres critères numériques doivent être fournis pour comparer deux programmes de lancer de rayon. Quelques uns d'entre eux seront présentés dans la suite de cet article.

II.2.c) Complexité mémoire

La complexité mémoire est un indicateur du comportement de l'algorithme en fonction de la scène (et plus généralement des données). Ce critère a un lien fort avec la complexité calculatoire : complexités mémoire et calculatoire donnent la complexité effective du programme. L'ordre de grandeur de la mémoire qu'il faut allouer doit tenir compte de plusieurs points :

- la mémoire nécessaire pour décrire un objet (n champs pour la couleur, k pour la géométrie...),
- celle nécessaire pour les optimisations (fonction du nombre de voxels, de n champs liés aux variables d'optimisation...),
- la possibilité ou pas de découper et de distribuer l'espace mémoire nécessaire en cas de parallélisation (ce point est essentiellement qualitatif).

Même si cela ne fournit qu'un ordre de grandeur alors que l'on pourrait avoir un compte exact, il semble préférable d'exprimer la complexité en nombre de

champs (en indiquant leur type) plutôt qu'en octets pour tenir compte de l'évolution des machines et du fait que celles-ci ne codent pas nécessairement les types sur le même nombre d'octets.

III) Recherches de paramètres

Heckbert [He89] propose de compter un certain nombre de paramètres pour analyser un programme de lancer de rayons :

- nombre moyen de rayons par pixel (analyse de la scène),
- nombre moyen de branchements dans l'arbre des rayons (analyse de la scène),
- nombre moyen d'objets testés par rayon (analyse des performances),
- pourcentage de rayons primaires, secondaires ou d'ombrage (analyse de la scène),
- pourcentage de tests manqués par volume englobant, réussis par volume englobant mais manqués par objet, réussis par volume englobant et par objet (analyse des performances),
- nombre moyen de voxels parcourus par rayon (analyse de la scène et des performances),
- temps CPU par rayon (analyse des performances).

Nous nous proposons de définir plus précisément des paramètres significatifs pour l'évaluation des performances et de la justesse d'un programme.

III.1) Définitions

Soient :

O	le nombre d'objets de la scène (1 lumière = 1 objet)	
S	le nombre de lumières	$S \leq O$
T	le nombre d'objets transparents (pas de sources transparentes)	$T \leq O$
P_0	le nombre de pixels de l'écran	

Le calcul d'une image en lancer de rayon peut être vue en trois phases distinctes :

- calculs liés aux rayons primaires (ceux qui partent de l'œil),
- calculs liés aux rayons secondaires (réfléchis et transmis),
- calculs liés aux rayons d'ombrage.

À moins d'une remarque explicite, on ne considérera dans la suite que des calculs d'image où il y a effectivement des pixels à calculer (on voit des objets à travers l'écran car sinon le calcul des pixels est trivial).

Dans la suite, une scène sera dite "fermée" lorsqu'il existe toujours au moins un objet intercepté pour tout rayon primaire ou secondaire.

III.2) Paramètres concernant les rayons primaires

III.2.a) Définitions

* Nombre théorique de rayons primaires = taille de l'image : $\mathbf{P_0}$

Pour pouvoir faire des comparaisons entre deux images, il est nécessaire d'avoir au départ un nombre de pixels identiques. Ce paramètre est facilement ajustable pour la majorité des programmes. Nos tests sont effectués sur un écran de 1024*1024 pixels.

* Nombre de rayons primaires lancés : P_l $\mathbf{0 < P_l \leq P_0}$

Pour le lancer de rayon de base (sans aucune optimisation), on lancera autant de rayons qu'il y a de pixels (dans ce cas $P_l = P_0$). Pour un lancer de rayon optimisé et si l'on considère une image où l'écran n'est pas totalement recouvert par les objets de la scène (non "fermée"), le nombre de rayons lancés peut être très inférieur au nombre de pixels (par exemple, on ne lancera des rayons que dans des zones écran recouvertes par une boîte englobante). Au minimum, on ne lancera aucun rayon (zone écran vide).

* Nombre de rayons primaires utiles (touchant au moins 1 objet) : P_u, $\mathbf{P_u \leq P_l}$

Parmi les rayons primaires lancés, certains toucheront effectivement au moins un objet alors que d'autres se perdront dans le vide. Ceux qui touchent un objet (et donc au moins l'objet qui sera effectivement conservé pour le calcul de couleur), seront appelés rayons utiles car ils ont absolument besoin d'être calculés pour fournir l'image finale (alors que les autres passent par un pixel qui est de la couleur du fond - fixée a priori -). Il y a en général moins de rayons utiles que de rayons lancés et il y en a au plus le même nombre (cas d'une scène fermée).

Ce critère est essentiel pour vérifier que deux images sont justes : quelle que soit la méthode utilisée, le nombre de rayons utiles doit être le même (aux effets d'aliassage près). Ces trois critères peuvent être illustrés par la Figure V-3 :

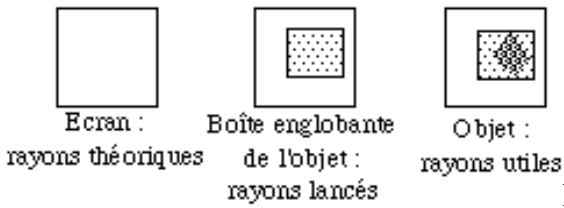


Figure V-3 : scène avec un seul objet

* Nombre d'intersections primaires calculées : I_p $\mathbf{I_p}$

Pendant l'algorithme, on va calculer un certain nombre d'intersections entre les rayons primaires et les objets de la scène (les boîtes étant éventuellement comptées à part). L'ensemble des calculs effectués pour tous les rayons primaires donne le nombre d'intersections primaires calculées : I_p.

Le nombre d'intersections calculées entre les rayons et les objets est un indicateur direct de la performance du programme. Plus I_p sera faible et plus le programme pourra être considéré comme performant. Ce paramètre est lié au nombre de rayons lancés. Dans le meilleur des cas, on aura le même nombre d'intersections calculées que de rayons utiles.

Si une telle intersection rayon/objet est indispensable au calcul final d'un pixel, elle sera dite intersection utile.

NB : le nombre de rayons utiles P_u donne également le nombre d'intersections utiles. Par conséquent, il est possible et intéressant de faire apparaître le pourcentage de calculs réellement utiles par rapport à ceux effectivement effectués :

$$\mathbf{U_p = 100 * P_u / I_p}$$

Dans le meilleur des cas, ce résultat est de 100% ce qui donne une performance maximale de cet algorithme, tous les calculs d'intersections effectués ayant été utiles.

* Nombre moyen d'objets testés par rayon primaire : $\mathbf{O_p = I_p / P_l}$

Dans un cas idéal pour une scène fermée (dans ce cas $U_p=100/O_p$), on ne testera qu'un objet par rayon primaire alors que pour le lancer de rayon de base, on est obligé de tester tous les objets. En pratique, on pourra considérer que, pour une scène quelconque ayant un grand nombre d'objets, lorsque l'on teste deux objets ou moins par rayon, l'algorithme a atteint un degré de performance qui est difficilement améliorable (intuitivement, cela veut dire que lorsque deux objets se recouvrent ou sont très proches, on a autant de chance de tester d'abord le premier ou le second car il n'y pas de critères généraux pour faire un meilleur choix).

Ce nombre d'objets testés par rayons primaires peut également être exprimé en pourcentage du nombre d'objets de la base de données. Cela a deux intérêts :

- il est ainsi possible de comparer des résultats obtenus sur des grosses scènes aussi bien que sur des petites scènes (il est plus parlant de dire que l'on teste 10% des objets dans une scène A composée de 100 objets, et 50% dans une scène B de 10 objets que de dire que l'on teste 10 objets dans cette même scène A et 5 dans la scène B ; dans ce cas, le fait que les deux scènes n'aient pas le même nombre d'objets aurait pu aboutir à la conclusion fausse que l'algorithme a un meilleur comportement sur la scène B),

- ce pourcentage exprime en partie le comportement de l'algorithme par rapport aux besoins mémoire. En effet, si un algorithme teste en moyenne 10% des objets d'une base de données, cela signifie que pour de très grosses scènes impossibles à charger totalement en mémoire vive, il sera possible d'utiliser une base de donnée répartie avec un système de cache mémoire sans trop pénaliser les performances de l'algorithme (si l'écart type n'est pas trop fort).

III.2.b) Algorithme

```
initialisation :
       Ip = Pu = Pl = 0 ;
Pour chaque rayon primaire lancé
       Pl ++ ;
       Pour chaque calcul d'intersection Ip ++ ;
       Fin Pour
       Si rayon primaire touche au moins 1 objet Alors Pu ++ ;
       Fsi
Fin Pour
```

III.3) Paramètres concernant les rayons réfléchis et transmis

III.3.a) Définitions

Les rayons réfléchis et transmis seront confondus sous le terme de rayons secondaires. En effet, une fois calculé le rayon réfléchi/transmis, ce rayon donnera lieu aux mêmes types de calculs dans les deux cas. Ce calcul ne privilégie aucun point particulier comme dans le cas des rayons primaires, et n'a recours à aucun test d'arrêt particulier comme dans le cas des ombres. Il semble donc logique de les regrouper.

* Nombre de rayons secondaires lancés : $\mathbf{S_l \leq}$

(pour chaque rayon primaire ou secondaire lancé, on a au plus un rayon réfléchi et un rayon transmis relancé, et on a R niveaux de récursivité).

* Nombre de rayons secondaires utiles : S_u $\mathbf{S_u \leq S_l}$

Comme pour les rayons primaires lancés, certains rayons secondaires toucheront effectivement au moins un objet donnant une information indispensable au calcul d'un pixel, alors que d'autres ne serviront pas. Il y a en général moins de rayons utiles que de rayons lancés ($S_u = S_l$ pour une scène fermée).

Ce critère est essentiel pour vérifier que deux images sont justes : quelle que soit la méthode utilisée, le nombre de rayons utiles doit être le même (aux effets d'aliassage près, effets qui seront cumulés à ceux des rayons primaires).

* Nombre total d'intersections secondaires calculées = I_s **I_s**

I_s est un indicateur direct de la performance du programme. Plus il sera faible et plus le programme pourra être considéré comme performant. Dans le meilleur des cas, on aura le même nombre d'intersections secondaires calculées que de rayons secondaires utiles.

* Pourcentage de calcul d'intersections secondaires utiles : **$U_s = 100 * S_u / I_s$**
* Nombre d'objets testés par rayon secondaire : **$O_s = I_s / S_l$**

Dans un cas idéal et pour une scène close, on ne testera qu'un objet par rayon alors que pour le lancer de rayon de base, on est obligé de tester tous les objets. Comme pour les rayons primaires, on exprimera cette donnée en pourcentage du nombre d'objets de la base de données.

III.3.b) Algorithme

```
initialisation : Sl = Su = Is = 0 ;
Pour chaque objet réfléchissant ou miroir touché par un rayon
      Sl ++ ;
      Pour chaque calcul d'intersection Is ++ ;
      Fin Pour
      Si le rayon touche au moins 1 objet Alors Su ++ ;
      Fsi
Fin Pour
```

III.4) Paramètres concernant les rayons d'ombrage

III.4.a) Définitions

Compter le nombre de rayons d'ombre est plus compliqué car les objets transparents multiplient les intersections et les cas particuliers. Le comptage des rayons se fera ainsi :

- en un point donné, on lance un rayon d'ombre pour chaque source,

- si ce rayon touche un objet transparent, on compte l'intersection puis on relance le calcul (avec le même rayon d'ombre). On cumule les calculs de pénombre jusqu'à épuisement des objets ou jusqu'à ce qu'une ombre complète soit générée).

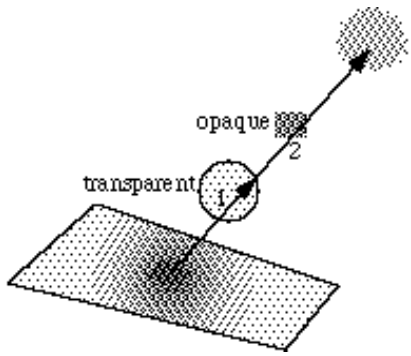


Figure V-4

- Si ce rayon touche (après des objets transparents) un objet opaque, les intersections précédentes dues aux objets transparents ne sont pas comptées comme utiles. Elles le sont sinon.

Cette méthode de comptage présente plusieurs lacunes (Cf. Figures V-5 et V-6):
- elle découle de l'algorithme de calcul d'ombre qui est une approximation grossière du trajet de la lumière dans le cas des ombres),

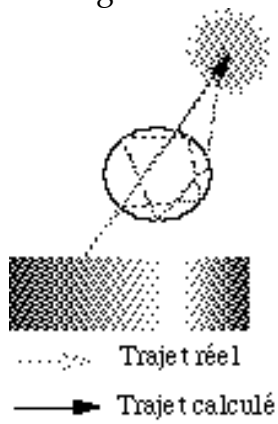


Figure V-5

- Dans le cas ou l'on a deux objets A, B, transparents capables de créer une ombre totale et un objet C opaque, suivant la configuration : A+B font de l'ombre (par "cumul d'opacité"), C fait de l'ombre, le nombre d'intersections utiles est de 1 (un calcul avec C suffit) mais si l'on intercepte A puis B par exemple, on va conclure que ce nombre est 2 (puisque l'on ne teste pas C). Donc le nombre de calculs d'intersections utiles pour les ombres est moins pertinent dans le cas général (Cf. Figure V-6).

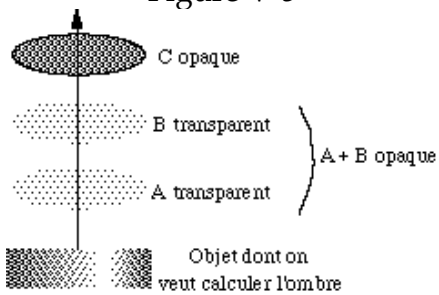


Figure V-6

* Nombre de recherches d'ombrage : $\mathbf{P_u+S_u}$

Chaque fois qu'un objet est touché, il y a une recherche d'ombrage (ceci permet de vérifier l'exactitude de l'algorithme sur ce point précis). Les calculs d'ombrages sont différents de toutes les autres intersections car, pour le lancer de rayon de base, on ne cherche pas un point d'intersection mais on veut seulement savoir si le rayon touche un objet. C'est pourquoi le nombre de recherches d'ombrage peut être différent du nombre de rayon d'ombre lancés (il peut arriver que l'on soit sûr d'être dans une ombre sans avoir à calculer d'intersection - règles A_4).

* Nombre de rayons d'ombre lancés : $\mathbf{N_l < P_u+S_u}$

On lance un rayon lorsque l'on cherche une intersection à calculer entre ce rayon et un objet. Cependant grâce aux règles A_4 (sur les cônes d'ombre), dans certain cas, on est capable d'établir qu'il y a intersection par simple comparaison de cosinus et donc sans avoir à rechercher d'intersection entre ce rayon et un objet.

* Nombre de rayons d'ombre utiles : $\mathbf{N_u}$

On définit un rayon d'ombre utile comme un rayon intersectant plusieurs objets transparents générant une pénombre ou une ombre, ou un seul objet opaque générant une ombre. Dans le cas où l'on peut assurer que le calcul du nombre de rayons d'ombre utiles est minimal (par exemple pour les scènes sans transparence), ce critère est essentiel pour vérifier que deux images sont justes : quelle que soit la méthode utilisée, le nombre de rayons d'ombre utiles doit être le même.

* Nombre d'intersections d'ombre calculées : $\mathbf{I_n}$
* Pourcentage de calculs d'ombre utiles : $\mathbf{U_n = 100*N_u/I_n}$
* Nombre d'objets testés par rayon d'ombre : $\mathbf{O_n = I_n / N_l}$

III.4.b) Algorithme

```
initialisation : Nl = Nu = In = 0 ;
Pour chaque rayon d'ombre
 transparent = 0
 Nl ++ ;
 Pour chaque intersection objet/rayon d'ombre lancé
       Si calcul d'intersection In++ ;
       Fsi
       Si le rayon d'ombre touche 1 objet Alors
             Si cet objet est transparent Alors
                   Nu ++ ; transparent++ ;
                   Si on atteint le seuil d'ombre Alors
                         Fin des calculs d'ombre pour cette source
                   Fsi
             Sinon Nu = Nu + 1 - transparent ;
       Fsi      Fsi
 Fin Pour
Fin Pour
```

III.5) Conclusion

Pour deux images calculées par des algorithmes différents, le nombre de rayons utiles (P_u , S_u, N_u) doit être le même et le nombre de recherches d'ombrage doit être $P_u + S_u$. Ceci fournit deux critères de vérification de la justesse d'un algorithme par comparaison ou déduction directe. De plus, il est envisageable de construire des scènes dont le nombre de rayons utiles est a priori connu, ce qui fournit alors un autre critère de validation sans avoir recours à des comparaisons (même si ce cas est réservé à des scènes très simples). Il est à noter que le nombre de rayons utiles est égal au nombre d'intersections utiles pour les rayons primaires et secondaires.

Les pourcentages de calculs utiles (U_p , U_s, U_n) pour des scènes fermées permettent de faire apparaître l'efficacité des optimisations et est par conséquent un indicateur essentiel de la performance de l'algorithme. Ces critères étant distincts, il est de plus possible de savoir sur quels types de rayons l'algorithme est le plus performant.

Le nombre d'objets testés par type de rayons (O_p, O_s, O_n) est une expression un peu différente de la performance de l'algorithme. Il permet de prendre en compte la taille de la base de donnée et par donc de faire une évaluation de la performance en fonction du nombre d'objets de la scène.

Ces deux derniers critères fournissent une évaluation qualitative d'un algorithme. Cependant, pour être complet, il faut leur adjoindre une évaluation quantitative, à savoir une prise de temps. En effet, les optimisations appliquées au programme ont pour conséquence d'améliorer U et O mais ces optimisations ont elles-mêmes un coût qui est de moins en moins négligeable au fur et à mesure que l'on se rapproche des optima théoriques. Ainsi, on cherchera à améliorer qualitativement l'algorithme tant que nous constaterons une amélioration quantitative (diminution des temps de calcul). De plus, ces critères ignorent les éventuels pré-traitements ; une prise de temps permet de le faire.

Enfin, les paramètres théoriques (**P_0, I_t, N_t**) serviront essentiellement de points de repère.

IV) Problème de l'aliassage

Le problème des arrondis de calculs se pose uniquement sur les bords de l'objet et est dû à deux causes principales :

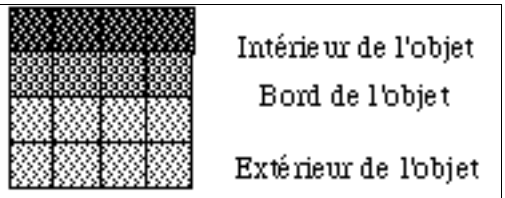


Figure V-7

1) Les arrondis sur la position du point de départ et les composantes du vecteur directeur.

2) Les arrondis de calculs lors de la résolution de l'intersection rayon/objet [Fr85].

IV.1) Point de départ et vecteur directeur

IV.1.a) Les rayons primaires

Dans ce cas, le point de départ est l'œil. Il ne devrait pas y avoir d'arrondi sur cette position puisque l'œil est un élément entièrement décrit dans les paramètres de la scène, à moins que l'on soit obligé d'appliquer une matrice à ce point de départ.

Le vecteur directeur est défini par les points œil/pixel par une simple soustraction. Cependant :

- il est généralement nécessaire de normer ce vecteur et la division en particulier, est source de différence d'arrondi,
- une matrice peut être appliquée à ce vecteur qui est une autre source d'erreur.

Le vecteur directeur des rayons primaires est donc la source principale d'erreur d'arrondi dans ce cas.

IV.1.b) Les rayons secondaires et d'ombrage

Le point de départ est un point résultant d'au moins un calcul d'intersection. Plusieurs opérations matricielles (directes et inverses) lui ont peut-être été appliquées.

Le vecteur directeur est soit le résultat d'un calcul sur le vecteur incident (réflexion/transmission) soit le résultat d'opérations entre le point de départ calculé et un point d'arrivée fixe (source lumineuse). La qualité des résultats sera donc a priori différente :

- rayon d'ombrage : un peu moins bonne que pour les rayons primaires,
- rayon réfléchi : moins bonne que pour les rayons d'ombrage,
- rayon transmis : légèrment moins bonne que pour les rayons réfléchis (le calcul vectoriel est un peu plus compliqué et on génère a priori deux rayons transmis pour un rayon réfléchi - on entre et on sort de l'objet -).

Enfin, les erreurs sur ces rayons vont se cumuler au fur et à mesure où l'on progresse dans la récursivité.

IV.1.c) Conclusion

Une scène avec beaucoup de réflexions et de transparences et d'opérations matricielles doit normalement générer beaucoup d'erreurs d'aliassage alors que les

scènes composées d'objets mats ne faisant intervenir aucune matrice doivent en produire très peu. Il faudra en tenir compte lors de la réalisation de scènes tests.

IV.2) Résolution des intersections

Les erreurs d'arrondi sont ici fonction de la qualité d'écriture de la procédure considérée et du type d'objet :

- les objets de type plan ou facette donnent lieu à peu de calculs donc peu d'erreurs,
- les quadriques impliquent la gestion de cas particuliers et sont plus compliquées,
- les autres primitives (quartiques...) sont encore plus délicates à gérer quant aux arrondis,
- les petits objets ou les objets très effilés sont des sources bien connues d'erreur d'aliassage.

IV.3) Évaluation quantitative de l'aliassage

Il ne nous paraît pas possible de fixer théoriquement la marge d'erreur due aux effets d'alias :

- Cette erreur est liée au type d'objet. La visualisation d'un cube peut être parfaite si celui-ci a un alignement idéal avec l'œil alors que pour une sphère il est certain que certains pixels du contour vont donner lieu à des arrondis de calculs (Cf. Figure V-8).

- Cette erreur est liée à la position de l'œil : si un cube placé dans un alignement parfait peut ne pas générer d'erreurs, il n'en est pas de même lorsqu'il est vu sous un autre angle.

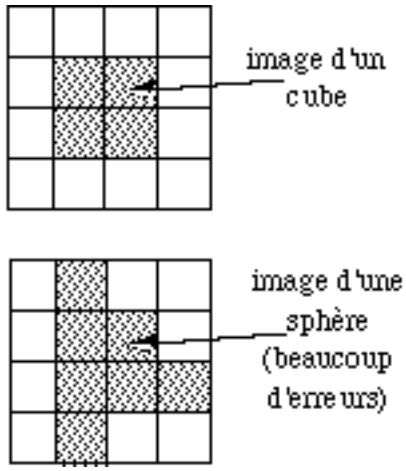
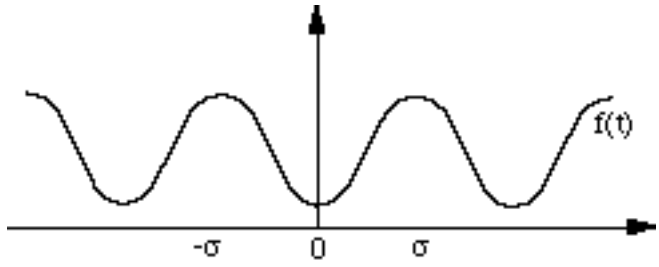


Figure V-8

- Cette erreur est liée à la position de l'objet ou encore à sa dimension relative par rapport à la taille des pixels. On retrouve là une conséquence du critère de Nyquist sur l'échantillonnage : une fonction f(t) peut être correctement

échantillonnée par un ensemble de ses valeurs f(nT), un point de l'échantillonnage étant à la distance T du précédent, en respectant le critère :



$$\frac{1}{T} = \frac{\sigma}{\pi}$$

Pour une fréquence d'échantillonnage trop grande, de l'information sera perdue.

Par conséquent, chaque scène est un cas particulier. Empiriquement, nous avons considéré qu'une différence du nombre d'intersections utiles de 1% entre deux images n'était pas significative.

Une tolérance plus grande pour des scènes très critiques (beaucoup de réflexions et surtout de transmissions sur des objets complexes dans une scène utilisant des matrices et nécessitant d'aller loin dans la récursivité des calculs) est, hélas, à envisager.

IV.4) Aliassage structurel

Enfin, nous avons constaté une dernière source d'erreurs : l'aliassage dû à la représentation des données. Imaginons deux facettes planes F_1 et F_2 s'intersectant en un segment [AB]. Considérons l'algorithme suivant :

```
Point_inter={+infini,+infini,+infini} ;
Pour tous les objets /* ici F1 et F2 */
Faire
       Calculer point d'intersection P
       Si P est plus près (strictement) du point
            de départ que Point_inter
       Alors Point_inter=P ;
       Fsi
Fait
```

Pour tous les points de [AB], cet algorithme ne donnera pas le même résultat du point de vue photométrique selon que F_1 est traité avant F_2 ou vice-versa. En particulier si l'une des facettes est réfléchissante mais pas l'autre (Cf. Figure V-9), le nombre de rayons secondaires et d'ombrage pourra varier (dans de faibles proportions cependant). Or comme les

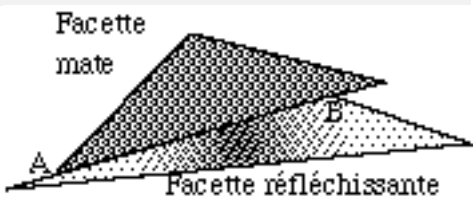


Figure V-9

données sont représentées de façon très différentes suivant les algorithmes (arbre, liste, données réparties,…), il n'est pas possible de supprimer cette source d'erreurs dans tous les cas.

V) Conclusion

Même si le protocole proposé dans ce chapitre ne peut pas prétendre avoir complètement résolu le problème de l'analyse des performances dans le cas du lancer de rayon, il propose une démarche de vérification et d'analyse des performances que nous jugeons indispensable et qui est ici directement exploitable par le biais de paramètres communs à la majorité des méthodes de lancer de rayon. Ces paramètres sont très importants pour comprendre le comportement d'une méthode mais également pour envisager de comparer différentes approches entre elles.

Le protocole présenté est un outils de mesure qui peut être appliqué à n'importe quel logiciel de lancer de rayon. Il n'est cependant pas parfait car il souffre du bruit engendré par différents effets d'aliassage qui, même s'ils sont faibles, semblent incompressibles.

Nous pensons de plus qu'il est possible d'appliquer une telle démarche à d'autres types de logiciels : Z-buffer, radiosité, …

Bibliographie

[Af91] Association Française des Utilisateurs d'Unix. Dossier Spécial Benchmarks, mars 1991, 169 p.

[BB94] Bache (R.), Bazzana (G.) - "Software Metrics for product assesment". Mac Graw-Hill, Maidenhead (U.K.), 1994, 249 pages.

[Fr85] Franklin (R.) - "Problems with raster graphics algorithms" in "Data Structures For Raster Graphics", EurographicSeminars, Springer-Verlag, 1985, p. 1-8.

[GP90] Green (A.), Paddon (D.J.) - "A highly flexible multiprocessor solution for ray tracing". The Visual Computer, 1990, p. 62-73.

[He86] Herscovici (A.) - "Introduction aux grands ordinateurs scientifiques". Paris : Eyrolles, 1986, 170 p.

[He89] Heckbert (P. S.) - "Writing a ray tracer", in "Introduction to ray tracing", ed. A.S. Glassner, Academic Press, 1989, p. 263-293.

[Ro94] Rost (R. A.) - "Reading the fine print : what benchmarks don't tell you". Computer Graphics Proceedings, ACM Siggraph, Orlando, July 1994, p. 497-498.

[MDC93] Maurel (H.), Duthen (Y.), Caubet (R.) - "A 4D Ray Tracing". Computer graphics forum, Actes d'Eurographics'93, Barcelone, septembre 1993, 12 (3), p. 285-294.

[Pi94] Piranda (B.) - "Optimisation de l'algorithme du lancer de rayon dans le traitement de scènes définies par des arbres CSG". Rapport de DEA, Besançon, 1994, 129 p.

[RA94] Ris (Ph.), Arquès (D.) - "Parallel ray tracing based upon a multilevel topological knowledge acquisition of the scene ", Computer Graphics Forum, Eurographics'94, Oslo, Septembre 1994, p. 221-232.