Protocole de test pour analyser les performances en lancer de rayon et modèle statistique simplifié

Didier ARQUÈS[1], Benoît PIRANDA[1], Philippe RIS[1,2]

(1) Institut Gaspard Monge
2 rue de la butte verte
93166 Noisy-Le-Grand Cédex
FRANCE

(2) Laboratoire d'Informatique
Fondamentale de Lille
U.F.R. IEEA, Bâtiment M3 Info
59655 Villeneuve d'Ascq
FRANCE

E-mail : arques@univ-mlv.fr, ris@lifl.fr

Résumé :

Cet article, après avoir brièvement fait le point sur quelques tests per performances des ordinateurs et de leur environnement logiciel, définit ur paramètres extraits d'un algorithme de lancer de rayon. Nous montrons paramètres peuvent servir à tester en partie la justesse d'un programm performances et à servir de base de comparaison avec d'autres lancers de modèle statistique basé sur ces paramètres permet de modéliser théor d'optimisations sur les rayons primaires dulancer de rayon que nus avons de

Mots clés : tests de performance, lancer de rayon, modèle statistique

1 État de l'art

Pour mesurer la puissance d'un processeur, il existe des critères de performances généraux qui indiquent le nombre d'instructions élémentaires par cycle d'horloge que peut traiter un processeur (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]. D'autres tests évaluent la puissance de calcul en situation "réelle" et ces résultats s'expriment la plupart du temps en terme de MFlops ou d'indice SPECfp92. La bande passante réelle des systèmes de stockage est estimable par des tests comme SAXER [Af91]. Ces premiers 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 la puissance du système d'exploitation de la machine avec un programme comme BYTE ou MUSBUS [Af91]. La puissance relative des compilateurs est malheureusement rarement connue. En effet, l'analyse du code 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 ce qui rend très difficile la mise au point et surtout la réalisation de tests comparatifs. 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 pour évaluer par exemple la performance de l'environnement graphique (Xmark93 par exemple pour X11). La qualité des processeurs graphiques 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.

Nous avons vu qu'il existait une large variété de programmes d'évaluation des performances allant du matériel aux plus hautes couches logicielles. Cet article a pour objet de mieux évaluer les performances du lancer de rayon en faisant abstraction de son environnement. Or il n'existe pas à notre connaissance de tests spécifiques et totalement satisfaisant sur le sujet. Nous nous proposons d'analyser ce problème et de donner un début de solution.

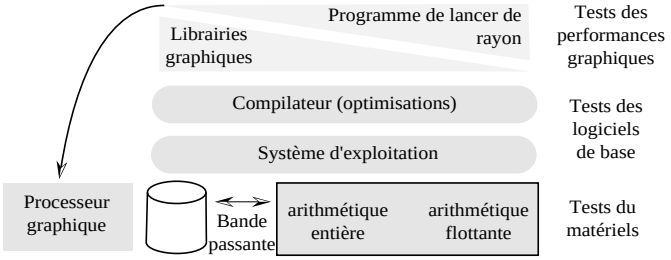

Figure 1: Les différents niveaux d'application des tests de performance

2 Buts des tests de performance en lancer de rayon

2.1 Vérifier la validité d'un programme de lancer de rayon

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 puis d'effectuer une comparaison automatique pixel par pixel. Ceci n'est pas possible à cause de la difficulté à trouver des matériaux dont on connaît parfaitement la photométrie, à cause du modèle d'illumination nécessairement approximatif et des modifications induites par la prise de vue et la numérisation notamment. 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. La solution la plus utilisée est la comparaison entre des scènes et des images tests (théière). Cette comparaison est visuelle ou recours à une analyse statistique sur l'image "différence" (pixel à pixel) entre les deux images. Mais cette approche pose elle aussi quelques problèmes puisqu'il faut être capable de lire les fichiers de description de ces scènes ou de reconstruire une scène équivalente, qu'il faut disposer de modèles d'illumination semblables et éliminer les problèmes d'aliassage. Enfin, ces scènes sont très générales alors que l'on souhaiterait parfois ne tester que certains points particuliers. On le voit, même si cette approche est plus réalisable que la comparaison directe, elle permet principalement de savoir si une image est plausible.
En Conclusion, en dehors de rares exceptions, il n'est pas possible d'affirmer qu'une image est juste par comparaison. L'analyse visuelle est suggestive et ignore trop de détails. Par conséquent, il est souhaitable de rechercher des critères numériques de comparaison, d'évaluer 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.

2.2 Évaluer les performances

2.2.a Complexité

L'étude de complexité C d'un algorithme donne généralement le premier indice de performance. Cependant, cet indice ne fournit pas ici une aide précieuse car les algorithmes de lancer de rayon ont une complexité du même ordre (au pire O(nombre d'objets * nombre de sources) - Cf. Figure 2 dans le cas où tout objet réfléchi et transmet la lumière). Si : P le nombre de pixels, N le nombre d'objets, R le niveau de profondeur de la récursivité (imposé), S le nombre de sources lumineuses. Alors /

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

Mais surtout, la complexité réelle est dans la majorité des cas bien inférieure à la théorie 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 exploitable.
La complexité mémoire est un indicateur du comportement de l'algorithme en fonction de la scène. Ce critère complète la complexité calculatoire : ces éléments donnent la complexité effective du programme. Elle doit distinguer la partie description de la scène, la mémoire nécessaire pour les optimisations et faire apparaître l'éventuelle possibilité de découper et de distribuer l'espace mémoire nécessaire en cas de parallélisation.

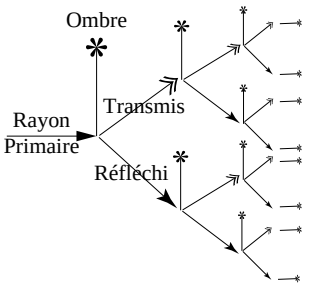

Figure 2 : Arbre de calcul

2.2.b Prise de temps de calculs

L'autre approche qui est plus utilisée, est la prise du temps d'exécution du programme. Cependant, elle présente de graves lacunes. En effet, le protocole de prise de temps est rarement spécifié ce qui ne permet pas de fournir des bases de comparaison fiables. Généralement, les tests sous Unix utilisent le temps utilisateur+système, mais d'autres approches sont peut-être utilisées. Cette mesure a une précision limitée qui peut être source d'erreurs et prendre un temps et le stocker est une opération qui n'a pas un coût nul donc le temps donné est toujours une surestimation de la réalité. De plus, il n'est pas aisé de comparer des temps pris sur des systèmes très différents. Le même programme compilé sur des processeurs différents, avec des compilateurs ou des options de compilation différentes donnera un temps d'exécution différent. Or, il serait absurde de considérer que ce programme est plus ou moins performant que lui-même. La prise de temps mesure également l'habileté 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 alors que le champ d'application de tels programmes est restreint. 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 que rarement utilisés dans les mêmes conditions.
Nous pouvons conclure que la prise de temps permet 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, doit inclure tous les pré-traitements et permet de comparer deux logiciels 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 et sur des machines différentes, ce qui est hélas le cas général. D'autres critères doivent être fournis pour comparer deux lancers de rayon.

3 Recherche de paramètres

Heckbert [He89] propose de compter un certain nombre de paramètres pour analyser un programme de lancer de rayons comme le nombre moyen de rayons par pixel, le nombre moyen de branchements dans l'arbre des rayons, le nombre moyen d'objets testés par rayon, le pourcentage de rayons primaires, secondaires ou d'ombrage, le pourcentage de tests manqués par volume englobant, réussis par volume englobant mais manqués par objet, réussis par volume englobant et par objet, le nombre moyen de voxels parcourus par rayon et le temps CPU par rayon. Cependant, cette analyse n'est pas assez précise. Nous nous proposons de définir plus précisément des paramètres significatifs pour l'évaluation des performances et de la justesse d'un programme en tenant compte des impératifs d'indépendance de matériel, de puissance, de style de programmation et dans une certaine mesure, de la variante de lancer de rayon employée

3.1 Définitions

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

On distinguera les calculs liés aux rayons primaires, les calculs liés aux rayons secondaires (réfléchis et transmis) et ceux 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. Dans la suite, une scène sera dite "fermée" lorsqu'il existe toujours au moins un objet intercepté pour tout rayon primaire ou secondaire.

3.2 Paramètres concernant les rayons primaires

* 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.

* 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.

Il y a en général moins de rayons utiles que de rayons lancés et il y en a au plus le même nombre (scène fermée). Ce critère est essentiel pour vérifier que deux images sont justes : quelle que soit la méthode utilisée, le nombre de rayons utiles est le même, à l'aliassage près (Cf. Figure 3).

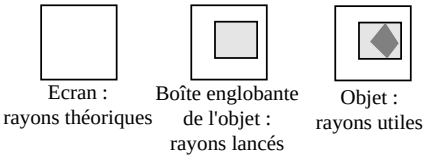

Figure 3 / scène avec un seul objet

* 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. 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. 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 et ce pourcentage exprime en partie le comportement de l'algorithme par rapport au besoin 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 intéressant d'utiliser une base de donnée répartie avec un système de cache mémoire (si l'écart type n'est pas trop fort).

3.2 Paramètres concernant les rayons réfléchis et transmis

Les rayons réfléchis et transmis seront confondus sous le terme de rayons secondaires, car une fois calculé le rayon réfléchi/transmis, il générera les mêmes types de calculs.

* Nombre de rayons secondaires lancés : $\mathbf{S_l \leq P_u * \sum_{i=1}^{R-1} 2^i}$

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

Comme pour les rayons primaires, certains rayons secondaires toucheront au moins un objet donnant une information indispensable au calcul d'un pixel. Il y a en général moins de rayons utiles que de rayons lancés ($S_u = S_l$ pour une scène fermée). Quelle que soit la méthode, le nombre de rayons utiles doit être le même (aux effets d'aliassage près).

* Nombre total d'intersections secondaires calculées = I_s $\mathbf{I_s}$

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 fermée, on ne testera qu'un objet par rayon.

3.3 Paramètres concernant les rayons d'ombrage

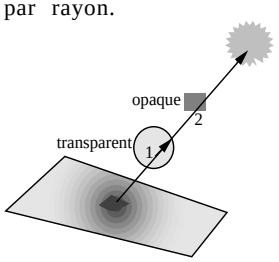

Figure 4

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 :

1) en un point donné, on lance un rayon d'ombre pour chaque source.

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

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

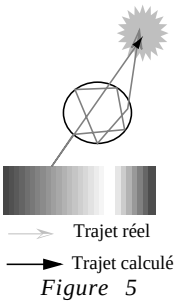

Figure 5

Cette méthode de comptage présente plusieurs lacunes (Cf. Figures 5 et 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), et 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, 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 6). C'est la raison pour laquelle nous ne proposons pas de variable associée à ce compteur dans la suite.

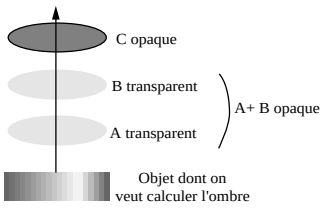

Figure 6

* Nombre de recherches d'ombrage : O $O = P_u + S_u$

Si un objet est touché, il y a recherche d'ombrage (on vérifie la justesse de l'algorithme avec ce paramètre). Les calculs d'ombrages sont différents des autres car 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 rayons d'ombre lancés (on peut être sûr d'être dans une ombre sans calculs d'intersection avec un simple calcul de produit scalaire avec la normale par exemple, sans lancer de rayon).

* Nombre de rayons d'ombre lancés : $N_l < O$

On lance un rayon lorsque l'on cherche une intersection à calculer entre ce rayon et un objet. Dans certain cas, on peut 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 : 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. Si on peut assurer que le calcul du nombre de rayons d'ombre utiles est minimal, quelle que soit la méthode utilisée, le nombre de rayons d'ombre utiles doit être le même.

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

3.4 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 N de recherches d'ombrage doit être (P_u + S_u).S. Ceci fournit deux critères de vérification de la justesse d'un algorithme par comparaison ou déduction directe : il est en effet 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.

4 Problème de l'aliassage

Un problème d'arrondis de calculs se pose sur les bords de l'objet et est dû à deux causes principales : 1) arrondis sur la position du point de départ et les composantes du vecteur directeur des rayons, 2) arrondis lors de la résolution de l'intersection rayon/objet [Fr85]. Une scène avec beaucoup de réflexions et de transparences et d'opérations matricielles doit normalement générer beaucoup d'erreurs d'aliassage. Il faudra en tenir compte lors de la réalisation de scènes tests. 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 est, hélas, à envisager. Enfin, nous avons constaté une dernière source d'erreurs : l'aliassage dû à la représentation des données. Imaginons deux facettes F_1 et F_2 s'intersectant en un segment [AB]. Tous les points de [AB] appartiennent à F_1 ou à F_2 et un 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, le nombre de rayons secondaires et d'ombrage pourra varier.

5 Comparaison avec d'autres algorithmes

Les graphiques suivants illustrent l'intérêt des tests de comparaisons. Deux algorithmes de lancer de rayon sont comparés :
1) un algorithme basé sur une méthode de Roth améliorée [Pi94] : boîtes englobantes au niveau des nœuds de l'arbre CSG, optimisation du parcours de cet arbre, rééquilibrage de l'arbre avant les calculs, discrétisation de l'écran permettant par projection de créer un sous arbre optimisé pour une portion d'écran donnée et voxellisation de l'espace.
2) Un algorithme parallèle basé sur une optimisation topologique[RA94].

Une série de scènes sert de support aux différents tests :

Scène I : calibrage de l'algorithme. La scène 1 est constituée d'un cube opaque dont la projection recouvre totalement l'écran (Cf. Figure 7).

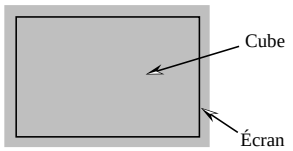

Figure 7

Scène II : scène composée de 100 cubes opaques de même taille. Les cubes forment un mur de "briques" dont la projection recouvre l'écran (Cf. Figure 8).

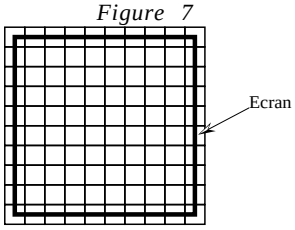

Figure 8

Scène III : test de l'illumination. 9 sources invisibles qui éclairent 91 cubes (tous visibles) répartis au hasard.

Scènes IV.1, IV.2 : tests des réflexions. L'écran est recouvert par la projection d'un cube A réfléchissant. À la sous-scène 1 (Cf. Figure 9a), un voit un cube B par reflet sur A. Pour la scène 2, on reprend la 1 avec une discrétisation de B en 99 cubes disjoints (Cf. Figure 9b). On construit ce second mur avec 100 cubes et en retirant un cube ou centre.

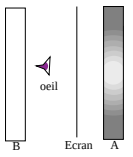

Figure 9a

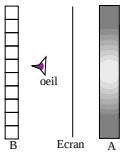

Figure 9b

Scène V : test des transmissions. L'écran est recouvert par un cube transparent (Indice de réfraction : 1,0). Derrière lui se trouvent 99 petits cubes (Cf. Figure 10)

Scènes VI : test de complexité. Devant l'écran, on place un pavage de cubes en faisant varier leur nombre : 1, 2, 4, 16, 64, 512.

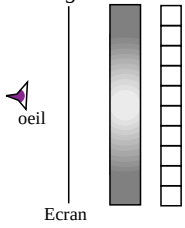

Figure 10

5.1 Vérification de la justesse des méthodes

Le nombre de rayons utiles entre les algorithmes1) et 2) est comparé. La différence étant sur ce jeux d'exemples toujours inférieure à 0,8%, nous en concluons que le protocole est valide et que les erreurs d'aliassage suffisent à expliquer cette différence. Les scènes où il y a beaucoup de réflexions ou de transmissions sont les scènes les plus sujettes aux erreurs d'aliassage.

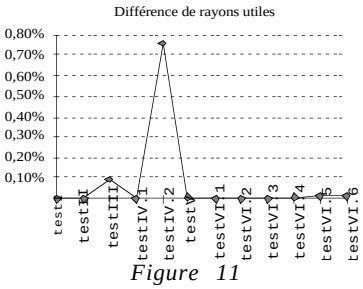

Figure 11

5.2 Objets testés par rayon primaire

Sur les deux courbes de la Figure 12 les performances des deux algorithmes sont comparées (nombre d'objets testés par rayon primaire).

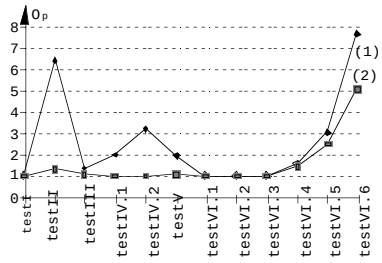

Figure 12

5.3 Pourcentage d'intersections utiles pour les rayons primaires

La Figure 13 montre le pourcentage d'intersections utiles pour les deux algorithmes. Ce pourcentage est un bon indice de performance si le coût d'élimination des calculs inutiles n'est pas trop important.

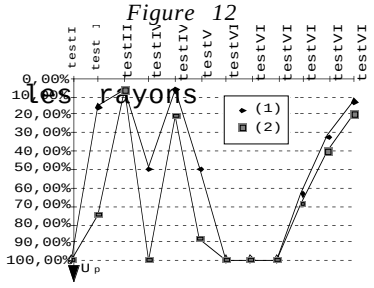

Figure 13

5.4 Nombre moyen d'objets testés tous rayons confondus

On retrouve la hiérarchie des tests précédents lorsque l'on considère tous les rayons. On notera qu'il est possible de mettre en évidence les cas favorables ou défavorables à un algorithme (par exemple l'algorithme 2 est mal optimisé dans le cas où des objets ne sont pas vus par les rayons primaires).

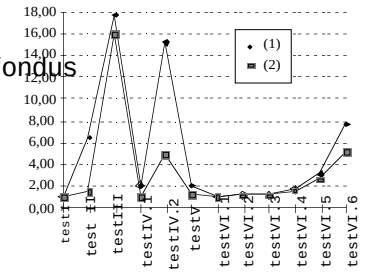

Figure 14

5.5 Temps de calculs

Les temps de calculs reflètent globalement les résultats exprimés par le nombre d'objets testés par rayon (Cf. Figure 15). Cependant, des divergences pourront apparaître si le coût des optimisations est très différent ou si l'on tombe dans un cas très favorable à un algorithme (test III).

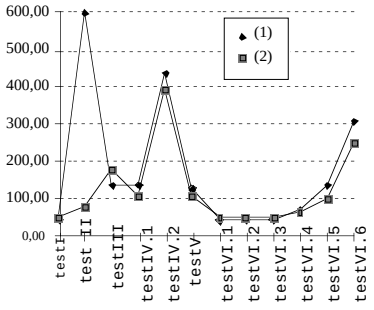

Figure 15

6 Analyse statistique

L'efficacité d'une méthode de lancer de rayon est très difficile à estimer a priori. Des approches statistiques ont été tentées. Goldschmidt & Al. [GS87] ont essayé d'évaluer les performances d'une hiérarchie de boîtes englobantes par ce biais : si l'on considère la probabilité conditionnelle qu'un rayon r tiré aléatoirement et de façon équiprobable touche un objet B englobé par un autre objet A déjà touché par r, et si A et B sont convexes, alors cette probabilité est égale au rapport des surfaces : $P(r \text{ touche } B \mid r \text{ touche } A) = \frac{S(B)}{S(A)}$. Une telle approche permet

certes d'estimer le coût probable d'une optimisation, mais pas a priori de connaître le comportement de l'algorithme.
À partir des définitions du paragraphe 3, nous allons construire un modèle statistique qui permet de prédire le comportement d'un algorithme de lancer de rayon vis à vis des rayons primaires optimisé selon [RA94] par des envois de messages ajustant dynamiquement des listes d'objets susceptibles d'être intersectés. Cette démarche nous semble importante car elle permet d'évaluer des performances sans avoir recours à l'implantation machine.

6.1 Hypothèses

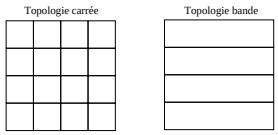

Figure 16

Les optimisations utilisées sont liées à une partition de l'écran en zones convexes (Cf. Figure 16) [AR93].
Hypothèse 1 : On considère une scène fermée.
Conséquence : $P_u = P_l, = P_0$ donc seul le calcul de I_p nous intéresse.
Hypothèse 2 : La partition de l'écran est réalisée selon une topologie bande.
Hypothèse 3 : Le temps de calcul des premières lignes de pixels de chaque bande est considéré comme négligeable par rapport au temps de calcul total des pixels de la bande.
Hypothèse 4 : le temps de transmission des messages à l'issue du calcul d'une première ligne d'une bande est considéré comme instantané.
Hypothèse 5 : les objets de la scène sont convexes.

6.2 Modèle théorique

On considère une partition de l'écran en n bandes de même hauteur.
Pour la création des messages, il suffit de connaître pour chaque objet, ses coordonnées max et min : un objet est un couple d'entiers $O=(x_1, y_1)$ avec $x_1 \leq y_1$ (représentant ses coordonnées min et max , Cf. Figure 17).

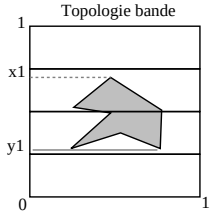

Figure 17

Définition : une scène est une famille d'objets. L'espace sous-jacent à la famille d'objets supposés en nombre aléatoirement grand est l'ensemble Ω qui peut être représenté selon la figure 18.

Se donner une scène, c'est à dire une famille d'objets, c'est choisir un échantillon $(O_1, ..., O_p)$ de p points de Ω. Cette scène sera caractérisée par la loi de probabilité P utilisée pour générer cet échantillon. On considèrera souvent P définie par une densité f sur Ω et on notera a la tribu des événements sur Ω . Le **modèle théorique d'une scène** est alors défini par le triplet :

S = (Ω , a, P)

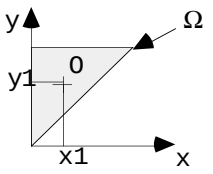

Figure 18

Définitions :
- Un objet O est un événement élémentaire de Ω.
- **a** est la tribu des événements liés au phénomène aléatoire étudié (cad générer statistiquement une scène).
- **P** est l'application qui permet de calculer la probabilité qu'un événement m se produise.
- f est la densité de la loi de probabilité P, c'est à dire que si χ est sur partie de Ω :

$$P(\chi) = \iint_{\chi} f(x,y)\,dx\,dy$$

6.3 Nouvelles définitions de la topologie

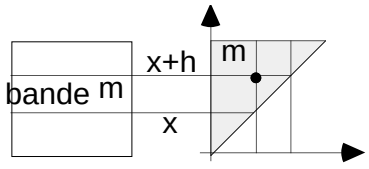

Figure 19

Définition : une bande située entre les lignes d'ordonnées x et y=x+h de la partition de l'écran est un point m=(x,x+h) de Ω (Cf. Figure 19). On notera de façon identique par m une bande et le point associé dans Ω.
Définition : un ensemble cohérent de bandes (Cf. Figure 20) est un ensemble $B = \{m_0, ..., m_p\}$ de p points de Ω tels que si $m_i=(x_i, x_{i+1})$ alors $m_{i+1}=(x_{i+1}, x_{i+2})$ et $x_0=0$, $x_{p+1}=1$.

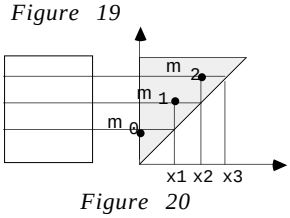

Figure 20

Hypothèse : on pourra choisir B tel que pour tout m_i de B, on ait $x_{i+1} - x_i = h$, avec $h=1/(p+1)$ constante donnée.

6.4 Définition de l'événement : "générer un message"

On considère un objet O=(x,y) de la scène S et une bande m_i telle que $m_i \in B$ ensemble cohérent de bandes $\{m_0, ..., m_p\}$. L'algorithme d'optimisation va générer deux types de messages pour O (Cf. Figure 21). Le message de suppression de cet objet envoyé vers le Nord, et le message de suppression de cet objet envoyé vers le Sud.

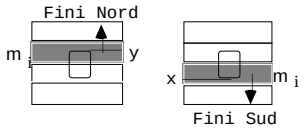

Figure 21

Définition : l'événement "générer objet O FINI vers le Nord à partir de m_i" se formalise par : ∃ **un objet O=(x,y) / y ∈ m_i =(x_i, x_{i+1}).**
L'événement "générer objet O FINI vers le Sud à partir de m_i" se formalise par :
∃ **un objet O=(x,y) / x ∈ m_i =(x_i,x_{i+1}).**
La probabilité h_i que la bande m_i émette un message vers le Nord est liée à la surface grisée de la figure 22. Pour une loi de probabilité P uniforme sur Ω on a :

$$P(\text{messageNord}) = P(h_i) = x_{i+1}^2 - x_i^2 \text{ avec } x_{i+1} = x_i + h$$

et pour une densité f(x,y) quelconque :

$$P(\text{messageNord}) = P(h_i) = \iint_{hi} f(x,y)\,dxdy$$

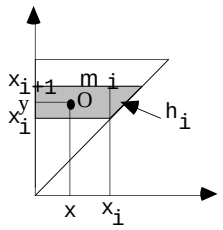

Figure 22

La Probabilité v_i que la bande m_i émette un message vers le Sud est liée à la surface grisée de la figure 23. Pour une densité f(x,y) quelconque :

$$P(\text{messageSud}) = P(v_i) = \iint_{v_i} f(x,y)\,dxdy$$

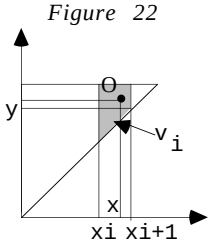

Figure 23

6.5 Nombre de messages reçus par une bande

Ce critère est très important car du nombre de messages reçus dépend directement l'optimisation du programme. En effet, sous l'hypothèse de non-redondance des messages de suppression des objets, chaque message reçu permet de tester un objet de moins par rayon à lancer.

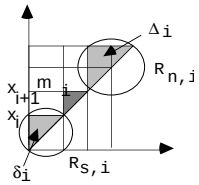

Figure 24

$$R_{n,i} = \sum_{j=0}^{i-1} E_{n,j} \text{ et } R_{s,i} = \sum_{j=i+1}^{p} E_{s,j}$$

où $E_{n,j}$ (resp. $E_{s,j}$) est le nombre de messsages envoyés au Nord (resp. Sud) générés par
la bande m_j et $R_{n,i}$ (resp. $R_{s,i}$) est le nombre de messages venant du Sud (resp. Nord) reçus par la bande m_i de B.

6.6 Nombre d'objets restant dans la liste des objets primaires

Si T_i est le nombre d'objets qu'il reste à tester pour la bande m_i on a sous les hypothèses 3 et 4 :

$$T = N - R_{n,i} - R_{s,i} = N\left(1 - \sum_{j=0}^{i-1} \iint_{h_i} f(x,y)\,dxdy - \sum_{j=i+1}^{p} \iint_{v_i} f(x,y)\,dxdy\right)$$

Si δi et Δi sont les aires définies sur la figure 24 on a finalement :

$$T_i = N\left(1 - \iint_{\delta_i \cup \Delta_i} f(x,y)\,dxdy\right)$$

Notation : on notera $Z_i = \delta_i + \Delta_i$ dans la suite

7 Comparaison statistique et réalité

7.1 Gain théorique

Notations : écran de résolution $r_x * r_y$ avec une partition en un ensemble cohérent B=(m_0, ..., m_p) de bandes.

Les critères statistiques donnés en 6) permettent d'évaluer I_p. Or I_p est un critère directement extractible du calcul d'une scène. Dans le programme, le calcul de I_p se décompose en deux phases :

a) calcul des premières lignes des bandes : $N*r_x*k$. Ceci représente le nombre d'intersections testés pour les rayons primaires lancés au travers des r_x pixels de la $1^{ère}$ ligne de chacune des k=p+1 bandes.

b) calcul des lignes suivantes (optimisées par les messages qui sont arrivés) :

$$\sum_{i=0}^{p}\left[(x_{i+1}-x_i)*r_y-1\right]*r_x*T_i$$

Ce qui nous donne statistiquement le paramètre I_p :

$$I_p = r_x\left\{N*k+\sum_{i=0}^{p}\left[(x_{i+1}-x_i)*r_y-1\right]*T_i\right\} = r_x*N\left\{r_y-\sum_{i=0}^{p}\left[(x_{i+1}-x_i)*r_y-1\right]*\iint_{Z_i} f(x,y)dxdy\right\}$$

Comme la scène est fermée (hyp. 1 : $P_l=r_x*r_y$) nous pouvons connaître le nombre d'objets testés par rayon primaire (O_p) :

$$O_p = \frac{I_p}{N*P_l} = 1-\frac{1}{r_y}\sum_{i=0}^{p}\left[(x_{i+1}-x_i)r_y-1\right]\iint_{Z_i} f(x,y)dxdy = 1-\sum_{i=0}^{p}\left[(x_{i+1}-x_i)-\frac{1}{r_y}\right]\iint_{Z_i} f(x,y)dxdy$$

Cas particulier 1: $\forall\, i : x_{i+1}-x_i = \frac{1}{p+1}, \eta_p = 1-O_p = \left(\frac{1}{p+1}-\frac{1}{r_y}\right)\sum_{i=0}^{p}\iint_{Z_i} f(x,y)dxdy$

Cas particulier 2: $\forall\, i : x_{i+1}-x_i = \frac{1}{k}$ et $f \equiv$ constante $= \frac{1}{S(\Omega)} = 2$

La distribution des objets est dans ce cas uniforme sur l'espace.

$$\eta_p = 2\left(\frac{1}{p+1}-\frac{1}{r_y}\right)\sum_{i=0}^{p}S(Z_i)$$

Calcul de la surface des Z_i (Cf. Figure 25) :

$$S(Z_i) = \frac{1}{2}\left\{\left(1-\frac{i+1}{p+1}\right)^2+\left(\frac{i}{p+1}\right)^2\right\} = \frac{1}{2(p+1)^2}\left\{(p-i)^2+i^2\right\}$$

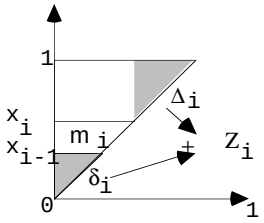

Figure 25

Nous en déduisons la somme des $S(Z_i)$ pour les k bandes :

$$\sum_{i=0}^{p}S(Z_i) = \frac{1}{2(p+1)^2}\left\{\sum_{i=0}^{p}(p-i)^2+\sum_{i=0}^{p}i^2\right\}$$

$$= \frac{1}{2(p+1)^2}\left\{\sum_{i=0}^{p}i^2+\sum_{i=0}^{p}i^2\right\} = \frac{p*(2p+1)}{6(p+1)}$$

Nous pouvons donc donner l'expression du nombre d'objets testés par rayon primaire O_p, ou plutôt le nombre d'objets que la méthode a réussi à éliminer de la liste des objets à tester :

$$\eta_p = \frac{1}{3}\left(\frac{1}{p+1}-\frac{1}{r_y}\right)\frac{p(2p+1)}{p+1}$$

7.2 Comparaisons numériques

Les données expérimentales sont les suivantes :

- 1000 objets O(x,y) définis en tirant x et y aléatoirement dans [0…1] grâce à la fonction drand48 du C ansi.

- r_x=1024 , r_y= 32 et p+1=32

- une topologie bande et un algorithme calculant les pixels de chaque bande ligne par ligne, de la ligne du bas à la ligne du haut.

- les calculs sur les bandes sont faits en parallélisme simulé sur une machine, en calculant une ligne de la bande i, puis une ligne de la bande i+1, etc… Les messages sont envoyés après le calcul de toutes les premières lignes et lus au début des lignes l, l>1 (pour se placer dans le cadre du parallélisme vrai alors que nous sommes en simulé).

La Figure 26 représente le pourcentage d'objets non testés (déduit de η) en fonction du nombre de bandes de façon théorique et réelle, ainsi que la différence des deux courbes. Les courbes théoriques et pratiques concordent très bien puisque l'écart maximal enregistré est de 0,59%. Les écarts enregistrés sont probablement dus au fait que l'échantillon de 1000 objets n'est pas parfaitement réparti puisque qu'il est issu d'une fonction C pseudo-aléatoire. Malgré cette restriction, les résultats obtenus valident l'approche statistique utilisée ainsi que les résultats théoriques concernant le gain espéré.

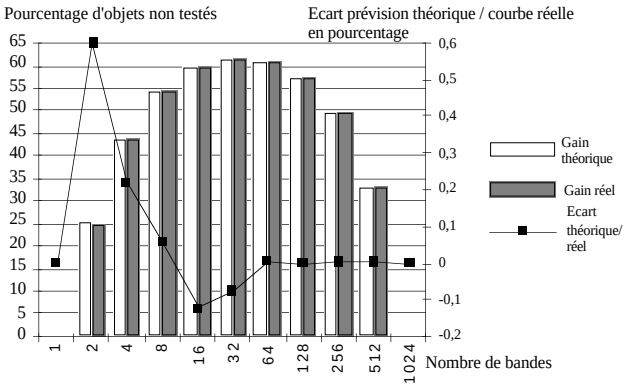

Figure 26

Conclusion

Cet article montre l'intérêt de l'analyse des performances d'un programme de lancer de rayon. Par la définition d'un ensemble de critères, il est possible de mettre en évidence les points faibles d'un algorithme, d'en vérifier en partie la justesse et de la comparer à un autre logiciel. De plus, en utilisant des outils statistiques, il est possible de créer un modèle totalement théorique du lancer de rayon et d'analyser ses performances en dehors de toutes considérations matérielles. Il reste cependant à étendre ce modèle à tous les types de rayons pour un ensemble plus vaste d'optimisations.

Bibliographie

[Af91] Association Française des Utilisateurs d'Unix. Dossier Spécial Benchmarks, mars 19

[AR93] : Arquès (D.), Ris (Ph.) - "Optimisation du lancer de rayon fondée sur une approch parallélisme simulé". Actes des journées AFIG-GROPLAN 1993, Bordeaux, p. 193-202.

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

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

[GS87] : Goldsmith (J.), Salmon(J.). - "Automatic creation of object hierarchies for Computer Graphics Application, 7(5), May 1987, p. 14-20.

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

[He86] Herscovici (A.) - "Introduction aux grands ordinateurs scientifiques". Paris : Eyr

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

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

[MDC93] Maurel (H.), Duthen (Y.), Caubet (R.) - "A 4D Ray Tracing". Computer graphics f 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 par des arbres CSG". Rapport de DEA, Besançon, 1993, 129 p.

[RA94] Ris (Ph.), Arquès (D.) - "Parallel ray tracing based upon a multilevel topolo acquisition of the scene ", Comp. Graph. Forum, Eurographics'94, Oslo, p. 221-232.