THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par **l'Institut Supérieur de l'Aéronautique et de l'Espace**
Spécialité : Informatique

Présentée et soutenue par **Gilles CADET**
le **26 septembre 2008**

Accélération de la requête d'intersection par la réorganisation des rayons

---

JURY

M. Mathias Paulin, président du jury
M. Laurent Chodorge
M. Christophe Coustet
M. Bernard Lécussan, directeur de thèse
M. Bernard Péroche, rapporteur
M. Claude Puech, rapporteur

École doctorale : **Mathématiques, informatique et télécommunications de Toulouse**

Unité de recherche : **Équipe d'accueil ISAE-ONERA MOIS**

Directeur de thèse : **M. Bernard Lécussan**

Résumé

La performance de la requête d'intersection, nécessaire à la simulation intensive de la propagation d'ondes, et indispensable aux rendus interactifs à base de lancer de rayons, est située au coeur du travail de cette thèse. En partant des meilleures structures de recherches et algorithmes de calculs d'intersections rayon/surface présents dans l'état de l'art, ce travail s'est concentré sur l'efficacité des groupes de rayons, sources de performance supplémentaire.

Cependant, à l'inverse des rayons simples, la cohérence nécessaire aux groupes de rayons pose la problématique de leur organisation spatiale pour approcher la performance optimale. Après avoir montré comment construire et traiter efficacement les groupes de rayons avec deux structures de recherche particulières, le cas des rayons à origine commune a été étudié, et optimisé, pour l'évaluation des sources ponctuelles et de l'occlusion ambiante.

Pour répondre à la problématique des rendus générant aussi bien des rayons cohérents que des rayons incohérents, une solution de couplage des structures de recherche est proposée en s'appuyant sur une stratégie globale d'affectation des rayons. Une autre solution, plus générale et destinée à exploiter la cohérence dans les cas les plus difficiles, est proposée pour regrouper automatiquement les rayons selon un critère a priori de cohérence.

En se plaçant dans le contexte des modèles ayant une très forte complexité géométrique, la mise en place d'une pile de rayons, ainsi que sa distribution sur une grappe de calculateurs sont proposées afin d'améliorer la localité d'accès aux données, et de fournir une solution de visualisation interactive.

Pour permettre la réorganisation automatique des rayons, et donc faciliter l'exploitation des groupes de rayons, cette thèse conclut qu'il est impératif de définir une nouvelle interface de programmation, en remplacement de la simple requête d'intersection.

Table des matières

Chapitre I

Introduction

1 Introduction générale

La requête d'intersection est une fonction qui permet, dans sa plus simple définition, de déterminer la surface la plus proche de l'origine d'un rayon. Par analogie au parcours rectiligne de la lumière, cette fonction trouve un intérêt majeur dans la simulation de sa propagation, pour laquelle l'utilisation intensive de rayons lumineux dans les techniques de rendu d'images réalistes requiert une grande performance.

La synthèse d'image est, depuis plusieurs décennies, divisée en deux classes de rendus : le rendu interactif contraint par le besoin de performance, et le rendu hors ligne privilégiant la qualité d'image. Le rendu interactif est généralement assuré par un algorithme de rastérisation très performant, "câblé" sur toutes les cartes graphiques standard. Cette technique est très largement utilisée dans la réalité virtuelle, les jeux vidéo et les outils de modélisation, car elle répond parfaitement à la contrainte de temps réel. Le rendu hors-ligne est essentiellement réalisé sur le processeur central en exploitant des techniques dérivées du lancer de rayons [25]. La qualité d'image atteinte par la simulation précise des effets d'ombrage, de réflexion, de réfraction et d'illumination globale, dépasse alors celle de la rastérisation, au prix d'une plus grande consommation de ressources de calculs.

Cependant, depuis les années 2000, la performance croissante de la requête d'intersection tend à supprimer la limite entre ces deux classes de rendus pour les raisons suivantes :

- Les différents travaux de recherche ont permis d'élaborer des structures de recherche [31], ainsi que des méthodes de calculs d'intersection rayon/triangle [76] très optimisées ;
- L'utilisation d'instructions vectorielles SIMD [82], présentes sur les processeurs modernes, a apporté un facteur d'accélération entre x2 et x3 ;
- Les architectures multi-coeurs se sont banalisées, et ont permis de profiter pleinement de la grande capacité de parallélisation des requêtes d'intersections.

Cependant, avant de pouvoir basculer d'une technologie de rastérisation à une technologie de "lancer de rayons", il reste nécessaire de démontrer des capacités de

rendu temps-réel tout en fournissant une meilleure qualité de rendu. Cette thèse s'inscrit alors dans un contexte où la performance de la requête d'intersection est primordiale.

En remarquant qu'un certain nombre d'opérations étaient réalisées de manière redondante par différentes requêtes d'intersection, des travaux [66, 77] ont mis en évidence la possibilité de factoriser des calculs en regroupant spatialement les rayons. Cet agencement permet alors d'accroître très significativement la performance de la requête d'intersection, par un facteur x3 à x10, et permettrait à la technique de lancer de rayons d'être compétitive avec la technique de rastérisation.

Cependant, ce facteur d'accélération n'est pas systématique, et dépend fortement de la cohérence des rayons. Cette thèse s'est alors concentrée sur la construction et l'utilisation efficaces des groupes de rayons, ainsi que sur différentes stratégies de réorganisation afin de former au mieux, et aisément, des groupes cohérents, gage de performance.

Dans ce contexte, les principales contributions de cette thèse sont :
- l'étude de la construction, et l'optimisation de la propagation de groupes de rayons ;
- la mémorisation de points d'entrée pour accélérer les calculs d'ombrage avec une source ponctuelle ;
- la détection d'espaces vides par l'utilisation d'une boîte de visibilité pour limiter l'évaluation de l'occlusion ambiante ;
- le couplage de structures de recherche pour traiter efficacement les rayons cohérents, et les rayons incohérents ;
- la définition d'un indicateur fiable et a priori de cohérence ;
- la définition d'une heuristique pour exploiter la cohérence partielle dans un groupe de rayons ;
- le regroupement automatique des rayons en exploitant l'indicateur de cohérence ;
- la mise en place d'une pile de rayons pour masquer les latences d'accès au support de masse lors du rendu de très grands modèles.

D'autres contributions, mineures ou en marge de la problématique principale de réorganisation des rayons, sont décrites dans le développement de la thèse.

2 Organisation de la thèse

Dans le chapitre II, après avoir rappelé le rôle de la requête d'intersection dans le contexte de la simulation de la propagation d'ondes, nous présentons l'interface d'un intersecteur géométrique. Les structures de recherche utilisées, ainsi que leur méthode de construction sont présentées avant de décrire les algorithmes de traversée, et de calculs d'intersection rayon/surface pour des rayons simples et des rayons SIMD.

Nous montrons ensuite dans le chapitre III comment regrouper efficacement ces rayons SIMD, et nous comparons plusieurs méthodes de propagation de ces groupes dans deux structures de recherche opposées : le kd-Tree et l'arbre BVH. Après avoir sélectionné les meilleures méthodes de propagation, nous présentons les performances absolues et relatives des groupes de rayons.

En nous limitant aux cas particuliers des rayons à origine commune, nous exploitons les groupes de rayons dans le chapitre IV afin d'optimiser les calculs d'ombrage depuis une source ponctuelle, et nous montrons une technique de détection des espaces vides pour limiter l'évaluation de l'occlusion ambiante.

En proposant deux solutions dans le chapitre V, nous répondons différemment au problème de divergence dans le contexte d'un rendu complet. La première solution exploite le couplage de deux structures de recherche pour dissocier et optimiser le traitement de rayons cohérents et incohérents. La deuxième solution propose un regroupement automatique des rayons afin d'améliorer dans des cas difficiles la cohérence, et de masquer la complexité de création de groupes.

Dans le chapitre VI, nous proposons l'utilisation d'une pile de rayons, et sa distribution sur une grappe de calculateurs, pour augmenter la localité d'accès à la géométrie, et permettre alors de traiter des très grands modèles ne pouvant tenir dans la mémoire centrale d'une seule machine.

En conclusion, nous résumons par chapitre les principales contributions de cette thèse, avant de présenter les perspectives.

Chapitre II

La requête d'intersection

Introduction

Dans le cas commun, la *requête d'intersection* est une fonction simple qui permet de trouver pour une demi-droite donnée la surface la plus proche parmi un ensemble de surfaces. Le terme de *lancer de rayons* désigne aussi bien cette fonction générale que l'algorithme de rendu en synthèse d'images qui simule la propagation de la lumière [85]. Pour lever toute ambiguïté, le terme de requête d'intersection est employé pour distinguer la recherche d'une intersection de l'algorithme de rendu.

Bien que simple dans sa définition, l'efficacité de cette fonction dépend de l'exploitation efficace et conjointe :

- des structures de recherche ;
- des méthodes de parcours des structures ;
- des méthodes d'intersections rayon/surface.

Dans la première section de ce chapitre, l'origine de cette requête d'intersection et son lien étroit avec la détection de visibilité pour la synthèse d'images sont présentés. Le grand potentiel d'application de cette requête aux différents domaines de la simulation de propagation d'ondes amène à proposer une librairie générique [33] pour répondre à l'ensemble des besoins. Dans la seconde section, les structures de recherche sont présentées en montrant particulièrement les méthodes de construction pour deux d'entre elles : le kd-Tree et l'arbre BVH (Bounding Volume Hierarchy). Dans la troisième et dernière section, les algorithmes de parcours de ces structures de recherche et de calculs d'intersections rayon/surface sont présentés aussi bien pour des rayons simples que pour des groupes de quatre rayons exploitant les instructions SIMD des processeurs modernes.

1 La requête d'intersection

1.1 Domaines d'application de la requête d'intersection

1.1.1 Origine

La première forme connue de recherche d'intersections le long d'un rayon est le *ray-casting*. Cette technique est l'ancêtre direct du lancer de rayons (ray-tracing) et son invention est attribuée à Appel [2]. Utilisée en synthèse d'image, cette fonction a pour but de déterminer la surface visible pour un pixel donné. Des rayons lancés depuis la caméra à travers chaque pixel de l'image permettent de trouver l'intersection avec la surface la plus proche. Un modèle d'éclairage [56] est ensuite appliqué pour tenir compte des effets du matériau et des lumières visibles depuis la surface. La principale différence avec le ray-tracing réside essentiellement dans le fait qu'aucun rayon secondaire n'est lancé.

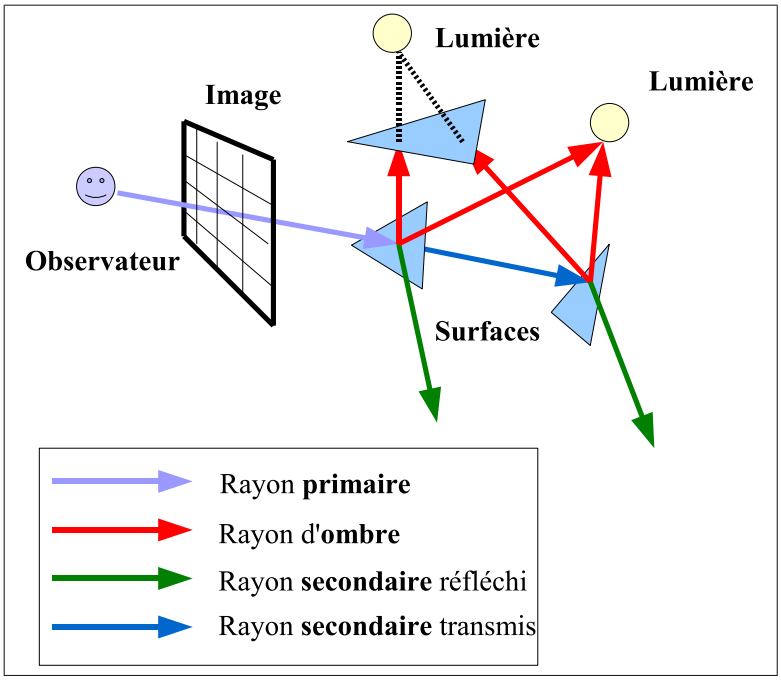


FIG. II.1 – Principe du Ray Tracing.

Bien qu'Appel soit considéré par la communauté scientifique comme le *créateur* du ray-tracing, des applications de la requête d'intersection ont été employées depuis 1966 par le groupe MAGI [37]. Pour les besoins du département de la défense américaine, des calculs de propagations d'ondes γ sont réalisés afin de simuler des expositions radioactives. Pour les besoins de l'étude, le groupe a alors développé une outil de visualisation de ces radiations qui par la suite, par le remplacement de la simulation des radiations par des rayons lumineux, deviendra le premier outil de production de films d'animations [60].

1.1.2 Autres techniques de résolution du problème de visibilité

Plusieurs techniques permettent de résoudre le problème de visibilité. Le scanline et le z-buffering [16] sont des techniques de visualisation d'une géométrie 3D sur une image (plan 2D). Plus performantes que la requête d'intersection pour ces types de calculs, elles sont toutefois limitées par la nature de leur algorithme.

La technique de scanline trie les surfaces projetées sur le plan de l'image par leurs ordonnées. Chaque ligne de l'image est alors balayée successivement pour trouver pour chaque pixel, la surface la plus proche. Les surfaces situées en dehors des lignes restantes à traiter sont éliminées successivement. Cette technique a l'avantage, comme la requête d'intersection de ne travailler que sur un seul pixel à la fois. Cette propriété est notamment intéressante pour limiter, dans le contexte de la synthèse d'image, les calculs d'illumination aux parties des surfaces visibles.

La technique de z-buffering est la technique la plus utilisée actuellement. La nature simple et force brute fait que cet algorithme peut être efficacement câblé dans les cartes graphiques. La technique bénéficie alors d'une puissance de calculs dédiée et complètement adaptée à des rendus temps réel comme le nécessite les jeux vidéos. Les sommets des surfaces sont d'abord projetés sur le plan de l'image puis traités en 2D. Trois composantes sont déduites de la projection d'un sommet : les coordonnées x et y dans le plan de l'image et la distance (ou profondeur) z par rapport au plan de l'image. La profondeur de chaque sommet est ensuite utilisée pour déduire par interpolation la profondeur de tous les pixels couverts par cette surface. Afin de déterminer les pixels visibles, seuls ceux ayant une profondeur inférieure à celle mémorisée dans un tampon de profondeur (z-buffer) sont retenus. Leur coloration est alors utilisée et leur profondeur mémorisée dans le z-buffer. Le z-buffering permet alors aisément de détecter les surfaces visibles.

1.1.3 Avantages et inconvénients

En utilisant la requête d'intersection, le problème de visibilité est résolu en lançant des *rayons primaires* utilisés dans la technique du lancer de rayons (voir la figure II.1) ou plus simplement par ray-casting. L'intérêt d'un intersecteur par rapport aux autres techniques de résolution du problème de visibilité est multiple. D'une part, la requête d'intersection n'est pas dépendante d'une projection 3D vers 2D ce qui permet une expression libre de tout besoin de calculs d'intervisibilité entre deux points de l'espace. Le ray-casting n'est qu'une forme simplifiée des possibilités de la requête d'intersection. D'autre part, les requêtes d'intersection sont indépendantes, d'un point de vue calculatoire, entre elles. La parallélisation de ces requêtes est alors triviale et bénéficie largement des capacités multitraitement des calculateurs modernes [38].

Cependant la requête d'intersection possède des points faibles. D'un point de vue fonctionnel, cette requête d'intersection est représentée par un rayon et n'a donc pas de volume. Si le besoin est de vérifier la visibilité dans un volume donné, celui-ci peut alors être échantillonné par plusieurs requêtes d'intersection. Du nombre de

ces requêtes d'intersection dépendra la précision de la réponse. Mais il est toujours possible d'échapper à un détail "epsilonesque" situé entre deux rayons.

De plus, d'un point de vue calculatoire, l'indépendance des calculs entre les rayons, qui permet une parallélisation efficace, amène cette technique à être moins performante que les techniques de scanline ou de z-buffering ; cette perte de performances est due à une redondance de calculs entre les rayons proches. Pour y remédier, il est alors envisageable de traiter les rayons par groupes (voir les sections 3.1.2 à la page 43 et III à la page 52) afin d'exploiter la cohérence de certains rayons et atteindre ainsi des performances proches des techniques concurrentes.

1.1.4 Domaines d'intérêts

1.1.4.1 Simulation générale de la propagation d'ondes La requête d'intersection est utilisable d'une manière plus générale pour simuler toutes formes de propagation d'ondes. Les lois de l'optique sont alors applicables pour peu que la longueur d'onde simulée soit d'un ordre de grandeur plus petite que la taille des surfaces modélisées. On peut classer les ondes en deux catégories, les ondes électromagnétiques et les ondes matérielles.

Les ondes électromagnétiques ne s'appuient pas sur la matière ce qui leur permet de se propager dans le vide. Elles sont la composante d'oscillations de champs électriques et de champs magnétiques. La vitesse de ces ondes dans le vide est celle de la lumière et leur longueur ne dépend alors que de leur fréquence. Par longueur d'onde décroissante, on peut citer :

- Les rayons γ dont la longueur d'onde est inférieure au picomètre ;
- La lumière visible dont la longeur d'onde est comprise entre $400nm$ pour la lumière violette et $700nm$ pour la lumière rouge ;
- Les ondes radios dont la longueur d'onde est supérieure au décimètre ;

La propagation d'une onde électromagnétique à travers un modèle composé de surfaces peut conduire à la recherche de points d'impacts, par l'utilisation de la requête d'intersection, lorsque les surfaces sont suffisamment grandes relativement à la longueur d'onde. Dans le cas concret du rendu de synthèse d'image, cette précision est amplement suffisante et en réalité très rarement prise en compte.

Les ondes matérielles utilisent la matière pour se déplacer. En créant un choc sur une molécule du milieu participant, celle-ci va propager l'onde de choc sur les molécules voisines avant de reprendre sa position initiale dans le cas d'un solide ou de continuer son mouvement dans le cas d'un gaz ou d'un liquide. La vitesse de ces ondes en raison de cette propagation au coup par coup est nettement inférieure à celle de la lumière. On peut citer :

- Les ondes acoustiques qui se propagent à la vitesse du son ($340m/s$). Les fréquences audibles pour l'homme sont situées entre $20Hz$ et $20KHz$. Les longueurs sont alors comprises entre une dizaine de mètres et le centimètre. On peut noter que la longueur d'onde d'une voix humaine est inférieure au mètre ($> 300Hz$).

– Les ondes sismiques qui se propagent à une vitesse maximale de $6km/s$. Bien qu'elles soient plus rapides que les ondes sonores, elles sont situées dans les très basses fréquences si l'on observe l'amplitude d'observation d'un sismographe située entre $0,01Hz$ et $10Hz$. Les ondes sont alors kilométriques.

La simulation des ondes matérielles par des calculs de requêtes d'intersection est nettement plus approximative et suppose que toutes les surfaces plus petites que la longueur d'onde soient ignorées. Les phénomènes physiques basses fréquences autre que le rayonnement doivent alors être simulés par des calculs plus adaptés. Pour les surfaces plus grandes que la longueur d'onde, les modèles de réflexion et de réfraction du rayonnement peuvent être parfaitement utilisés.

1.1.4.2 Importance de la requête de visibilité directe

Toutefois, sans tenir compte de la nature de l'onde, beaucoup de domaines exploitent tout simplement les trajets optiques. La requête de visibilité directe peut être assimilée à une onde qui n'aurait pas de dimension et qui offrirait une précision de calcul infinie. Cette requête est très utilisée dans les jeux vidéo [35] et outils de CAO pour détecter des lignes de visibilité et des collisions [68]. En synthèse d'image, bien qu'il soit possible de considérer les trajets optiques en tenant compte des particularités de l'onde lumineuse, la majorité des calculs exploite simplement la requête de visibilité directe comme le ray-tracing, les calculs d'occlusion ambiante, les calculs d'illumination globale ou plus généralement les calculs d'intervisilités.

Cette simplification est souvent utilisée sur un domaine restreint pour permettre la simulation d'un phénomène physique plus compliqué. Il est ainsi possible de simuler par des trajets optiques la réception de signaux satellites au sol [14] ou encore d'effectuer des études balistiques pour déterminer les points faibles d'un véhicule blindé [13].

1.1.4.3 Nécessité d'une librairie de résolution efficace de la requête d'intersection

L'étendue des domaines d'utilisation de la requête d'intersection invite à proposer une librairie générique pour ces calculs. Dans le souci de répondre à un besoin industriel de pluralité, de simplicité et de performances, la librairie RayBooster [33] a été développée. Cet intersecteur est situé au coeur du travail de cette thèse, son interface est présentée en suivant.

1.2 RayBooster : intersecteur multi-domaines

1.2.1 Organisation des données

Afin de satisfaire les besoins multiphysique évoqués précédemment, RayBooster définit une couche abstraite d'accès aux données. Les données nécessaires aux calculs sont les données géométriques représentées par des surfaces. Bien que RayBooster exploite de manière optimisée les surfaces triangulaires, il est possible de définir tous types de surfaces.

La figure II.2 montre l'organisation interne de la couche d'abstraction. La géométrie est référencée dans RayBooster par la création d'un *bloc*. Ce bloc indique le nombre et la nature des surfaces qui le composent. Les surfaces peuvent être des triangles, des instances ou des surfaces utilisateurs. Dans chacun des cas, une fonction de type callback associée au bloc permet de décrire plus précisément la géométrie. Au cours des calculs d'intersection, RayBooster interroge automatiquement ces callbacks. Ainsi pour la définition des triangles, il suffit de définir une callback qui renvoie les coordonnées des trois sommets. Pour les instances et les surfaces utilisateurs, le mécanisme mis en place est moins direct. Il convient d'abord d'expliquer la notion de *database*.

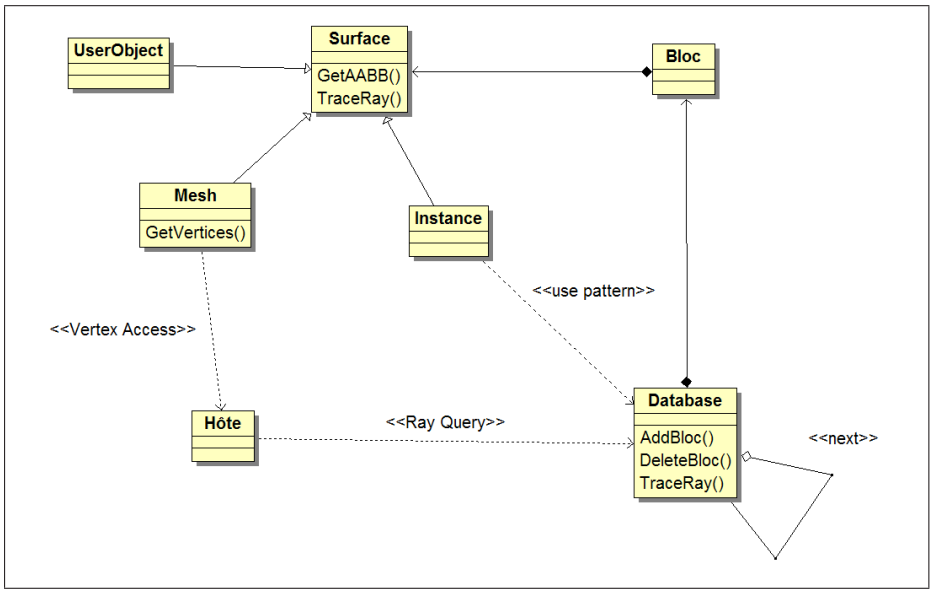


FIG. II.2 – Organisation des données.

L'ensemble des blocs créés est ajouté dans un objet database. Cette collection de blocs est un objet manipulable par l'application hôte et avec lequel il peut procéder à des calculs d'intersection. RayBooster supporte la création de plusieurs databases qui peuvent être regroupées sous une même entité pour participer simultanément aux calculs d'une requête d'intersection. Cette fonctionnalité peut être utilisée pour gérer des groupes d'objets de nature différente qui peuvent participer ou pas selon les besoins.

Une instance dans RayBooster est l'association d'une database et d'une transformation matricielle. Une instance est alors considérée comme une surface particulière et peut être ajoutée en tant que telle dans un bloc. La fonction de callback associée à la création de blocs d'instances devra alors fournir la matrice de chaque instance.

Afin d'ouvrir RayBooster à tous types de surfaces, l'hôte peut définir une surface utilisateur. Plutôt que de fournir la description de la géométrie, l'utilisateur fournit par l'intermédiaire de la callback la boîte englobante alignée aux axes associée à la surface. RayBooster utilise cette information en interne pour gérer au mieux la requête d'intersection avec cette surface. Une seconde callback est nécessaire pour

définir une surface utilisateur. Elle doit permettre de donner le point d'impact le plus proche entre un rayon et la surface définie. Il est ainsi aisé, à condition que l'on possède le code d'intersection entre un rayon et le type de surface que l'on souhaite traiter, d'intégrer une surface particulière dans un modèle géométrique classique.

1.2.2 Interface de programmation

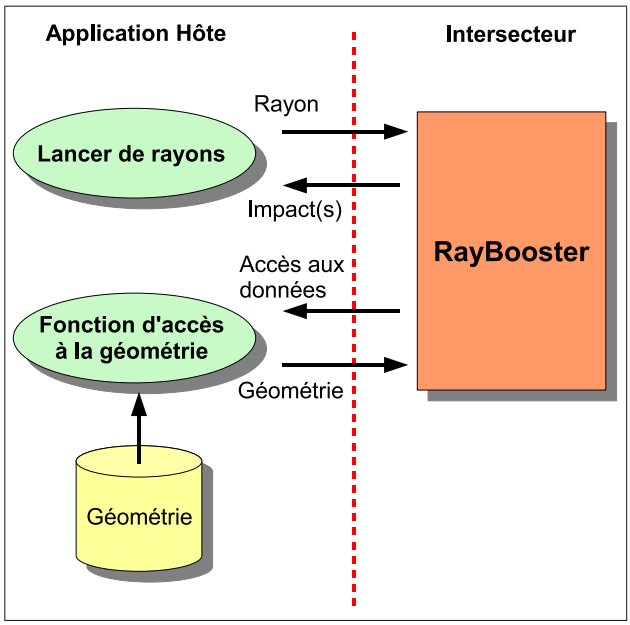


FIG. II.3 – Relations entre l'application hôte et l'interface de RayBooster.

La requête d'intersection est une fonction qui prend en entrée un rayon et qui retourne un ou plusieurs points d'impact. Les structures d'un rayon et d'un point d'impact sont présentées en détail dans la section 3 à la page 37. Dans sa forme minimale, un rayon est représenté par une origine et une direction. Cette demi-droite peut être segmentée en contraignant la distance de recherche par rapport à l'origine du rayon sur un intervalle de recherche restreint. La structure du point d'impact renseigne l'identifiant de la surface impactée ainsi que la position, la distance et les coordonnés barycentriques.

Dans un souci de généralisation, RayBooster permet de dissocier les trois types suivants de requête :

- la requête d'*occlusion* : recherche du premier impact ;
- la requête *mono-impact* : recherche de l'impact le plus proche de l'origine du rayon ;
- la requête *multi-impacts* : recherche des N premiers impacts les plus proches de l'origine du rayon ;

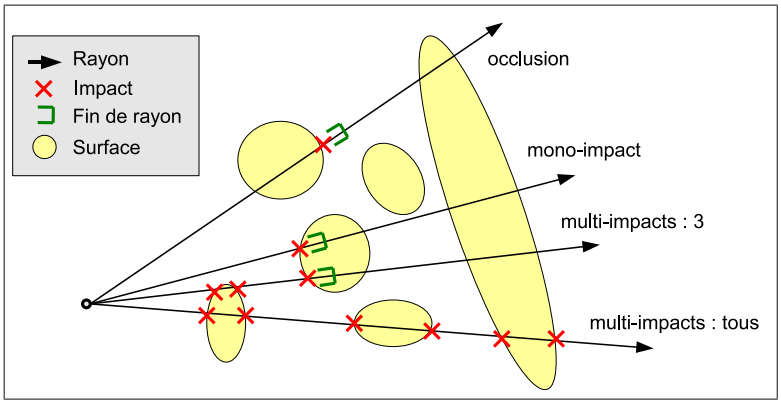


FIG. II.4 – Différents types d'occlusion.

La requête d'*occlusion* a l'avantage d'être plus rapide que les autres types de requêtes car son exécution est arrêtée dès la première intersection trouvée. Selon les structures de recherche utilisées par le moteur de requêtes d'intersection (voir la section 2.1 à la page 13), l'accélération des calculs est alors plus ou moins remarquable. On l'utilise fréquemment pour déterminer l'ombrage d'un point par rapport à une source lumineuse en lancer de rayons, ou pour détecter une éventuelle collision dans une direction précise ou plus généralement pour contrôler l'intervisibilité entre deux points de l'espace.

La requête *mono-impact* est la fonction principalement utilisée qui, pour un rayon donné, fournit le point d'impact le plus proche. Son utilisation en synthèse d'image par lancer de rayons est très fréquente et couvre la majorité des besoins. Cette fonction est également très intéressante pour une application hôte qui désire réaliser du picking : action de recherche de l'objet situé sous le curseur de la souris.

Enfin la requête *multi-impacts* est une optimisation qui évite de relancer un rayon de proche en proche dans le cas où celui-ci poursuit la même direction. En synthèse d'images, cette requête est intéressante pour simuler à moindre coût la transparence au travers de matériau n'ayant pas d'influence sur l'angle d'incidence. On peut également par exemple exploiter cette fonctionnalité pour un calcul d'irradiation par rayons γ pour évaluer toutes les surfaces traversées par ce type d'onde.

Quels que soient les domaines d'application, un matériau est affecté à chaque surface. De la propriété de ce matériau découle les lois physiques permettant de contrôler l'arrêt ou la continuité d'un rayon qui le traverse. Pour permettre une interaction pendant les calculs, l'application hôte peut fournir pour chaque requête une fonction de type callback qui permet de contrôler le déroulement d'une requête d'intersection. Cette callback, si définie, est appelée pour chaque point d'impact. Elle doit renvoyer un choix parmi quatre états pour contraindre la requête à :

- poursuivre les calculs en ignorant cet impact ;
- poursuivre les calculs en prenant en compte cet impact ;
- poursuivre les calculs en ne prenant en compte que des impacts éventuels plus proches ;
- arrêter les calculs ;

Le code source présenté dans la II.5 montre un exemple minimal d'utilisation de RayBooster pour créer une database, référencer un triangle et lancer un rayon de type mono-impact vers ce triangle. A condition de disposer de la bibliothèque RayBooster, cet exemple est directement compilable.

```
#include <stdio.h>
#include "rb.h" //librairie RayBooster

//callback appelée par RayBooster pour accéder aux sommets d'un triangle
static int vertex_access(int surface_id, //identifiant de surface
             double *v1, double *v2, double *v3) { //sommets
  v1[0]=2;  v1[1]=-1; v1[2]=-1;
  v2[0]=1;  v2[1]=-1; v2[2]=1;
  v3[0]=1;  v3[1]=2;  v3[2]=1;
  return 1;
}

//exemple minimaliste d'utilisation de RayBooster
int main() {
  RBrayimpact ray, impact;
  //creation d'une bdd vide
  void *db = rb_init_database();
  //ajout d'un bloc à la bdd contenant un unique triangle
  rb_add_triangle_bloc(1, vertex_access, NULL, db);
  //initialisation d'un rayon (axe X)
  rb_INITIALIZE_RAYIMPACT(&ray); ray.direction[0] = 1;
  //requête d'intersection de ce rayon dans la bdd
  if (rb_get_first_impact(&ray, &impact, db))
    //résultat : "Hit point (1.5, 0, 0) into bloc 0 on surface 0"
    printf("Hit point (%g, %g, %g) into bloc %d on surface %d\n",
      impact.position[0], impact.position[1], impact.position[2],
      impact.bloc_id, impact.surface_id);
  //destruction de la bdd
  rb_destroy_database(db);
  return 0;
}
```

FIG. II.5 – Exemple d'utilisation de RayBooster

1.3 Discussion

La requête d'intersection est une fonction qui permet de détecter un ensemble de points d'intersections entre une demi-droite et des surfaces. Cette fonction est exploitée dans de nombreux domaines d'application qui nécessitent de contrôler l'intervisibilité entre deux points ou de simuler tout phénomène de propagation d'ondes physiquement valide par la représentation d'un rayon. RayBooster [33] est alors développé comme un intersecteur généraliste, ce qui le distingue d'autres moteurs [61, 58, 36] dédiés aux lancer de rayons pour la synthèse d'image.

Tous les domaines d'application partagent un besoin commun de performance. La simple méthode force brute, qui vérifierait toutes les intersections entre le rayon et les surfaces pour ne garder que les points d'impacts pertinents, n'est pas envisageable. La performance de cette requête d'intersection induit alors la performance globale de la fonction qui la contient : l'organisation des données associées aux algorithmes de construction et de parcours fixent alors la performance de cette opération basique.

Certaines optimisations visent à réduire le nombre de calculs d'intersections en exploitant une structure de recherche qui permet d'éliminer très rapidement des groupes de surfaces inutiles par rapport à un rayon donné. Il est alors nécessaire de savoir construire ces structures de recherche de la meilleure façon possible en un minimum de temps (voir la section 2).

Il sera également nécessaire de mettre en place des algorithmes performants pour parcourir ces structures d'accélération et optimiser l'aspect calculatoire des intersections (voir la section 3 à la page 37).

Pour aller plus loin dans la recherche de performances, on peut considérer les requêtes d'intersection demandées au moteur dans leur globalité. Si le moteur a connaissance d'un ensemble de requêtes à exécuter en même temps, il peut alors chercher à exploiter une probable cohérence entre ces rayons (voir la section III à la page 52).

Toutes les expérimentations ayant permis d'étudier ou valider ces choix sont décrites dans la suite de ce document.

2 Construction des structures de recherche

La difficulté majeure d'une implémentation efficace de la requête d'intersection tient dans la nécessité de construire une structure de recherche au dessus de la géométrie. L'organisation de cette structure de données doit permettre de minimiser le nombre de calculs rayons/surface lors de son parcours. Les multiples formes de construction rencontrées dans le seul domaine de la synthèse d'image montrent l'étendue des possibilités et des difficultés posées par cette abstraction de la géométrie.

Après un bref rappel de l'état de l'art sur les structures de recherche, les méthodes de construction sont présentées pour deux d'entre elles : le kd-Tree et l'arbre BVH. Les idées majeures de ces travaux se retrouvent désormais dans l'état de l'art. Il n'en demeure pas moins que certaines furent initiées dans le même temps de réflexion que les travaux concurrents comme par exemple la méthode de tri par intervalles pour accélérer la construction (voir la section 2.3.2 à la page 27). Enfin, des résultats de construction de ces types d'arbres sont présentés et suivis d'une discussion.

2.1 Etat de l'art

2.1.1 Principe d'une structure de recherche

Le principe d'une structure de recherche est de limiter la complexité des calculs d'intersection rayons/surfaces lors de l'exécution la requête d'intersection. L'implémentation triviale d'une requête d'intersection pour trouver le point le plus proche serait de calculer l'intersection entre le rayon et chaque objet puis de retenir la surface la plus proche. La complexité d'un tel algorithme est alors en $O(N)$ pour N

surfaces. En regroupant les surfaces par entité (boîtes, sphères, objets...), il est possible de ramener la complexité des calculs à $O(log\ N)$. Bien que cette complexité soit une limite théorique ne correspondant à aucun modèle réaliste [30], et qu'il est simple de trouver des cas ne correspondant pas à cette complexité logarithmique, l'expérience montre que cette propriété est vérifiée pour la très grande majorité des modèles en synthèse d'images.

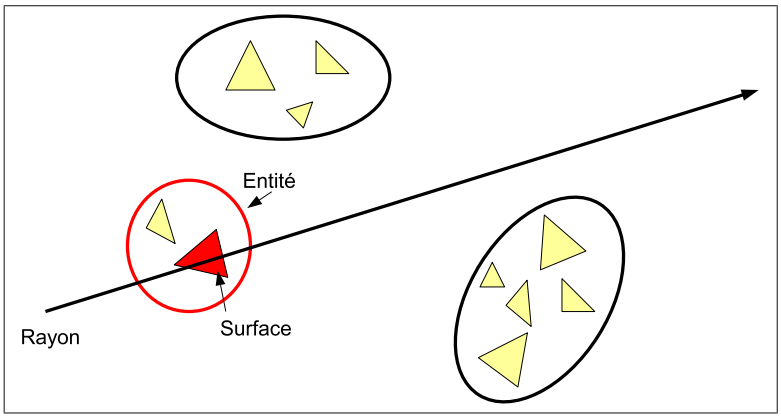


FIG. II.6 – Principe de regroupement des surfaces dans des entités simplifiées. Une dizaine de triangles sont regroupés dans 3 entités. La requête d'intersection réalise 5 calculs, dont 3 tests d'intersection rayon / entité, et 2 tests d'intersection rayon / triangle, pour au final déterminer une intersection valide avec un triangle. Cet exemple simple montre l'apport d'une organisation des surfaces dans le but de réduire le nombre d'opérations.

Pour permettre une diminution de la complexité, un calcul d'intersection est d'abord réalisé avec les entités regroupant les surfaces. En cas de succès, des calculs d'intersection sont réalisés avec les surfaces à proprement parler. Selon la forme des entités et leur organisation, on obtient différents types de structures de recherches. Deux approches distinctes séparent les structures de recherches en familles. On distingue d'un côté les entités qui séparent les objets ; dans ce cas de figure un objet (ou surface) n'appartient qu'à une seule entité. Et l'on distingue d'un autre côté les entités qui subdivisent l'espace ; un objet peut alors appartenir à plusieurs entités qui partagent une frontière commune. Dans chacune de ces familles, les entités peuvent être indépendantes ou hiérarchisées. Si une structure de recherche est constituée d'une hiérarchie d'entité, on pourra alors parler d'arbre (de recherche).

2.1.2 Séparation des surfaces : notion de groupes

Une façon simple de construire une structure de recherche est de regrouper les surfaces par l'organisation intrinsèque des objets. Il suffit alors de définir une entité de type boîte englobante (parallépipède) autour de chaque objet [67]. Cette construction pose cependant un problème d'équité entre objets qui ne comportent pas nécessairement le même nombre de surfaces. Cette disproportion peut alors générer des temps

de recherche d'intersections déséquilibrés. De plus le coût d'intersection d'un rayon avec une boîte n'est pas négligeable.

L'arbre BVH (Bounding Volume Hierarchy) [26] réunit une amélioration et une hiérarchisation du principe des boîtes englobantes tel que proposé dans [67]. D'une part les boîtes sont alignées aux axes ce qui permet de simplifier les calculs d'intersection rayon/boîte et d'autre part la hiérarchisation des boîtes permet de mieux équilibrer les temps de recherche et donc de se rapprocher d'une complexité de calcul logarithmique.

Une des plus récentes exploitations des volumes englobants provient des travaux de Wächter *et al.* [75]. Assez proches des travaux de Havran *et al.* [32] et publiés dans la même période, Wachter *et al.* proposent le BIH (Bounding Interval Hierarchy), une structure de recherche hiérarchique qui n'exploite pas des boîtes mais des plans séparateurs. L'avantage principal de la méthode réside dans une technique de construction très efficace et dans des coûts de traversée plus faibles. En effet, la coûteuse fonction d'intersection rayon/boîte est remplacée par une simple fonction d'intersection rayon/plan qui doit être exécutée au plus deux fois pour chaque étape de traversée.

2.1.3 Subdivision de l'espace : notion de plans séparateurs

Une autre façon de créer des entités de regroupement des surfaces est de découper l'espace. La plus simple façon est de construire une grille régulière [23] à partir du volume englobant de toutes les surfaces. La résolution de cette grille est choisie de façon à obtenir un niveau de maillage suffisamment fin pour limiter le nombre de surfaces par élément de la grille et pas trop dense pour limiter le nombre de cellules à traverser lors d'une recherche d'intersection. L'avantage de cette méthode est la simplicité de construction et la rapidité de parcours de proche en proche obtenue avec un algorithme de type 3D-DDA [23]. Le principal inconvénient provient du déséquilibre en nombre de surfaces dans les cellules. Le fameux problème du "teapot in stadium" [43] (théière dans le stade), qui évoque un modèle dont les densités spatiales et les dimensions sont très hétérogènes, génère dans le cas d'une grille, une occupation mémoire importante et inutile en raison de nombreuses cellules vides ainsi qu'une grande perte de performances en raison de certaines cellules référençant localement un grand nombre de surfaces.

De la même manière que les volumes englobants, la hiérarchisation de l'espace permet de résoudre les problèmes d'échelle. La méthode de subdivision distingue plusieurs types de structures de recherches :

- En divisant récursivement l'espace par deux à l'aide d'un plan séparateur, on génère un arbre de type BSP (Binary Space Partitionning) [22] ;
- En divisant récursivement l'espace par huit à l'aide de trois plans médians alignés chacun sur un axe, on génère un octree (arbre octal) [25] ;
- En divisant récursivement au moyen d'une grille régulière, on obtient un arbre multi-grilles [7] dans le cas ou chaque grille fille appartient exactement à une

maille de la grille mère, ou une hiérarchie de grilles [17] dans le cas ou les grilles filles peuvent chevaucher plusieurs cellules de la grille mère.

Un problème commun à toutes ces techniques est de bien choisir le plan séparateur. Le BSP au sens général rend possible le placement de n'importe quel plan séparateur. Un positionnement optimal est alors difficile à trouver. Le kd-Tree [72], qui est un cas particulier des arbres BSP, limite le positionnement de ce plan à un alignement sur un des trois axes. Le domaine de positionnement possible est alors réduit et l'intersection entre un rayon et un plan aligné aux axes peut être réalisée simplement par la combinaison d'une multiplication et d'une addition (voir la section 3.1 à la page 38). Pour l'octree, le positionnement des trois plans séparateurs est réalisé par définition au milieu de la boîte. Pour les grilles uniformes, l'espacement entre les plans étant fixe, il suffit de déterminer la résolution de la grille. Il existe cependant des variantes pour l'octree et la grille qui autorisent un positionnement libre des plans séparateurs [84, 24].

2.1.4 Quelle est la meilleure structure de recherche ?

Toutes ces structures de recherche partagent un but commun : accélérer la recherche d'intersections. Bien que la problématique semble se limiter à choisir la structure de recherche qui parvienne le mieux à cette mission, le choix n'en reste pas moins délicat ; la façon dont elles se construisent est aussi un choix déterminant. La construction est source de temps de prétraitement et d'occupation mémoire. De plus, selon l'objectif choisi de rapidité de construction, de faible occupation mémoire, de support de scènes dynamiques ou de vitesse d'exécution de la requête d'intersections, de nombreuses combinaisons peuvent s'opérer.

Les structures hiérarchiques s'imposent d'une manière générale par une meilleure adaptation à une large gamme de modèles. En terme de performances, le kd-Tree est l'arbre de prédilection pour le rendu temps réel. Son excellente efficacité dans la recherche d'intersections a été montrée dans de nombreux travaux [30, 82, 66, 6]. L'arbre BVH, bien que négligé et mal noté dans les tests exhaustifs élaborés par Havran [30], a fait cependant l'objet d'un intérêt nouveau suite aux travaux de Wald *et al.* [77]. En s'inspirant des multiples améliorations apportées au kd-Tree, Wald a montré que cet arbre était très intéressant quand on désirait supporter des scènes dynamiques et/ou que l'on pouvait exploiter efficacement la cohérence des rayons.

Les arbres kd-Tree et BVH sont deux structures hiérarchiques de familles différentes. Pour leur aspect complémentaire, des études ont été menées principalement sur celles-ci. Les sections III à la page 52 et 1 à la page 118 décrivent en détails la nature de leur complémentarité. Dans la suite de ce chapitre, les méthodes de construction sont présentées simultanément pour ces deux arbres, et lorsque des techniques similaires sont appliquées alors seules les différences sont mises en valeur.

Nos choix d'implémentation ont été guidés par la nécessité de définir une méthode de construction rapide, économe en mémoire et qui s'adapte à un grand nombre

de modèles. Des choix communs ont donc été appliqués à ces deux types d'arbre concernant :

- *Les paramètres de construction* Aucun paramétrage de l'utilisateur ne doit être requis. L'utilisation d'un modèle de coût nommé SAH (Surface Area Heuristic) [47], tel que décrit dans la section 2.2, s'avère alors indispensable.
- *Le prétraitement* Dans un souci de réactivité et d'économie mémoire, la durée de prétraitement est limitée par une construction paresseuse des arbres (voir la section 2.4 à la page 33).
- *Le sens de construction* En raison de la nature paresseuse de la construction, une construction de type descendante est privilégiée.

2.2 Le modèle de coût

La construction de toute structure de recherche est guidée par une heuristique qui définit des paramètres de construction. Ceux-ci peuvent être fixés a priori avant la construction ou de manière adaptative pendant la construction. Pour une grille uniforme, on cherchera par exemple à trouver les meilleures dimensions de discrétisation qui permettront de minimiser le temps de parcours de cette structure ainsi que le temps de calcul des intersections. Pour un arbre (voir la figure II.7), c'est à dire une structure de recherche hiérarchique, les différents paramètres doivent régir les élément suivants :

- la méthode de subdivision : comment répartir au mieux les surfaces lors de la subdivision d'un noeud ?
- le critère d'arrêt de subdivision : quand arrêter la subdivision de l'arbre ?

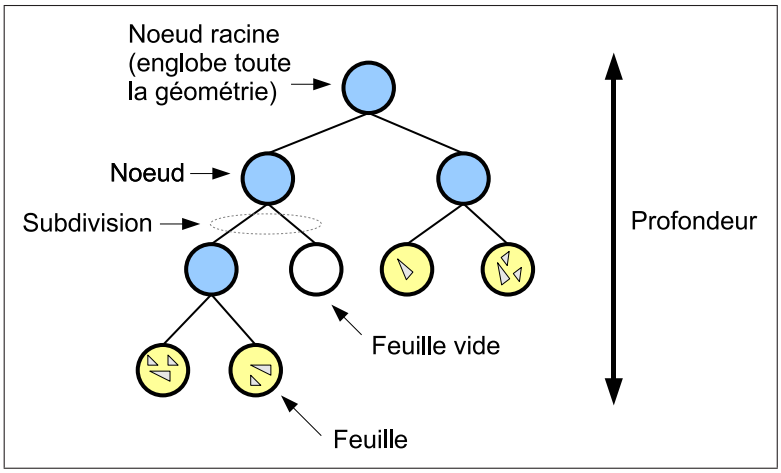


FIG. II.7 – Structure d'un arbre de recherche.

Dans le cas du kd-Tree et de l'arbre BVH, le nombre de subdivisions pour chaque noeud est fixé à deux. Cela permet d'une part de simplifier la méthode de subdivision, et d'autre part d'optimiser le parcours de cet arbre en limitant le nombre de tests. Les paramètres régissant la construction d'un arbre ne sont pas déterminés a priori mais s'appuient sur un modèle de coût présenté en suivant.

2.2.1 Le modèle

2.2.1.1 La méthode de subdivision La racine d'un arbre est représentée par la boîte englobante alignée aux axes de l'ensemble des surfaces. Par usage, le terme de "boîte" est toujours utilisé dans le reste du document pour désigner un parallépipède aligné aux axes, bien qu'une désignation plus reconnue est $AABB$ pour Axis Aligned Bounding Box. Pour construire un arbre, l'ensemble des surfaces incluses dans une boîte est réparti en deux sous-ensembles et ce de manière récursive. Mac Donald *et al.* [47] ont montré que le meilleur moyen d'évaluer la qualité, et donc de guider cette répartition, est de considérer le nombre de surfaces contenu dans chaque sous-boîte en rapport avec leurs surfaces. Cette considération tient du fait que la probabilité de passage d'un rayon dans une boîte fille est égale au rapport des surfaces entre celle-ci et la boîte mère, tel qu'illustré dans la figure II.8.

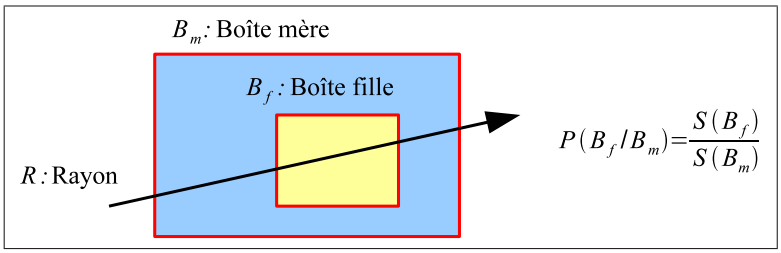


FIG. II.8 – Probabilité de passage d'un rayon dans une boîte fille.

Soit un rayon R passant dans une boîte mère B_m, la probabilité de passage $P(B_f|B_m)$ de ce rayon dans la boîte fille B_f est exprimée par :

$$P(B_f|B_m) = \frac{S(B_f)}{S(B_m)}$$

$S(B)$ désigne la surface de la boîte B tel que :

$$S(B) = 2 \times [d_x(B).d_y(B) + d_x(B).d_z(B) + d_y(B).d_z(B)]$$

avec $d_{axe}(B)$ la dimension de la boîte B pour l'axe donné.

A partir de la probabilité de passage $P(B_f|B_m)$, Mac Donald [47] propose un modèle de coût qui permet d'évaluer le coût de répartition $C_{repartition}$ d'une boîte en deux sous-boîtes :

$$\begin{aligned} C_{repartition}(B_m, B_{f1}, B_{f2}) &= P(B_{f1}|B_m).N_{surf}(B_{f1}) + P(B_{f2}|B_m).N_{surf}(B_{f2} \\ &= \frac{S(B_{f1}).N_{surf}(B_{f1}) + S(B_{f2}).N_{surf}(B_{f2})}{S(B_m)} \end{aligned}$$

$N_{surf}(B)$ désigne le nombre de surfaces contenues dans une boîte B. Cette fonction est appliquée à la boîte mère B_m, à la première boîte fille B_{f1} et à la seconde

boîte fille B_{f2}. Une façon intuitive de comprendre ce modèle est de considérer qu'il est préférable de regrouper un maximum de surfaces dans une petite boîte et à l'inverse de maximiser la taille des boîtes comportant peu ou pas de surfaces. On a alors respectivement une probabilité faible de calculer des intersections avec beaucoup de surfaces et une probabilité forte de calculer des intersections avec un nombre restreint de surfaces. Un algorithme de construction d'arbre cherchera simplement à minimiser la fonction $C_{repartition}$ dans sa méthode de subdivision.

2.2.1.2 Le critère d'arrêt Le critère d'arrêt de construction permet de décider si une boîte doit être subdivisée ou ne pas l'être. Pour cela, il est possible de définir a priori une profondeur maximale de l'arbre et un nombre minimal de surfaces par boîte. Mais plutôt que de fixer des paramètres qui sont difficiles à définir pour un ensemble de modèles hétérogènes, Havran [30] propose d'étendre la fonction de coût de Mac Donald *et al.* [47] pour évaluer la nécessité de subdivision. L'algorithme de recherche d'intersections est alors modélisé dans son ensemble en tenant compte du temps nécessaire à la traversée de la structure de recherche et du temps nécessaire aux calculs des intersections rayons/surfaces. Chaque étape de traversée est représentée par un temps de parcours T_{trav} et chaque calcul d'intersection est représenté par un temps de calcul T_{int}. Les coûts de subdivision et de non subdivision sont alors modélisés de la façon suivante :

$$\begin{aligned} C_{int}(B) &= N_{surf}(B).T_{int} \\ C_{trav}(B) &= T_{trav} \end{aligned}$$

$$\begin{aligned} C_{\overline{sub}}(B_m) &= C_{int}(B_m) \\ &= N_{surf}(B_m).T_{int} \end{aligned}$$

$$\begin{aligned} C_{sub}(B_m, B_{f1}, B_{f2}) &= C_{trav}(B_m) + P(B_{f1}|B_m).C_{int}(B_{f1}) + P(B_{f2}|B_m).C_{int}(B_{f2}) \\ &= T_{trav} + \frac{[S(B_{f1}).N_{surf}(B_{f1}) + S(B_{f2}).N_{surf}(B_{f2})] \times T_{int}}{S(B_m)} \end{aligned}$$

Le coût de non subdivision $C_{\overline{sub}}$ se comprend comme étant le coût de calcul d'une intersection avec les surfaces incluses dans la boîte mère B_m. Le coût de subdivision C_{sub} s'explique d'une part par la somme des calculs d'intersection avec les surfaces de la première boîte B_{f1} et de la seconde boîte B_{f2}, pondérés par les probabilités respectives de passage des rayons dans chacune des deux sous-boîtes. D'autre part vient s'ajouter un coût additionnel lié à la nécessité de progresser d'une étape supplémentaire dans l'arborescence, c'est à dire le passage d'un rayon de B_m à B_{f1} et/ou B_{f2}.

En considérant ces coûts, on peut déduire une méthode pour arrêter ou continuer la procédure de subdivision :

$$V_{sub}(B_m, B_{f1}, B_{f2}) = C_{sub}(B_m, B_{f1}, B_{f2}) < C_{\overline{sub}}(B_m)$$

La fonction V_{sub} rend obsolète tout paramètre de construction de l'arbre tels que la profondeur maximale ou le nombre minimal de surfaces. La construction est alors mieux adaptée à chaque modèle. Il reste néanmoins nécessaire d'évaluer T_{trav} et T_{int}, mais ces paramètres sont plus fonction des algorithmes utilisés et de leur implémentation que des modèles traités. Il est aisé, après avoir réalisé divers jeux de tests, de les déterminer empiriquement et de les fixer définitivement.

2.2.2 La réalité

Le modèle, tel que proposé, est composé d'une succession d'optimisations locales. Devant l'impossibilité calculatoire d'optimiser globalement la construction, des adaptations mineures de ce modèle sont proposées afin d'étendre la vision locale dans certains cas de figure prédéterminés.

2.2.2.1 Approche pour une optimisation globale La méthode de subdivision et le critère d'arrêt sont des heuristiques qui sont appliquées au niveau local pour décider au mieux de la subdivision d'une boîte. Or, comme tout principe d'optimisation, il est possible que le choix d'un optimum local ne permette pas d'atteindre l'optimum global. Il est donc nécessaire de minimiser le coût global d'un arbre plutôt que le coût local d'une boîte. Ce coût global peut être modélisé de la sorte :

$$C_{arbre}(B_{arbre}) = \overline{N}_{trav}(B_{arbre}).T_{trav} + \overline{N}_{int}(B_{arbre}).T_{int}$$

Dans cette fonction C_{arbre}, B_{arbre} désigne la boîte englobante du modèle, $\overline{N}_{trav}(B_{arbre})$ représente le nombre moyen de traversées par rayon et $\overline{N}_{int}(B_{arbre})$ représente le nombre moyen d'intersections par rayon. Soient $N_{noeud}(B_{arbre})$ le nombre de noeuds de l'arbre B_{arbre} et $N_{feuille}(B_{arbre})$ son nombre de feuilles, défini tel que $N_{feuille}(B_{arbre}) = N_{noeud}(B_{arbre}) + 1$, les nombres moyens de traversées et d'intersections sont approximés selon les formules suivantes :

$$\begin{aligned}
\overline{N}_{trav}(B_{arbre}) &= \sum_{i}^{N_{noeud}(B_{arbre})} P(B_{noeud_i}|B_a) \\
&= \frac{1}{S(B_{arbre})} \sum_{i}^{N_{noeud}(B_{arbre})} S(B_{noeud_i}) \\
\overline{N}_{int}(B_{arbre}) &= \sum_{i}^{N_{feuille}(B_{arbre})} P(B_{feuille_i}|B_a).N_{surf}(B_{feuille_i}) \\
&= \frac{1}{S(B_{arbre})} \sum_{i}^{N_{feuille}(B_{arbre})} S(B_{feuille_i}).N_{surf}(B_{feuille_i})
\end{aligned}$$

Pour mener à bien une optimisation globale, il est donc nécessaire de minimiser la fonction C_{arbre}. Ce problème est de type NP-Complet [30] et pour des raisons évidentes d'explosion combinatoire et de temps de calculs, une telle optimisation n'est pas réalisable dans le cadre d'un moteur de recherche d'intersections général. L'optimisation locale sera donc utilisée systématiquement par la suite. Il demeure tout de même intéressant d'évaluer la fonction C_{arbre} pour comparer la performance d'un algorithme par rapport à un autre, et de vérifier les améliorations ou les régressions.

Le problème des optimisations locales peut être aménagé dans certains cas particuliers connus pour améliorer l'optimisation globale. Ces cas particuliers s'observent seulement pour le kd-Tree car il est le seul arbre (par rapport au BVH) à pouvoir générer des feuilles vides et à pouvoir partager des triangles entre deux feuilles soeurs. Les limitations constatées sont le cas d'une feuille vide et le cas d'une topologie en coin.

2.2.2.2 Avantager les espaces vides Le choix d'un plan séparateur avec un kd-Tree peut amener l'heuristique à positionner toutes les surfaces d'un côté du plan et ainsi laisser un espace vide de surfaces de l'autre côté. La présence de ces espaces vides est grandement souhaitée et constitue notamment un des points forts de cette structure de recherche. Cela permet, par rapport à une boîte mère, de réduire la probabilité d'intersection de l'ensemble des surfaces. Cependant les espaces vides constituent un cas particulier qu'il convient d'avantager comme l'illustre la figure II.9. En détectant au plus tôt les espaces, il est possible de réduire significativement le nombre de noeuds de l'arbre et donc de diminuer le nombre moyen de traversées par rayon. De plus, lors de la traversée, une feuille vide est traitée plus rapidement qu'une feuille contenant des surfaces puisque l'appel de la fonction d'intersection rayon/surface est évité (voir la figure II.19).

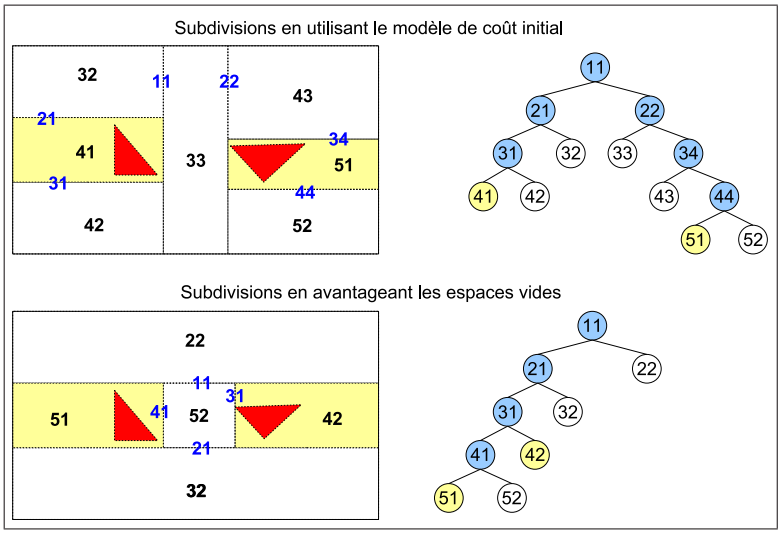


FIG. II.9 – Comparaison de deux kd-Tree générés avec et sans avantage donné aux espaces vides. Dans cet exemple, l'adaptation du modèle de coût permet de réduire le nombre de noeuds créés (9 contre 13), et de ramener au plus prêt de la racine les espaces vides, augmentant ainsi la probabilité d'arrêter au plus tôt la traversée de l'arbre.

Pour avantager les espaces vides, Wald [76] propose de biaiser la fonction de coût en attribuant un bonus. La fonction C_{sub} est alors corrigée selon le pseudo-algorithme suivant :

Fonction : C_{sub*}
Input: B_m, B_{f1}, B_{f2}
Input: $BonusEspaceVide$
Result: $Cout$
$Cout \longleftarrow C_{sub}(B_m, B_{f1}, B_{f2})$
if $N_{surf}(B_{f1}) = 0$ **or** $N_{surf}(B_{f2}) = 0$ **then**
  $Cout \longleftarrow Cout \times BonusEspaceVide$
end

Algorithm 1: Adaptation des espaces vides

Le principe utilisé est simple à mettre en oeuvre puisqu'il consiste à diminuer le coût de subdivision C_{sub} si l'une des feuilles issue de la subdivision ne comporte pas de surfaces.

2.2.2.3 Détecter les topologies en coin Un autre cas particulier est détecté pour les coins. On comprend par coin l'intersection de trois faces orthogonales alignées aux axes comme représentées dans la figure II.10. Pour simplifier la présentation du problème, les trois faces sont supposées être de même dimension comme celles d'un cube ayant un sommet en commun : le coin. Le problème apparaît dès le positionnement du premier plan séparateur. Celui-ci est accolé à l'une des trois

faces, dans notre cas F_B, ce qui implique la répartition suivante si l'on suppose que la face F_B est située à droite :

- à droite du plan séparateur, 3 faces sont référencées : la totalité de la F_B et une partie "epsilonesque" des faces F_A et F_C.
- à gauche du plan séparateur, 2 faces sont référencées : la quasi totalité des faces F_A et F_C.

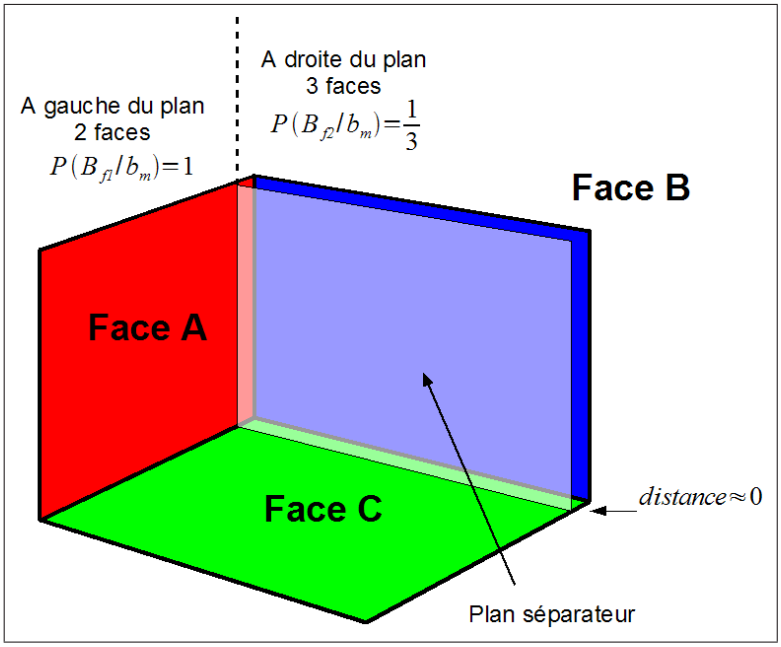


FIG. II.10 – Erreur de l'heuristique de surfaces pour des topologies en coin. Le critère d'arrêt rejette le positionnement d'un plan séparateur.

En supposant l'aire d'une face égale à 1 et en appliquant les répartitions constatées dans les coûts de subdivision, on obtient d'une part :

$$\begin{aligned}
C_{sub}(B_m, B_{f1}, B_{f2}) &= T_{trav} + [P(B_{f1}|B_m) \times 2 + P(B_{f2}|B_m) \times 3] \times T_{int} \\
P(B_{f1}|B_m) &= \frac{S(B_{f1})}{B_m} \simeq \frac{6}{6} \simeq 1 \\
P(B_{f2}|B_m) &= \frac{S(B_{f2})}{B_m} \simeq \frac{2}{6} \simeq \frac{1}{3} \\
C_{sub}(B_m, B_{f1}, B_{f2}) &= T_{trav} + [2 + \frac{1}{3} \times 3] \times T_{int} \\
&= T_{trav} + 3 \times T_{int}
\end{aligned}$$

D'autre part, le coût de non subdivision est :

$$C_{\overline{sub}}(B_m) = 3 \times T_{int}$$

On remarque alors que la fonction V_{sub} qui vérifie que $C_{sub} < C_{\overline{sub}}(B_m)$ ne pourra jamais valider cette subdivision pour $T_{trav} \geq 0$. Pourtant, il est important de réaliser

cette subdivision en accolant successivement un plan séparateur près de chaque face. Un moyen simple de parvenir à ces résultats est de toujours autoriser les subdivisions si toutes les surfaces sont alignées aux axes. Puisque la nature du kd-Tree impose que les plans séparateurs soient alignés aux axes, il sera toujours possible de subdiviser l'espace et l'on ne risque pas d'entrer dans une boucle sans fin. Ce cas particulier est bien sûr limité à des modèles comportant des coins avec des faces alignées ; les méthodes de construction des modèles font que ces cas se présentent souvent.

Ces deux cas particuliers constituent les seules dérogations recensées et adaptées par rapport au modèle de coût. On peut remarquer que l'arbre BVH ne comporte aucun de ces cas particuliers. Par définition, les boîtes de cet arbre englobent au plus près la géométrie ; l'espace vide n'est pas représenté en tant que tel, mais est le résultat de l'espace restant lorsque l'on soustrait l'espace des boîtes filles à l'espace de la boîte mère. De plus, comme les surfaces ne sont pas partagées entre les boîtes, des problèmes de chevauchement comme dans le cas de la topologie en coin ne peuvent pas exister.

Modèles	# feuilles vides (x 1000)			
	Référence	Bonus 80%	Détection Coin	Bonus 80% et Détection Coin
ERW6	0,3	0,5	0,3	0,6
Villa Digy	10,3	15,1	11,2	15,5
Conference	8,1	12,4	8,4	12,6
Clipper	30,5	37,6	30,6	37,5
Soda Hall	14,8	21,9	20,3	26,3

TAB. II.1 – Nombre de feuilles vides (en milliers) pour le kd-Tree en fonction de la favorisation des espaces vides par un bonus de 80% et de la détection des topologies en coin. Individuellement, ces dérogations aux modèles de coût permettent d'augmenter sensiblement le nombre de feuilles vides ce qui permet un parcours des rayons plus efficace. Utilisées conjointement, l'efficacité de ces deux cas particuliers n'est pas cumulée sauf pour des modèles hautement architecturaux (avec beaucoup de plans alignés aux axes) comme peut l'être le modèle *Soda Hall*.

2.3 Choix du plan séparateur

La construction des arbres kd-Tree et BVH est réalisée de manière récursive avec une approche descendante. Les pseudo-algorithmes ci-dessous (2, 3 et 4) montrent comment est réalisée cette construction d'une manière générique. La construction

totale d'un arbre est opérée en faisant un appel à $ConstruireArbre(B_{arbre})$.

Fonction : $ConstruireArbre$
Input: B_m
$B_{f1}, B_{f2} \longleftarrow SubdiviserBoite(B_m)$
if $B_{f1} \neq \emptyset$ **then**
  $ConstruireArbre(B_{f1})$
end
if $B_{f2} \neq \emptyset$ **then**
  $ConstruireArbre(B_{f2})$
end

Algorithm 2: Construction récursive d'un arbre

Fonction : $SubdiviserBoite$
Input: B_m
Output: B_f1, B_f2
$B_f1, B_f2 \longleftarrow RepartirSurfaces(B_m)$
if $C_{\overline{sub}}(B_m) < C_{sub*}(B_m, B_f1, B_f2)$ **then**
  $B_f1, B_f2 \longleftarrow \emptyset$
end

Algorithm 3: Subdivision d'une boîte

Fonction : $RepartirSurfaces$
Input: B_m
Output: B_f1, B_f2
$C_{meilleur} \longleftarrow \infty$
$p_{meilleur} \longleftarrow \emptyset$
forall $a \in x, y, z$ **do**
  $Plans = CreerPlans(B_m, a)$
  forall $p_{courant} \in Plans$ **do**
    $C_{repartition} \longleftarrow EvaluerPlan(B_m, p_{courant})$
    if $C_{repartition} < C_{meilleur}$ **then**
      $C_{meilleur} \longleftarrow C_{repartition}$
      $p_{meilleur} \longleftarrow p_{courant}$
    end
  end
end
$B_{f1}, B_{f2} \longleftarrow CreerSousBoites(Bm, p_{meilleur})$

Algorithm 4: Répartition des surfaces selon le meilleur plan séparateur

2.3.1 Domaine de recherche du plan séparateur

La difficulté majeure de construction du kd-Tree et de l'arbre BVH est de devoir trouver le plan séparateur qui minimise le coût de subdivision C_{sub}. Le modèle de

coût permet aisément de comparer la qualité de répartition d'un plan à un autre. Cependant, la construction d'un arbre doit être la plus rapide possible. Il convient alors, avant de choisir le meilleur plan, de limiter l'ensemble des plans potentiels. Ce rôle revient à la fonction $CreerPlans$ qui renvoie un ensemble choisi de plans pour un axe donné.

Des heuristiques simples permettent de ne sélectionner qu'un seul plan sans faire de calculs. Ces heuristiques sont réservées à des usages où la rapidité de construction prime sur la qualité de construction. Cela se révèle utile quand le nombre de requêtes d'intersection exécutées sur un modèle ne permet pas d'amortir le temps de construction de l'arbre. Ces heuristiques limitent dans un premier temps le domaine de recherche à l'axe de plus grande dimension de la boîte. Les trois heuristiques les plus courantes sont :

- le plan moyen : la moyenne des centres des surfaces définit la position de ce plan ;
- le plan médian : il s'agit du plan qui sépare équitablement le nombre de surfaces ;
- le plan moyen (ou médian) ajusté : il s'agit d'un des deux plans précédents que l'on déplace vers la surface la plus proche afin de ne pas créer d'espace vide inutile.

Ces types simples de positionnement ne permettent pas d'obtenir une qualité optimale de construction. Le temps de recherche des intersections peut alors être très largement augmenté en raison d'une mauvaise répartition des surfaces. A l'opposé, on peut envisager d'évaluer l'ensemble des plans potentiels. Dans le cas du kd-Tree le modèle de coût ne varie qu'aux limites d'une surface [31], l'ensemble des plans potentiels est composé par l'ensemble des limites des surfaces. Pour un groupe de N surfaces, on obtient alors un ensemble de $2N$ plans potentiels. On peut noter que la notion de plan séparateur est également utilisée pour l'arbre BVH. Cette approche offre l'avantage de séparer facilement les surfaces en deux ensembles distincts en comparant la position de leur centre au plan séparateur. Cependant, contrairement au kd-Tree, même une évaluation exhaustive de tous les plans pour chacun des axes ne permet pas de trouver la meilleure subdivision. Il est nécessaire dans ce cas d'évaluer toutes les combinaisons fournissant deux sous-ensembles de surfaces non vides. La complexité en $O(2^N)$ de cette recherche rend cette approche impraticable.

Le kd-Tree comporte une particularité par rapport à l'arbre BVH car les surfaces peuvent être partagées entre deux boîtes filles. Il est donc préférable pour cet arbre de ne pas considérer les limites des surfaces telles quelles, mais de considérer les limites de leurs boîtes englobantes ajustées aux dimensions de la boîte que l'on souhaite subdiviser (voir la figure II.11). La nécessité d'ajuster la boîte englobante des surfaces à chaque étape de subdivision ajoute du temps de construction additionnel compensé par une amélioration significative des performances de recherche d'intersections [29].

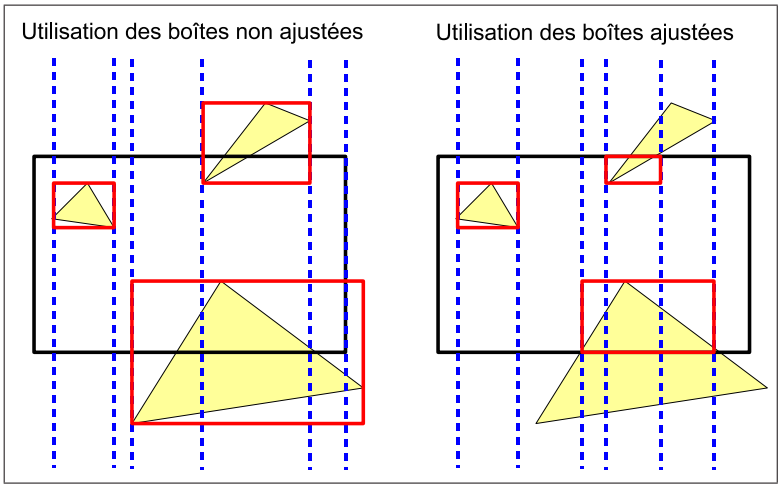


FIG. II.11 – Comparaison du positionnement des plans séparateurs aux limites des boîtes englobantes des triangles. L'ajustement permet de positionner les plans au plus prêt de la géométrie.

2.3.2 Tri par intervalles

2.3.2.1 Complexité de construction La complexité minimale de construction d'un arbre pour un modèle de N surfaces est théoriquement en $O(N\ log\ N)$. Si l'on considère l'ensemble des plans potentiels, cette complexité peut très vite augmenter avec un algorithme naïf. Par exemple, si la fonction $EvaluerPlan$ est définie par une boucle d'évaluation de chacune des surfaces (voir l'algorithme 5), la complexité devient $O(N^2)$.

Szecsi [73] a montré qu'il était possible de rendre la construction plus efficace tout en évaluant l'ensemble des plans potentiels. Pour cela, chaque limite de surfaces est mémorisée dans un tableau selon son type : début d'une surface ou fin d'une surface. Ces événements, correspondant aux plans potentiels, sont ensuite triés de manière croissante. En itérant sur chacun de ces événements, il est possible de construire incrémentalement le nombre de surfaces se situant à gauche et à droite de chaque plan potentiel, et ainsi d'évaluer par la même occasion le coût de la répartition. Etant donné le besoin de trier l'ensemble des événements, la complexité de cet algorithme est en $O(N\ log^2\ N)$.

Wald et Havran [79] se sont inspirés des travaux de Szecsi pour mettre au point un algorithme d'une complexité en $O(N\ log\ N)$. Pour y parvenir, les chercheurs ont enlevé la nécessité de trier les événements à chaque subdivision. Un tri initial est réalisé sur l'ensemble des surfaces et pour chaque axe. Pendant la construction, la liste des événements est séparée en sous-liste de manière à maintenir le tri initial. Cette façon de procéder revient à juxtaposer deux algorithmes en $O(N\ log\ N)$ plutôt que les imbriquer. Bien que la complexité minimale semble atteinte, derrière cette technique se cache une constante assez élevée qui implique toujours des temps de construction assez longs. De plus, la nécessité d'ajuster la boîte englobante des

surfaces avant d'évaluer la subdivision d'un noeud implique d'ajouter certains événements à la volée.

Fonction : $EvaluerPlan$
Input: B, p
Output: B_{f1}, B_{f2}
$F(B_{f1}) \longleftarrow \emptyset$
$F(B_{f2}) \longleftarrow \emptyset$
forall $f \in F(B)$ **do**
  if $PositionRelative(f, p) <= 0$ **then**
    $F(B_{f1}) \longleftarrow F(B_{f1}) \cup f$
  end
  if $PositionRelative(f, p) >= 0$ **then**
    $F(B_{f2}) \longleftarrow F(B_{f2}) \cup f$
  end
end

Algorithm 5: Evaluation naïve de la répartition des surfaces autour d'un plan séparateur. $F(B)$ représente l'ensemble des surfaces de la boîte B.

Arbre	Méthode	Temps relatif de construction	Efficacité relative d'exécution
kd-Tree	1024 intervalles	x1	x1
	256 intervalles	x0,96	x1,02
	64 intervalles	x0,97	x1,01
	16 intervalles	x0,96	x0,99
	1024 intervalles (boîtes non ajustées)	x0,66	x0,97
	Plan médian ajusté	x0,37	x0,64
BVH	1024 intervalles	x1	x1
	256 intervalles	x0,91	x0,99
	64 intervalles	x0,85	x0,98
	16 intervalles	x0,81	x0,94

TAB. II.2 – Efficacités relatives de différentes méthodes de positionnements du plan séparateur pour le kd-Tree et pour l'arbre BVH en terme de temps de construction et de temps d'exécution des requêtes d'intersection. Lors d'une construction par tri par intervalles, le nombre d'intervalles a relativement peu d'influence sur les temps de construction et les temps d'exécution. L'arbre BVH est toutefois plus sensible à la variation du nombre d'intervalles ; un petit nombre de 16 intervalles permet de réduire de près de 20% le temps de construction en occasionnant une perte légère de performances de 6%. Pour le kd-Tree, le fait de ne pas ajuster les boîtes englobantes des surfaces avant l'évaluation d'une subdivision ou de limiter la recherche du plan séparateur à un plan spatial médian ajusté peut diminuer d'un facteur 2 à 3 le temps de construction. L'efficacité d'exécution reste acceptable sans l'ajustement des boîtes, mais est significativement dégradée quand seul le plan médian ajusté est utilisé.

2.3.2.2 Tri par intervalle en $O(N\ log\ N)$ Pour des raisons d'efficacité et de simplicité, un tri par intervalles est proposé. Bien que similaire à des approches concurrentes [34, 57], cette technique été réalisée indépendamment. Le tri par intervalle n'évalue pas tous les plans séparateurs potentiels mais seulement un nombre fini d'intervalles décorrélé de la position des surfaces (voir la figure II.12). Le nombre de plans séparateurs N_{plan} est fonction du nombre d'intervalles N_Δ tel que $N_{plan} = N_\Delta + 1$. La taille d'un intervalle I_Δ est égale à la distance séparant chaque plan séparateur sur un axe donné.

L'implémentation de ce tri diffère quelque peu entre le kd-Tree et l'arbre BVH. Les pseudo-algorithmes 6 et 7 présentent ces différences. En terme de complexité, pour chaque axe, deux itérations sont réalisées sur l'ensemble des surfaces et trois itérations sur l'ensemble des intervalles. Le nombre d'axes étant constant et le nombre d'intervalles étant inférieur au nombre de surfaces, la complexité de répartition des surfaces est donc dominée par le nombre de surfaces, soit $O(N)$. La fonction de répartition des surfaces est appelée lors de chaque subdivision, soit $log\ N$ fois. L'imbrication de la répartition des surfaces dans la récursivité implique une complexité optimale en $O(N\ log\ N)$.

Fonction : $RepartirSurfaces\_kdTree$
Input: B_m
Output: B_f1, B_f2
$C_{meilleur} \longleftarrow \infty, p_{meilleur} \longleftarrow \emptyset$
$N_\Delta \longleftarrow min(MinN_\Delta, N(B_m) \times CoefN_\Delta)$
forall $a \in x, y, z$ **do**
  $I_{debut} \longleftarrow \infty, I_{fin} \longleftarrow -\infty$
  forall $f \in F(B_m)$ **do**
    $I_{debut} \longleftarrow min(I_{debut}, f_{debut}), I_{fin} \longleftarrow max(I_{fin}, f_{fin})$
  end
  $I_\Delta \longleftarrow (I_{fin} - I_{debut})/N_\Delta$
  for $i = 0 \rightarrow N_\Delta$ **do**
    $N_1[i], N_2[i] \longleftarrow 0$
  end
  forall $f \in F(B_m)$ **do**
    $i1 \longleftarrow 1 + \lfloor (f_{min} - I_{debut})/I_\Delta \rfloor, i2 \longleftarrow \lfloor (f_{max} - I_{debut})/I_\Delta \rfloor$
    $N_1[i1] \longleftarrow N_1[i1] + 1, N_2[i2] \longleftarrow N_2[i2] + 1$
  end
  for $i = 0 \rightarrow N_\Delta - 1$ **do**
    $N_1[i+1] \longleftarrow N_1[i+1] + N_1[i]$
    $N_2[N_\Delta - i - 1] \longleftarrow N_2[N_\Delta - i - 1] + N_2[N_\Delta - i]$
  end
  for $i = 0 \rightarrow N_\Delta$ **do**
    $B_{f1}, B_{f2} \longleftarrow InitialiserSousBoites(N_1[i], N_2[i], Plans[i])$
    $C_{repartition} \longleftarrow C_{sub*}(B_m, B_{f1}, B_{f2})$
    if $C_{repartition} < C_{meilleur}$ **then**
      $C_{meilleur} \longleftarrow C_{repartition}, p_{meilleur} \longleftarrow Plans[i]$
    end
  end
end
$B_{f1}, B_{f2} \longleftarrow CreerSousBoites(Bm, p_{meilleur})$

Algorithm 6: Evaluation de la répartition des surfaces par tri par intervalle pour le kd-Tree.

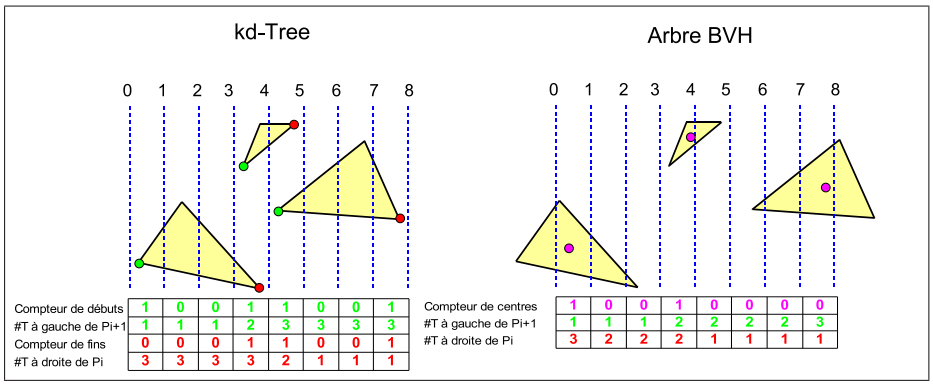


FIG. II.12 – Comparaison des méthodes de tris par intervalle entre le kd-Tree et l'arbre BVH. Dans les deux cas, des plans séparateurs sont positionnés à intervalles réguliers. L'évaluation du nombre de triangles situés de part et d'autre de chaque plan peut alors être réalisée en discrétisant les limites, ou le centre, des triangles.

Fonction : $RepartirSurfaces\_BVH$
Input: B_m
Output: B_f1, B_f2
$C_{meilleur} \longleftarrow \infty, p_{meilleur} \longleftarrow \emptyset$
$N_\Delta \longleftarrow min(MinN_\Delta, N(B_m) \times CoefN_\Delta)$
forall $a \in x, y, z$ **do**
  $I_{debut} \longleftarrow \infty, I_{fin} \longleftarrow -\infty$
  forall $f \in F(B_m)$ **do**
    $I_{debut} \longleftarrow min(I_{debut}, f_{centre}), I_{fin} \longleftarrow max(I_{fin}, f_{centre})$
  end
  $I_\Delta \longleftarrow (I_{fin} - I_{debut})/N_\Delta$
  for $i = 0 \rightarrow N_\Delta - 1$ **do**
    $BB_1[i], BB_2[i] \longleftarrow \emptyset$
    $N_c[i] \longleftarrow 0$
  end
  forall $f \in F(B_m)$ **do**
    $i \longleftarrow \lfloor (f_{centre} - I_{debut})/I_\Delta \rfloor$
    $BB_1[i], BB_2[i] \longleftarrow BB_1[i] \cup BB(f)$
    $N_c[i] \longleftarrow N_c[i] + 1$
  end
  for $i = 0 \rightarrow N_\Delta - 2$ **do**
    $BB_1[i+1] \longleftarrow AABB_1[i+1] \cup BB_1[i]$
    $BB_2[N_\Delta - i - 1] \longleftarrow BB_2[N_\Delta - i - 1] \cup BB_2[N_\Delta - i]$
    $N_c[i+1] \longleftarrow N_c[i+1] + N_c[i]$
  end
  for $i = 0 \rightarrow N_\Delta - 2$ **do**
    $N_1 \longleftarrow N_c[i], N_2 \longleftarrow N(F(B_m)) - N_c[i+1]$
    $B_{f1}, B_{f2} \longleftarrow InitialiserSousBoites(N_1, N_2, BB_1[i], BB_2[i])$
    $C_{repartition} \longleftarrow C_{sub}(B_m, B_{f1}, B_{f2})$
    if $C_{repartition} < C_{meilleur}$ **then**
      $C_{meilleur} \longleftarrow C_{repartition}, p_{meilleur} \longleftarrow Plans[i]$
    end
  end
end
$B_{f1}, B_{f2} \longleftarrow CreerSousBoites(Bm, p_{meilleur})$

Algorithm 7: Evaluation de la répartition des surfaces par tri par intervalle pour le BVH.

Première itération sur les surfaces. Une première itération est réalisée sur l'ensemble des surfaces afin de déterminer les limites minimales et maximales à partir de points de repère liés aux surfaces. Dans le cas du kd-Tree, les limites de la boîte englobante des surfaces sont utilisées tandis que pour le BVH on exploite seulement le centre des surfaces. Cette différence tient du fait que le plan séparateur utilisé pour le BVH n'est qu'un plan virtuel qui sert à répartir les surfaces en deux groupes.

Première itération sur les intervalles. A partir de ces limites, le segment

est divisé en N_Δ intervalles tel que $N_\Delta = min(MinN_\Delta, N(B_m) \times CoefN_\Delta)$. Les constantes $MinN_\Delta$ et $CoefN_\Delta$ désignent respectivement un nombre minimum d'intervalles à définir et un coefficient multiplicateur permettant de définir un nombre d'intervalles proportionnel au nombre de surfaces. Un nombre N_Δ d'intervalles génère $N_{plan} + 1$ pour le kd-Tree et $N_{plan} - 1$ pour le BVH car les deux plans limites ne sont pas utiles. Une boucle sur l'ensemble des intervalles permet d'initialiser des compteurs :

- pour le kd-Tree, un tableau $N1$ contenant le nombre de surfaces situées à gauche du plan p_{i+1} ;
- pour le kd-Tree, un tableau $N2$ contenant le nombre de surfaces situées à droite du plan p_i ;
- pour l'arbre BVH, un tableau BB dont les éléments représentent les boîtes englobantes des surfaces dont les centres sont situés dans les intervalles appropriés ;
- pour l'arbre BVH, un tableau N_c contenant le nombre de surfaces projetées dans chaque intervalle ;

Seconde itération sur les surfaces. Lors d'une seconde itération sur l'ensemble des surfaces, les points de repères de chaque surface sont projetés afin de déterminer l'intervalle associé. Les compteurs sont alors incrémentés selon leur type.

Seconde itération sur les intervalles. Puisque les compteurs ne représentent qu'un état local de l'intervalle, une seconde itération sur les intervalles permet de cumuler de proche en proche les compteurs afin de connaître, pour chaque plan situé entre chaque intervalle, les répartitions en terme de nombre de surfaces et en terme de dimensions de boîtes.

Troisième itération sur les intervalles. Disposant alors de toutes les données pour calculer le coût de la répartition, une troisième et dernière itération sur l'ensemble des intervalles permet d'évaluer la qualité de chaque plan. Le plan ayant le coût le plus faible est retenu et utilisé par la suite pour effectuer la réelle subdivision.

2.4 Construction paresseuse

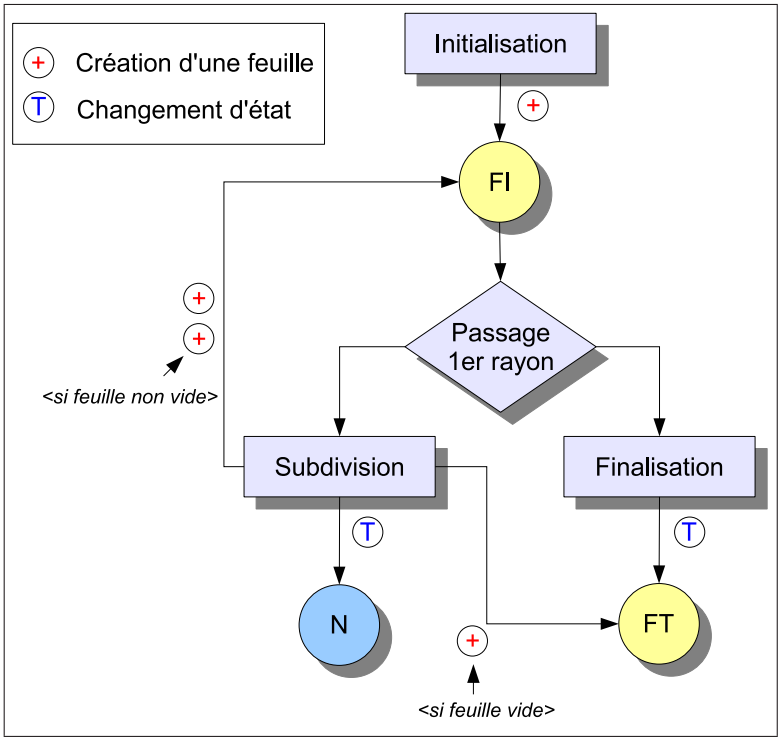


FIG. II.13 – Principe de la création paresseuse à trois états : noeud N, feuille intermédiaire FI et feuille terminale FT.

L'intérêt d'une structure de recherche est d'accélérer le traitement des requêtes d'intersection. Cependant ce bénéfice est obtenu au détriment d'un temps de prétraitement et d'une occupation mémoire additionnels. Pour minimiser ces problèmes, il est intéressant de ne raffiner l'arbre qu'aux seuls endroits où sont effectuées les requêtes d'intersection. La construction paresseuse [7, 75] répond à ce problème en ne raffinant la structure de recherche qu'au moment où celle-ci est traversée par un rayon.

2.4.1 Principe

Dans un arbre, les éléments sont classés en noeuds (internes) ou en feuilles. Par définition, un noeud est un élément qui possède des fils tandis que les feuilles sont des éléments terminaux. Les surfaces sont référencées dans les feuilles. La construction paresseuse dissocie deux types de feuilles : les feuilles intermédiaires et les feuilles terminales. Une feuille intermédiaire est une feuille qui peut encore être subdivisée. Son état n'est pas défini, il le sera seulement lorsqu'un rayon la traversera. Par opposition, une feuille terminale est une feuille qui ne peut plus être subdivisée.

La figure II.13 illustre les différents types d'éléments et les étapes de création et de transformation au cours de la construction paresseuse. Une seule étape de prétraitement est nécessaire pour construire une feuille intermédiaire unique contenant

toutes les surfaces, c'est la racine de l'arbre. Ensuite, pendant l'exécution d'une requête d'intersection, lorsqu'un rayon traverse une feuille intermédiaire, une procédure de subdivision est exécutée. Si la procédure de subdivision échoue, la feuille intermédiaire est transformée en feuille terminale et celle-ci est directement exploitée. Si la procédure de subdivision réussit, la feuille intermédiaire est transformée en noeud et deux nouvelles feuilles intermédiaires sont créées. Le kd-Tree comporte une exception sur ce point car une des deux feuilles peut être vide, elle sera alors directement identifiée en tant que feuille terminale. La requête d'intersection se poursuit à partir de la feuille intermédiaire nouvellement transformée.

2.4.2 Implémentation

La mise en place de la construction paresseuse est simple de réalisation. Il suffit de rajouter un état sur les feuilles pour dissocier celles qui sont intermédiaires de celles qui sont finales. Dans la fonction de subdivision toute nouvelle feuille est intermédiaire sauf si elle est vide. Dans la fonction de recherche d'une d'intersection, si une feuille intermédiaire est rencontrée, il suffit d'appeler la procédure de subdivision et de traiter le nouveau type : noeud ou feuille terminale. Toutefois certains algorithmes de construction d'arbre nécessitent des données propres à la construction qui sont par la suite "nettoyées" lorsque la construction est terminée. Une construction paresseuse n'offre pas cette souplesse et il est important que la fonction de subdivision dispose de tous les éléments pour pouvoir être exécutée.

Si l'on considère l'arbre BVH, chaque élément de l'arbre comporte une information sur la boîte englobante et la liste des surfaces. Ces éléments sont suffisants à la procédure de subdivision. Un codage compact de cette structure peut être réalisé sur 32 octets [77]. Dans le cas du kd-Tree, l'information de boîte englobante n'est pas nécessaire lors de l'exécution de la requête d'intersection, seule la position du plan séparateur, l'axe de séparation et la liste des surfaces sont nécessaires. Le codage compacte d'un kd-Tree peut être alors réduit à 8 octets [82]. Il n'est pas envisageable d'étendre cette structure avec le volume englobant qui utiliserait alors 24 octets de plus : 6×4 octets (taille d'un réel 32 bits).

L'information de boîtes englobantes est cependant utile aux feuilles intermédiaires. Ce type de feuilles ne représente qu'une fraction des éléments d'un arbre comme le montre le tableau II.3. Il est plus intéressant d'allouer une zone mémoire distincte pour ces données plutôt que d'ajouter le codage d'une boîte englobante sur chaque élément de l'arbre. La gestion de cette zone mémoire, implémentée par un système d'allocation par page, permet une allocation rapide ainsi que la réutilisation des boîtes englobantes pour le cas où les feuilles intermédiaires sont transformées en noeud ou en feuilles terminales. Cette solution a l'avantage de minimiser l'occupation mémoire et de maintenir le codage compact des éléments du kd-Tree. Un simple index est nécessaire pour référencer la boîte englobante. Il peut être intégré sans conséquences dans la structure générale d'un noeud (voir la figure II.14).

```
typedef struct {
  union {
    //2 bits de poids faible 01, 10, 11 : axe de separation
    //2 bits de poids faible 00 : feuille ...
    //... si bb_index = 0 alors "finale" ...
    //... sinon "intermediaire"
    int axis;
    //30 bits de poids forts : index sur une boite
    int bb_index; //Utile aux feuilles itermediaires
    //30 bits de poids forts : index sur le premiere feuille
    int child_index; //Utile aux noeuds
  } //4 octets
  union {
    //Position du plan separateur pour le noeud interne
    float split_location;
    //index sur une liste de surfaces ...
    //... pour la feuille intermediaire ou finale
    int surface_index;
  } //4 octets
} KDTreeNode;
```

FIG. II.14 – Structure d'un noeud du kd-Tree.

2.5 Résultats

kd-Tree										
	Const. Totale		Construction Paresseuse					Statistiques		
Modèles	# FT	Mém.	# FT	# FI	Mém.	T_p	T_t/T_p	C_a	$\overline{S}$	$\overline{P}$
erw6	3,7	0,1 Mo	1,3	0,5	0,1 Mo	0,1 s	x1,0	20,3	2,5	18,7
bunny	126,0	3,1 Mo	46,4	2,7	1,5 Mo	2,1 s	x1,8	71,9	2,1	23,6
villa digy	168,9	4,5 Mo	26,2	6,8	1,4 Mo	1,6 s	x2,5	39,2	2,6	24,6
fairy	216,9	6,0 Mo	67,1	10,3	3,1 Mo	5,3 s	x1,6	67,5	2,9	25,3
conf	303,7	9,8 Mo	20,6	7,1	2,3 Mo	3,5 s	x3,7	66,8	4,1	25,5
armadillo	537,7	13,4 Mo	178,3	24,0	6,9 Mo	14,7 s	x1,6	68,9	2,3	27,3
clipper	164,8	9,2 Mo	76,0	9,7	6,5 Mo	13,6 s	x1,3	36,1	10,0	24,4
sully	883,1	28,3 Mo	27,5	9,2	4,6 Mo	6,9 s	x5,1	105,2	4,0	26,4
buddha	1 171,7	31,8 Mo	381,4	45,6	16,2 Mo	30,8 s	x1,7	127,5	2,8	26,9
soda hall	3 997,6	110,2 Mo	49,2	27,6	11,8 Mo	24,5 s	x4,4	101,8	2,9	29,6
Arbre BVH										
erw6	0,14	0,1 Mo	0,12	0,01	0,1 Mo	0,1 s	x1,0	24,3	5,9	9,0
bunny	14,16	1,2 Mo	6,3	0,72	0,7 Mo	0,4 s	x1,3	40,9	4,9	14,0
villa digy	16,3	1,4 Mo	5,18	1,3	0,7 Mo	0,4 s	x1,4	37,5	5,0	16,7
fairy	33,86	2,9 Mo	12,91	4,71	1,8 Mo	1,4 s	x1,2	39,5	5,1	17,3
conf	56,89	4,8 Mo	4,05	2,08	1,5 Mo	1,1 s	x2,3	56,6	5,0	18,6
armadillo	71,22	5,9 Mo	33,05	4,05	3,7 Mo	3,2 s	x1,2	35,8	4,9	16,3
clipper	70,65	6,9 Mo	21,89	11,84	4,5 Mo	4,3 s	x1,2	12,5	8,3	18,2
sully	133,18	11,6 Mo	5,04	1,71	3,4 Mo	3,7 s	x2,0	134,8	5,7	19,2
buddha	213,71	18,0 Mo	83,25	14,38	10,6 Mo	9,8 s	x1,2	70,9	5,1	18,1
soda hall	410,37	34,8 Mo	9,12	8,23	9,7 Mo	10,6 s	x2,3	94,9	5,2	21,9

TAB. II.3 – Résultats de construction de différents modèles pour le kd-Tree et l'arbre BVH. # FT désigne le nombre de feuilles terminales et # FI le nombre de feuilles intermédiaires. T_p désigne le temps de construction paresseux et T_t le temps de construction total. Dans les données statistiques, C_a représente le coût global de l'arbre pour des coûts fixes d'intersection T_{int} de 1 et de traversée T_{trav} de $1,5$. $\overline{S}$ désigne le nombre moyen de surfaces par feuille et $\overline{P}$ la profondeur moyenne de l'arbre.

Le tableau II.3 présente des résultats de construction pour un large ensemble de modèles aussi bien pour le kd-Tree que pour l'arbre BVH. Les deux premiers groupements de colonnes permettent de comparer les temps et l'occupation mémoire d'une construction totale contre une construction paresseuse des arbres. Les données de la construction paresseuses sont recueillies lors d'un rendu en résolution 1024x1024. On peut constater que cette construction à la demande permet d'accélérer notablement le temps de construction (jusqu'à $\times 5,1$) et de limiter considérablement l'occupation mémoire (jusqu'à $\times 10$).

En comparant les résultats entre le kd-Tree et l'arbre BVH, ce dernier est plus

rapide à construire et plus léger en occupation mémoire. Cela s'explique essentiellement par l'appartenance unique d'une surface à un noeud contrairement au kd-Tree qui partage les surfaces. Ces capacités limitent quelque peu les bénéfices de la paresse sans toutefois les effacer.

En observant le nombre moyen de surfaces par feuille, on peut remarquer que les arbres permettent efficacement de réduire la complexité de recherche d'intersections pour un rayon pour l'ensemble des modèles.

2.6 Discussion

Dans cette section nous avons traité de la construction des structures de recherche et présenté notamment deux d'entre elles, le kd-Tree et l'arbre BVH, jugées complémentaires car appartenant à deux familles distinctes. Nous avons ensuite montré des techniques performantes qui permettent de construire ces arbres dans un souci d'efficacité en temps, en mémoire et en adaptation à de nombreux modèles. Les techniques de tri par intervalle et de construction paresseuse ont été intégrées de la même manière, à quelques détails d'implémentation près, pour les deux structures de recherche.

Bien qu'important dans le domaine de la synthèse d'image, la problématique des scènes dynamiques n'a pas été traitée spécifiquement. Diverses approches existent à ce sujet. Certaines ajustent un arbre existant en fonction des surfaces qui ont bougé [46, 77]. La mise à jour est alors efficace au début des mouvements mais la qualité de l'arbre se dégrade dans le temps. D'autres approches parient sur la rapidité de construction de l'arbre [75, 69] pour pouvoir le reconstruire totalement à chaque mouvement de géométrie. Cette approche n'est appropriée qu'à des scènes de taille moyenne. En exagérant cette vision, Reshetov [65] propose de ne construire que des structures d'accélération transitoires qui ne servent qu'au passage d'un groupe de rayons. Enfin Wald *et al.* [40], pour rémédier à la dégradation de l'arbre dans le temps, propose de dédier en parallèle une tâche à la création d'un nouvel arbre et une autre tâche à la mise à jour de l'arbre courant. Lorsque le nouvel arbre est créé, il est utilisé pour les mises à jour en remplacement de l'arbre courant, et la tâche de création est réinitialisée.

Dans RayBooster, l'utilisation de plusieurs databases autorise la répartition des surfaces statiques et des surfaces dynamiques dans des structures de recherche distinctes. Cette répartition des surfaces permet de réduire le temps de reconstruction de la database contenant les surfaces en mouvement. De plus l'utilisation conjointe de techniques de construction rapides et paresseuses permet de traiter interactivement les scènes dynamiques.

3 Traversée et intersections

La qualité de construction des structures de recherche est très importante pour exécuter efficacement les requêtes d'intersection. Elle permet de réduire algorith-

miquement le nombre d'étapes de traversée et de calculs d'intersection par rayon. Cependant cette qualité ne suffit pas à elle seule à fournir une exécution efficace. Les fonctions de traversée et d'intersection jouent également un rôle important. De leur implémentation dépend la performance finale de l'intersecteur.

Cette section présente la définition de ces deux fonctions tant en terme algorithmique qu'en terme d'implémentation. Pour chacune d'elles sont présentées d'abord l'exécution individuelle de rayons simples puis l'exécution groupée de quatre rayons exploitant les instructions vectorielles SIMD.

3.1 Traversée

3.1.1 Rayons simples

La traversée d'une structure hiérarchique peut être réalisée de manière séquentielle ou récursive. La forme séquentielle [30] consiste à déplacer un point de recherche le long du rayon, du début à la fin. Pour un point donné, une recherche récursive est réalisée dans l'arbre pour trouver la feuille qui le contient, et calculer les intersections. Les calculs débutent à partir de l'origine du rayon et les points suivants sont déterminés, de proche en proche, par les points de sortie du rayon dans les feuilles. On remarque que cet algorithme est loin d'être optimal car chaque point engendre une recherche depuis la racine de l'arbre bien que la feuille suivante puisse être une feuille soeur ou une feuille très proche dans la hiérarchie.

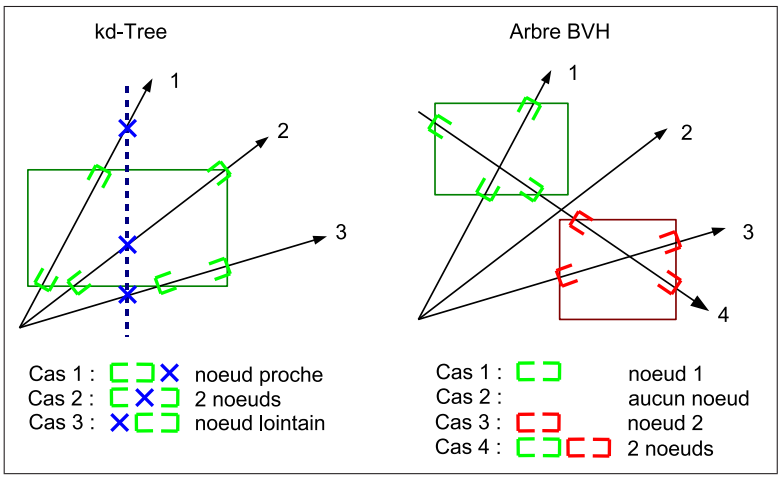


FIG. II.15 – Comparaison des différents cas de traversée possibles entre le kd-Tree et l'arbre BVH.

La forme récursive améliore cette stratégie en propageant les rayons depuis la racine de l'arbre. Pour chaque noeud, un calcul d'intersection avec les noeuds fils détermine le parcours du rayon dans la hiérarchie. La méthode de calcul de ces intersections diffère selon le kd-Tree ou l'arbre BVH (voir la figure II.15). Pour le kd-Tree, une simple intersection avec le plan séparateur permet de situer le rayon par rapport aux noeuds fils. Dans le cas de l'arbre BVH, il est nécessaire de calculer

l'intersection avec les deux sous-boîtes. Ces calculs sont expliqués et comparés ci-dessous.

3.1.1.1 Intersection d'un rayon avec un plan séparateur Un rayon est défini par l'ensemble de points R tel que :

$$R(t) = O + t.D$$

Dans cette équation, O désigne l'origine du rayon, D son vecteur directeur, et t la variable de distance à l'origine du rayon tel que $t \geq 0$.

L'équation d'un plan est représentée par l'égalité suivante :

$$N.p + K = 0$$

N représente le vecteur normal au plan, $|K|$ la distance de ce plan à l'origine, et p tout point appartenant à ce plan. En remplaçant la variable p dans l'équation du plan par l'équation d'un rayon, on obtient $N(O+tD)+K = 0$. Le plan séparateur est un plan particulier puisqu'il est aligné aux axes. Les vecteurs normaux de ce plan sont alors limités aux valeurs suivantes : $N_X = (1, 0, 0)$, $N_Y = (0, 1, 0)$ et $N_Z = (0, 0, 1)$. En considérant ces vecteurs unitaires, la résolution de l'intersection entre un rayon et un plan séparateur peut être simplifiée de la manière suivante :

$$\begin{aligned} N_{axe}(O + tD) + K &= 0 \\ O_{axe} + t.D_{axe} + K &= 0 \\ t &= \frac{-K - O_{axe}}{D_{axe}} \end{aligned} \tag{II.1}$$

La valeur $-K$ représente alors la position du plan dans l'axe pour lequel il est défini, et t la distance d'intersection au plan par rapport à l'origine du rayon. En considérant pour le rayon les distances du point d'entrée tn et du point de sortie tf pour une boîte donnée, il suffit de calculer la distance ts au plan séparateur pour déterminer les sous-boîtes à traverser (voir la figure II.15) :

- $tf < ts$: traversée de la sous-boîte la plus proche de l'origine du rayon ;
- $ts < tn$: traversée de la sous-boîte la plus éloignée de l'origine du rayon ;
- $tn < ts < tf$: traversée des deux sous-boîtes.

3.1.1.2 Intersection d'un rayon avec une boîte Une boîte peut être représentée par six plans séparateurs. Chaque axe comporte alors deux plans séparateurs dont un des plans est plus proche de l'origine du rayon. Pour déterminer le plan le plus proche, il suffit d'étudier le signe du vecteur directeur du rayon. Ainsi, pour une direction positive sur un axe, le plan le plus proche sera celui ayant la position la plus faible et inversement pour une direction négative. En calculant pour les trois axes les distances plus proches tn_x, tn_y, tn_z et les distances les plus lointaines tf_x, tf_y, tf_z selon l'équation II.1, il est possible de déterminer pour un rayon les distances tn du point d'entrée et tf du point de sortie dans une boîte. La figure II.17 illustre

en 2D les propriétés suivantes $tn = min(tn_x, tn_y, tn_z)$ et $tf = min(tf_x, tf_y, tf_z)$. Le rayon intersecte alors la boîte si et seulement si $tn \leq tf$.

```
//Points min et max de la boite
float bb[2][3];
node_bb(n, bb);
tn = r->tmin; tf = r->tmax;
for (int a = 0; a < 3; a++) { //Pour chacun des axes...
  //Distance au plan "a" le plus proche
  const float tin = (bb[r->s[a]][a] - r->o[a]) * r->id[a];
  //Distance au plan "a" le plus loin
  const float tout = (bb[1-r->s[a]][a] - r->o[a]) * r->id[a];
  tn = max(tin, tn);
  tf = min(tout, tf);
  //Sortie rapide si non intersection
  if (tn > tf) return 0;
}
//Intersection
return 1;
```

FIG. II.16 – Fonction d'intersection d'un rayon et d'une boîte

La figure II.16 montre l'implémentation d'une fonction qui pour un noeud n donné et un rayon r donné indique la validité d'une intersection. On peut remarquer que la fonction ajuste incrémentalement les valeurs tn et tf pour permettre une sortie plus rapide de la fonction en cas de non intersection.

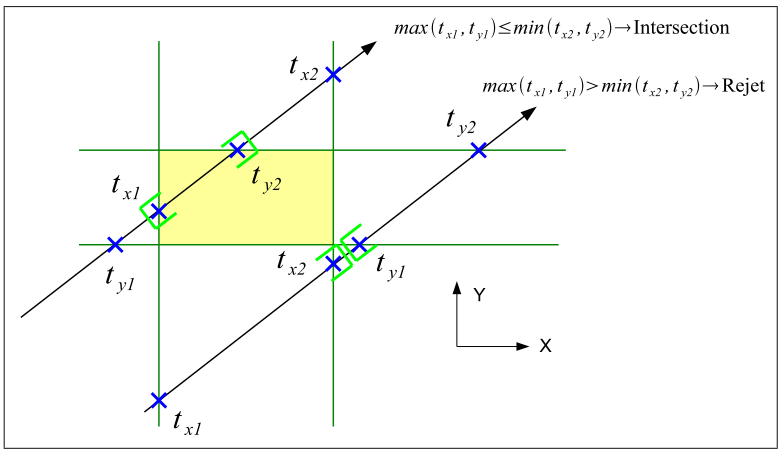


FIG. II.17 – Intersection d'un rayon avec une boîte selon les distances d'entrée et de sortie aux axes.

```
typedef struct {
  float o[3]; //origine
  float d[3]; //direction
  float id[3]; //1 / direction
  int s[3]; //s[i] = d[i] < 0 ? 1 : 0
  //Distance de recherche
  //généralment comprise entre  0 et +INF.
  float tmin, tmax;
  //distance au point d'impact
  float ti;
  //coordonnées barycentriques
  float b, g;
  //identifiant de triangle
  int tid;
} Ray;
```

FIG. II.18 – Structure d'un rayon.

```
float tn, tf;
Node *n = root;
ray_clip(r, n, &tn, &tf);
if (tn > tf) return 0; //pas d'intersection avec le modèle
do {
  while (!is_leaf(n)) { //recherche d'une feuille
    const int a = split_axis(n);
    //distance au plan séparateur
    const float ts = (n->split_location - r->o[a]) * r->id[a];
    //plan complètement à gauche du segment
    if (ts < tn) n = right_child(n, r->s[a]);
    //plan complètement à droite du segment
    else if (ts > tf) n = left_child(n, r->s[a]);
    else { //intersection du plan
      //sous-segment droit : ts..tf
      push_node(right_child(n, r->s[a]), ts, tf);
      //sous-segment gauche : tn..ts
      n = left_child(n, r->s[a]);
      tf = ts;
    }
  }
  //cas d'une feuille intermediaire et subdivision paresseuse
  if (!is_terminal(n) && node_split(n)) continue;
  //intersection avec une feuille non vide
  if (!is_empty(n) && ray_surface(r, n, tn, &tf)) return 1;
} while(pop_node(n, tn, tf)); //noeud suivant...
return 0; //aucun d'impact
```

(a) Fonction de traversée du kd-Tree.

```
float tn = r->tmin, tf = r->tmax;
Node *n = root;
int hit = 0;
do {
  if (ray_box(r, n, tn, tf)) { //intersection rayon/boite
    if (!is_leaf(n)) { //cas d'un noeud, traitement des fils
      const int a = split_axis(n);
      //empilement du sous-noeud droit et traitement du sous-noeud gauche
      push_node(right_child(n->child_index, r->s[a]));
      n = left_child(n->child_index, r->s[a]);
      continue;
    }
    else {
      //cas d'une feuille intermediaire et subdivision paresseuse
      if (!is_terminal(node) && node_split(n)) continue;
      //intersection rayon/surfaces
      hit = ray_surface(r, n, tn, &tf);
    }
  }
} while (pop_node(n)); //noeud suivant...
return hit;
```

(b) Fonction de traversée de l'arbre BVH.

FIG. II.19 – Fonctions de traversée des structures de recherche. *ray_clip*() est une fonction similaire aux calculs d'intersection rayon/boîte (voir la figure II.16) exceptée que celle-ci renvoie les distances aux points d'entrée tn et de sortie tf. *is_leaf*(), *is_terminal*() et *is_empty*() sont des fonctions d'états d'un noeud qui indiquent respectivement si un noeud est une feuille quelconque, une feuille terminale ou une feuille vide. *split_axis*(), *left_child*() et *right_child*() sont des fonctions d'accès aux données sur un noeud qui renvoient respectivement l'axe du plan séparateur, le sous-noeud le plus proche et le sous-noeud le plus lointain. *push_node* et *pop_node* sont des fonctions pour empiler/dépiler un noeud. Dans le cas du kd-Tree, en plus du noeud sont stockées les distances limites du segment pour un rayon. *node_split* est une fonction de subdivision paresseuse d'un noeud qui indique si la subdivision a eu lieu ou non. *ray_surface* est une fonction de calculs d'intersection avec les surfaces du noeud ; elle indique si une intersection a été trouvée.

3.1.1.3 Parcours récursif L'implémentation du parcours récursif du kd-Tree et de l'arbre BVH n'est pas réalisée par appels récursifs de fonctions mais de manière itérative. L'itération suit un parcours de l'arbre en profondeur d'abord. Lorsqu'il est nécessaire de visiter deux sous-boîtes, seule la plus proche de l'origine du rayon est immédiatement traitée tandis que l'autre est empilée [76]. Quand une feuille est atteinte et si la pile n'est pas vide, alors l'itération continue avec le noeud dépilé. Cette façon de procéder évite alors des appels coûteux de fonctions.

Les figures II.20(a) et II.20(b) montrent une implémentation de parcours récursif pour le kd-Tree et l'arbre BVH. Les paramètres des fonctions de traversée sont un rayon r et la racine de l'arbre $root$ (de type $Node$) tels que décrits dans la section 2.4 à la page 33). La structure simplifiée d'un rayon est présentée dans la figure II.18.

Les principales différences qui opposent ces fonctions de traversée sont le test d'intersection avec un noeud pour déterminer les sous-noeuds visités, et la condition d'arrêt de la traversée. Dans le cas du kd-Tree, l'ordre de traversée des noeuds est toujours dans le sens du rayon. Etant donné que les boîtes d'un kd-Tree ne se chevauchent pas, si une intersection a été trouvée dans une boîte, il est alors certain qu'aucune intersection plus proche ne pourra être trouvée par la suite. Avec l'arbre BVH, cette propriété n'est pas garantie, il est donc nécessaire de vérifier toutes les boîtes empilées avant d'arrêter la recherche d'intersection. Cette différence est fondamentale entre les deux arbres et sera étudiée plus en détails dans le chapitre III à la page 52. On peut remarquer que les boîtes sont empilées sans vérifier immédiatement si le rayon les intersecte. Ce test est volontairement reporté au dépilement car une intersection valide avec une surface peut raccourcir la distance maximale de recherche tf pour un rayon et éviter ainsi par la suite le parcours de certaines boîtes empilées situées plus loin que tf.

3.1.2 Rayons SIMD

3.1.2.1 Instructions SIMD Depuis le support d'instructions SIMD (Single Instruction Multiple Data) sur plusieurs architectures [39, 52, 19], il est désormais possible d'exécuter en une seule instruction la même opération pour un vecteur de N valeurs. Un algorithme scalaire peut alors être transformé pour exploiter ces instructions SIMD et voir son traitement accéléré par un facteur proche de N. L'implémentation d'Intel des instructions SIMD, nommée SSE [39], est utilisée dans les expérimentations de cette thèse. L'utilisation de ce jeu d'instructions est facilitée par les macros fonctions *intrinsics* pouvant être intégrées dans un langage de haut niveau évitant ainsi le codage en assembleur.

Dans un moteur de lancer de rayons, les types de valeurs traités sont majoritairement des entiers et des réels 32 bits. Comme les registres SSE ont une taille fixe de 128 bits, ces types de valeurs peuvent être traités par paquet de quatre limitant ainsi l'accélération potentielle par un facteur quatre. La relation directe entre un registre SSE et un vecteur de quatre valeurs implique que les données de ce vecteur doivent être contiguës en mémoire et alignées sur 16 octets (128 $bits/8$).

Pour bénéficier au maximum des performances SIMD, il est important de toujours disposer de quatre valeurs avec lesquelles on puisse exécuter la même opération. Ceci a pour conséquences de devoir réorganiser les structures de données et de transformer les algorithmes scalaires en algorithmes vectoriels.

3.1.2.2 Application au calcul de traversée L'algorithme de traversée est un algorithme séquentiel dans le sens où chaque opération dépend de la précédente. Ce type d'algorithme n'est pas facilement parallélisable. Par contre à un plus haut niveau, les requêtes d'intersection sont fortement parallélisables puisque chaque rayon peut être traité indépendamment. Wald *et al.* [82] propose alors de regrouper les rayons par groupe de quatre et de dérouler les opérations de traversée et d'intersections de surfaces en parallèle grâce aux instructions SIMD. Pour disposer de la même séquence d'opérations, les rayons doivent traverser les mêmes noeuds et les mêmes feuilles, et intersecter les mêmes surfaces. Or chaque requête d'intersection peut suivre un chemin différent au gré des besoins du générateur de rayons. Une contrainte forte impose alors que les rayons regroupés soient cohérents. Selon les domaines d'application, il est possible de trouver un générateur de rayons apportant beaucoup de cohérence. Dans la technique du lancer de rayons, les rayons primaires partent de la position de la caméra et passent par chaque pixel de l'image. En regroupant les rayons de quatre pixels voisins, un motif cohérent est grandement assuré. D'une manière générale, dès que l'échantillonnage en rayons est dense, il est possible et utile d'exploiter des rayons SIMD (voir la figure II.20). La figure II.21 montre une adaptation de la traversée du kd-Tree avec des rayons SIMD.

```
typedef _MM_ALIGN16 struct
{
  __m128 o4[3]; //Origine
  __m128 d4[3]; //Direction
  __m128 id4[3]; //1 / Direction
  int s[3]; //s[i] = d[i] < 0 ? 1 : 0
  //Contraintes sur la distance de recherche
  //Normalement 0 et +INF.
  __m128 tmin4, tmax4;
  //Distance au point d'impact
  __m128 ti4;
  //Coordonnées barycentriques
  __m128 b4, g4;
  //Identifiant de triangle
  __m128i tidi4;
} Ray4;
```

FIG. II.20 – Structure SIMD d'un rayon.

```
__m128 tn4, tf4, active4, active_start4, hit4;
Node *n = root;
ray_clip4(r4, n, &tn4, &tf4);
active_start4 = _cmpngt_ps(tn4, tf4);
//pas d'intersection avec le modèle
if (!_movemask_ps(active_start4)) return 0;
active4 = active_start4;
hit4 = _setzero_ps();
do {
  while (!is_leaf(n)) { //recherche d'une feuille
    const int a = split_axis(n);
    const __m128 s4 = _set1_ps(n->split_location);
    //distance au plan sépérateur
    const __m128 ts4 = _mul_ps(_sub_ps(s4, r4->o4[a]), r4->id4[a]);
    const __m128 left4 = _and_ps(_cmpnlt_ps(ts4, tn4), active4);
    //4 rayons a droite
    if (!_movemask_ps(left4)) n = right_child(n, r4->s[a]);
    else {
      const __m128 right4 = _and_ps(_cmpngt_ps(ts4, tf4), active4);
      //traversee du fils gauche obligatoire
      n = left_child(n, r4->s[a]);
      //4 rayons a gauche
      if (!_movemask_ps(right4)) continue;
      //traversee des deux fils
      push_node4(right_child(n, r4->s[a]), _max_ps(tnear4, ts4), tfar4, right4);
      active4 = left4;
      tf4 = _min_ps(ts4, tf4);
    }
  }
  //construction paresseuse
  if (!is_terminal(node) && node_split(node)) continue;
  //intersection avec une feuille non vide
  if (!is_empty(n)) {
    hit4 = _or_ps(hit4, ray_surface4(r4, n, tn4, &tf4, active4));
    //sortie rapide dans le cas ou tous les rayons sont occultes
    if (!_movemask_ps(_andnot_ps(hit4, active_start4))) break;
  }
  while (pop_node4(n, ts4, tf4, active4)) { //noeud suivant...
    active4 = _andnot_ps(hit4, active4);
    if (_movemask_ps(active4)) break;
  }
} while (n);
return _movemask_ps(hit4);
```

FIG. II.21 – Fonction de traversée SIMD du kd-Tree. Tous les préfixes "_mm"' des macros fonctions *intrinsics* ont été enlevés pour plus de lisibilité.

On peut noter que tous les scalaires de la structure du rayon SIMD (voir la figure II.20) ont été remplacés par des vecteurs de taille quatre, excepté le champ s qui représente un codage du signe de la direction des rayons. Les rayons SIMD doivent avoir les mêmes signes de direction car ceux-ci influent sur l'ordre de traversée des feuilles. Il n'est donc pas utile de dissocier la valeur s pour les quatre rayons puisqu'elle est commune au groupe.

A la différence de l'algorithme de traversée de rayons simples, il est nécessaire de maintenir une variable d'état des rayons : *active*4. Les rayons SIMD sont supposés cohérents mais il n'est pas rare que les rayons divergent lors d'une bifurcation. En exploitant cette variable, il est possible d'appliquer les calculs pour les seuls rayons concernés par une branche divergente et de masquer les calculs pour les autres rayons.

3.2 Intersections

Un moteur de requêtes d'intersection ne pourrait être performant sans un algorithme d'intersection rayon/surface efficace. Bien que les types de surfaces peuvent être très différents, RayBooster ne traite en interne que des triangles. Cette approche se justifie par la concentration des efforts d'optimisation sur une surface simple et par l'existence d'un grand nombre de mailleurs capables de transformer tous types de surfaces en triangles. De plus, comme évoqué dans la section 1.2.1 à la page 8, RayBooster permet d'intégrer un intersecteur personnalisé et supporte de cette manière tout type de surface.

3.2.1 Rayons simples

L'intersection rayon/triangle est un calcul largement traité par la communauté [5, 51, 70, 76] dont la forme la plus courante est le calcul des coordonnées barycentriques. Selon les implémentations, il est possible de calculer l'intersection à partir des simples coordonnées de sommets ou de procéder à des précalculs sur les triangles permettant un traitement plus efficace par la suite. Bien que cette dernière technique occasionne un temps de prétraitement additionnel, il est possible de limiter les précalculs aux triangles potentiels avec une stratégie de construction paresseuse. La méthode de calcul d'intersection retenue est celle de Wald [76] fondée sur une projection en 2D du triangle pour réduire le nombre d'opérations.

3.2.1.1 Intersection par coordonnées barycentriques Ce calcul s'effectue en deux étapes (voir la figure II.22) : un calcul de la distance au plan du triangle suivi d'un test d'inclusion du point d'intersection dans le triangle. Un calcul d'intersection est tout d'abord réalisé entre le rayon et le plan du triangle pour en déterminer la distance t_i. Si le plan du triangle est aligné aux axes, une version simplifiée de ce calcul (voir l'équation II.1) est utilisée. Comme le montre le tableau II.4, cette optimisation est plus ou moins efficace selon la proportion de surfaces alignées dans un modèle. Si t_i se situe en dehors de l'intervalle de validité du rayon $[t_{min}, t_{max}]$, alors l'intersection est rejetée. Sinon, le point d'intersection P est calculé tel que $P = R(t_i)$, soit $P = O + t_i.D$. A partir de ce point, les coordonnées barycentriques [5] sont calculées. Pour un triangle constitué des trois sommets S_a, S_b et S_c, ce calcul revient à trouver les coefficients α et β tel que :

$$\overrightarrow{S_aP} = \alpha\overrightarrow{S_aS_b} + \beta\overrightarrow{S_aS_c}$$

Les coefficients de ce système sont alors trouvés en résolvant ce système de trois équations :

$$\begin{aligned} p_x - a_x &= \alpha(b_x - a_x) + \beta(c_x - a_x) \\ p_y - a_y &= \alpha(b_y - a_y) + \beta(c_y - a_y) \\ p_z - a_z &= \alpha(b_z - a_z) + \beta(c_z - a_z) \end{aligned} \tag{II.2}$$

Si les coefficients trouvés respectent les conditions telles que $\alpha \geq 0$, $\beta \geq 0$ et $\alpha+\beta \leq 1$ alors le point se situe dans le triangle.

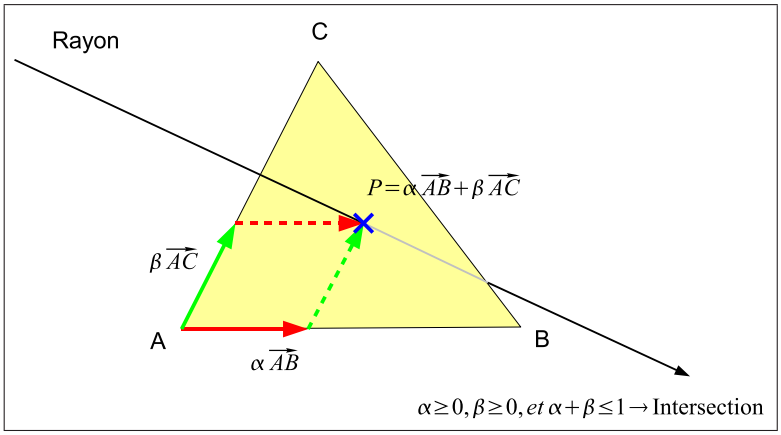


FIG. II.22 – Test d'intersection rayon/triangle par coordonnées barycentriques.

3.2.1.2 Projection 2D La projection 2D [76] permet de réduire le nombre d'opérations pour le calcul des coordonnées barycentriques. Le fait d'enlever une dimension permet de réduire à deux le nombre d'équations du système II.2, et ainsi de faciliter la résolution des coefficients. Cette réduction s'appuie sur une propriété d'invariance des coordonnées barycentriques entre un triangle 3D et sa projection 2D. Pour une meilleure précision numérique, le triangle est projeté sur l'axe dominant de la normale. Les figures II.23 et II.24 présentent les données nécessaires au précalcul d'un triangle ainsi que le code simplifié d'intersection rayon/triangles selon la technique de la projection 2D. La fonction d'intersection rayon/triangles prend en paramètre un rayon r, un noeud n et les distances d'entrée tn et de sortie tf dans le noeud.

```
typedef struct
{
  //Normale simplifiée : N / nw
  //Soit pour tout point P(pu, pv, pw)
  //nu*pu + nv*pv + pw - nd = 0
  float nu, nv, nd; //12 octets
  //Axe dominant
  int axis; //4 octets
  //Soit le point H(hu, hv)
  //beta =  hu*bu + hv*bv + bd;
  //gamma = hu*gu + hv*gv + gd;
  float bu, bv, bd; //12 octets
  float gu, gv, gd; //12 octets
  //Identifiant du triangle
  int tid; //4 octets
  //Complément à 48 octets
  int pad; //4 octets
} Triangle; //48 octets
```

FIG. II.23 – Structure des données d'accélération du triangle.

```
const int m3[5] = {0,1,2,0,1};
int hit = 0;
tn = max(r->tmin, tn), tf = min(r->tmax, tf);
Triangle *tri = first_tri(n);
while (tri) { //pour tous les triangles
  const int kw = tri->axis, ku = m3[kw+1], kv = m3[kw+2];
  //distance au plan
  float t;
  if (is_tri_aligned(acc)) t = (tri->nd - r->o[kw]) * r->id[kw];
  else {
    const float iden = 1 / (r->d[kw] + tri->nu*r->d[ku] + tri->nv*r->d[kv]);
    t = (tri->nd - r->o[kw] - tri->nu*r->o[ku] - tri->nv*r->o[kv]) * iden;
  }
  if (t >= tn && t <= tf) { //distance valide ?
    const float hu = r->o[ku] + t * r->d[ku];
    const float hv = r->o[kv] + t * r->d[kv];
    const float b = hu * tri->bu + hv * tri->bv + tri->bd;
    //point dans le triangle ?
    if (b >= 0) {
      const float g = hu * tri->gu + hv * tri->gv + acc->gd;
      if (g < 0 || g + b > 1) continue;
      hit_fill(r, t, b, g, tri->tid);
      tf = t - EPS;
      hit = 1;
    }
  }
  tri = next_tri(n); //triangle suivant...
}
return hit;
```

FIG. II.24 – Fonction d'intersection rayon/triangles. $first\_tri(), next\_tri()$ sont des fonctions d'itération sur la liste des triangles d'un noeud. $is\_tri\_aligned()$ est une fonction qui indique si un triangle est aligné à un axe ($tri.nu = 0$ *and* $tri.nv = 0$). $hit\_fill()$ est une fonction qui remplit la structure d'un rayon avec les données du point impacté.

Modèles	# tests intersection	# tests alignés	# rejets par distance	# rejets par coord. baryc.	# impacts
Conference	9,5	3,9 (41%)	6,84 (72%)	1,5 (16%)	1,2 (12%)
Fairy	8,5	0,8 (10%)	5,41 (64%)	1,8 (21%)	1,3 (15%)
Soda Hall	5,1	3,7 (73%)	2,95 (58%)	1,1 (20%)	1,1 (22%)

TAB. II.4 – Nombre de calculs d'intersection rayon/triangle (en millions) pour un rendu de 1024x1024 rayons primaires. Les chiffres entre parenthèses se rapportent à la proportion du nombre de la colonne par rapport au nombre total de tests d'intersection. Le calcul simplifié de la distance avec les triangles alignés est largement avantagé avec les scènes architecturales comme *Conference* ou *Soda Hall*. Les rejets par distance et par coordonnées barycentriques sont les deux barrières avant la validation d'un impact. On constate que le rejet par distance permet d'éliminer près de deux tiers des triangles et que les rejets par coordonnées barycentriques permettent d'éliminer la moitié des tests restants.

3.2.2 Rayons SIMD

De la même manière que la fonction de traversée, la fonction d'intersection (voir la figure II.25) peut être vectorisée pour bénéficier des instructions SIMD.

```
const int m3[5] = {0,1,2,0,1};
__m128 hit4;
Triangle *tri = first_tri(n);
hit4 = _setzero_si128();
tn4 = _max_ps(r4->tmin4, tn4), tf4 = min(r->tmax4, tf4);
while (tri) { //pour tous les triangles
  const int kw = tri->axis, ku = m3[kw+1], kv = m3[kw+2];
  const __m128 nd4 = _set1_ps(tri->nd), nu4 = _set1_ps(tri->nu),
    nv4 = _set1_ps(tri->nv);
  __m128 t4, m4;
  //distance au plan
  if (is_tri_aligned(acc))
    t4 = _mul_ps(_sub_ps(nd4, r4->o4[kw]), r4->id4[kw]);
  else {
    const __m128 den4 = _add_ps(r4->d4[kw], _add_ps(
      _mul_ps(nu4, r4->d4[ku]), _mul_ps(nv4, r4->d4[kv])));
    const __m128 num4 = _sub_ps(nd4, _add_ps(r4->o4[kw], _add_ps(
      _mul_ps(nu4, r4->o4[ku]), _mul_ps(nv4, r4->o4[kv]))));
    //division Newton-Raphson
    const __m128 iden4 = _rcp_ps(den4);
    t4 = _mul_ps(num4,
      _sub_ps(_add_ps(iden4, iden4), _mul_ps(den4, _mul_ps(iden4, iden4))));
  }
  m4 = _and_ps(active4, _and_ps(_cmpge_ps(t4, tn4), _cmple_ps(t4, tf4)));
  if (_movemask_ps(m4) > 0) { //distance valide ?
    const __m128 bu4 = _set1_ps(tri->bu), bv4 = _set1_ps(tri->bv),
      bd4 = _set1_ps(tri->bd);
    const __m128 hu4 = _add_ps(_mul_ps(r4->d4[ku], t4), r4->o4[ku]);
    const __m128 hv4 = _add_ps(_mul_ps(r4->d4[kv], t4), r4->o4[kv]);
    const __m128 b4 = _add_ps(bd4,_add_ps(_mul_ps(hu4, bu4), _mul_ps(hv4, bv4)));
    m4 = _and_ps(m4, _cmpge_ps(b4, ZERO4));
    //point dans le triangle ?
    if (_movemask_ps(m4) > 0) {
      const __m128 gu4 = _set1_ps(tri->gu), gv4 = _set1_ps(tri->gv),
        gd4 = _set1_ps(tri->gd);
      const __m128 g4 = _add_ps(gd4,
        _add_ps(_mul_ps(hu4, gu4), _mul_ps(hv4, gv4)));
      m4 = _and_ps(m4, _and_ps(_cmpge_ps(g4, ZERO4),
        _cmple_ps(_add_ps(b4, g4), ONE4)));
      if (!_movemask_ps(m4)) continue;
      hit_fill4(r4, t4, b4, g4, tri->tid, m4);
      tf4 = _or_ps(_andnot_ps(m4, tf4), _and_ps(_sub_ps(t4, EPS4, m4));
      hit4 = _or_ps(hit4, m4);
    }
  }
  tri = next_tri(n); //triangle suivant...
}
return hit4;
```

FIG. II.25 – Fonction d'intersection SIMD rayon/triangles.

On peut remarquer que la division dans le calcul de la distance au plan t est remplacée par la méthode de Newton-Raphson qui fonctionne par itération pour converger vers une solution exacte. L'utilisation directe de la fonction réciproque SSE *_mm_rcp_ps* ne permet pas d'obtenir une précision suffisante. Cette fonction est volontairement dégradée pour n'utiliser que 12 bits pour le codage de la mantisse

(contre 24 bits pour une division classique). Cependant le résultat obtenu est une très bonne valeur initiale pour la méthode de Newton-Raphson, et permet en une seule itération d'obtenir une valeur de bonne précision ($\simeq$ 22 bits). Il est bien sûr possible d'utiliser simplement la fonction classique de division $\_mm\_div\_ps$ afin d'obtenir la précision maximale pour environ 15% de cycles machine additionnels.

La difficulté principale d'implémentation des rayons SIMD réside dans la nécessité d'exécuter les mêmes opérations pour les quatre rayons même s'ils divergent. Dans le calcul d'intersection rayon/triangles, la variable d'état $m4$ est maintenue pour masquer les rayons qui ne suivent pas le contrôle logique. Par rapport au calcul scalaire, ces opérations de masquage occasionnent quelques calculs additionnels.

3.3 Discussion

Dans cette section, nous avons présenté des algorithmes performants pour traiter efficacement la requête d'intersection pour les deux structures de recherche choisies : le kd-Tree et l'arbre BVH. Ces algorithmes couvrent les deux fonctions majeures qui sont la traversée de la structure de recherche et les calculs d'intersection rayon/-triangle aussi bien pour des rayons simples que pour des rayons SIMD.

Le code d'intersection rayon/triangles présente une optimisation en considérant les triangles alignés. La nature des modèles fait que ces triangles particuliers sont souvent présents, et même majoritaires dans le cas des scènes architecturales. De plus, nous n'avons pas remarqué de pénalités occasionnées par un test supplémentaire dans le cas où cette optimisation ne serait pas applicable.

Le groupe de quatre rayons correspond exactement à la largeur SIMD et bénéficie directement d'une accélération matérielle. Pour des groupes de rayons plus grands, seules des optimisations algorthmiques peuvent apporter un gain de performance supplémentaire. Les expérimentations des rayons simples et des rayons SIMD sont alors reportées au chapitre III à la page 52 qui traite des regroupements de rayons. Cela permet de comparer directement les performances obtenues selon la taille du groupe : un rayon, quatre rayons, ou plus de quatre rayons.

Conclusion

Dans ce chapitre, nous avons introduit la requête d'intersection et montré son potentiel d'application dans tous les domaines de modélisation de la propagation d'ondes par lancer de rayons. La librairie RayBooster [33] a été développée afin de fournir une souplesse d'utilisation de cette fonction tout en cachant la complexité des nombreuses optimisations indispensables à une exécution efficace.

L'efficacité de cette requête est fondée sur l'utilisation de structures de recherche qui autorise une complexité logarithmique de l'algorithme de recherche d'une intersection. Nous avons présenté des méthodes pour construire rapidement et avec

qualité ces structures. Pour cela, nous nous sommes appuyés sur une technique d'évaluation paresseuse pour ne construire que les éléments nécessaires à l'évaluation d'un rayon et une technique de tri par intervalles qui s'affranchit d'un ordonnancement préalable ou à la volée des surfaces. Deux structures de recherche arborescentes ont fait l'objet d'une étude particulière : le kd-Tree et l'arbre BVH.

Les meilleurs algorithmes de l'état de l'art ont été intégrés pour parcourir les structures de recherche et calculer les intersections rayon/surface. Ceux-ci bénéficient de codage de structures compactes, de fonctions itératives en lieu et place de fonctions récursives, et d'un minimum de tests conditionnels gage d'une bonne prédiction du flot d'exécution pour le processeur. Ces algorithmes ont été déclinés pour le kd-Tree et l'arbre BVH et supportent aussi bien des rayons simples que des rayons SIMD.

Dans un souci d'amélioration des performances, il est intéressant d'avancer dans les techniques de regroupement de rayons autre que le rayon SIMD. L'étude des groupes de plus de quatre rayons est présentée dans le chapitre III sur la page suivante en distinguant deux implémentations distinctes pour le kd-Tree et l'arbre BVH.

Chapitre III

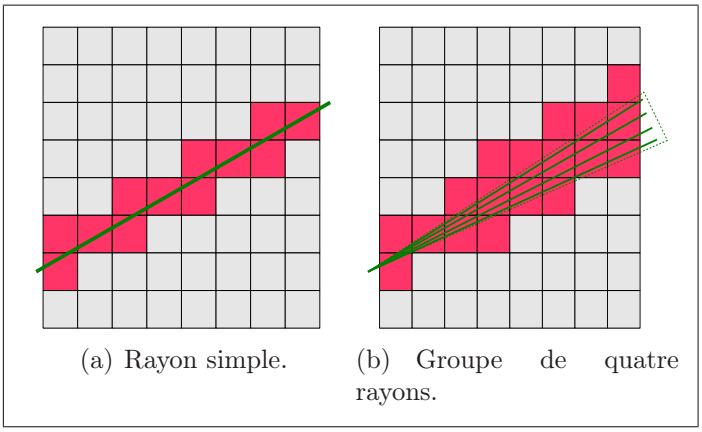
Le regroupement de rayons

Introduction

L'amélioration de performance de la simple requête d'intersection approche une asymptote ; en effet depuis l'apparition d'implémentations optimisées de construction et de traversée du kd-Tree [31, 76], et des algorithmes d'intersection rayon/-triangle très efficaces [76, 42], la complexité calculatoire n'a plus réellement diminué. Le chapitre II à la page 4 présente une implémentation efficace de ces algorithmes. Seules diverses "astuces" d'implémentation et le support plus ou moins étendu de fonctionnalités permettent de différencier un moteur de (simples) requêtes d'intersection d'un autre.

En exploitant des instructions vectorielles, les rayons SIMD [82] ont permis d'améliorer les performances nominales par un facteur 2 à 3 (voir les tableaux III.3 et III.10). De plus, la nature hautement parallélisable de la requête d'intersection permet d'exploiter efficacement les architectures multi-coeurs des processeurs modernes avec un facteur d'accélération quasi équivalent au nombre de coeurs. Toutefois, ces dernières optimisations résultent d'une évolution des capacités de calculs et non d'évolutions algorithmiques.

Le regroupement de rayons est une approche très attractive qui permet de factoriser des calculs de traversées et d'intersections. Cette technique a fait l'objet de nombreux travaux ces dernières années [31, 21, 66, 77, 65]. Pour que la factorisation soit effective, le regroupement doit être réalisé avec un ensemble de rayons *cohérents*. La portée exacte du terme *cohérent* n'est pas simple à définir. Il dépend de nombreux paramètres tels que la proximité et l'alignement des rayons entre eux, l'agencement et les dimensions des surfaces dans une scène, et la nature de la structure de recherche. Ce chapitre fait l'hypothèse de rayons cohérents pour ses expérimentations en exploitant essentiellement des rayons de types primaires. Le chapitre V à la page 117 traite plus en détails de la problématique de cohérence et apporte des éléments de solutions pour exploiter les rayons divergents.



(a) Rayon simple. (b) Groupe de quatre rayons.

FIG. III.1 – Schémas de traversée de rayons dans une structure d'accélération de type grille régulière. (a) 12 étapes de traversée pour le rayon simple. (b) 18 étapes de traversée pour le groupe de quatre rayons : soit moins de cinq étapes de traversée par rayon.

Bien que les techniques de regroupement de rayons puissent être employées avec des rayons simples, les travaux ont été essentiellement menés pour des groupes constitués de rayons SIMD plus performants. Des techniques de construction de groupes de rayons sont présentées dans la première section. Ensuite, des techniques d'accélération des intersections entre un groupe de rayons et un triangle sont présentées. Enfin, les deux dernières sections montrent des méthodes d'exécution des groupes de rayons, d'abord avec le kd-Tree puis avec l'arbre BVH.

1 Construction d'un groupe de rayons

1.1 Problématique

Pour pouvoir factoriser les calculs, il est nécessaire que le groupe de rayons soit représenté par une unique forme géométrique que l'on puisse manipuler indépendamment des rayons. Cette entité doit alors permettre de réaliser le parcours de la structure de recherche et des calculs d'intersections de surfaces au titre de tous les rayons. En utilisant une forme géométrique de type enveloppe convexe, il est possible de garantir des opérations conservatives. La construction d'un telle enveloppe peut devenir un traitement complexe si sa représentation n'est pas simplifiée. Une approche souvent retenue est d'approximer cette forme par quatre plans à l'intérieur desquels sont contenus tous les rayons. Une autre approche est de modéliser le groupe par quatre rayons encadrants. Cette dernière approche a été privilégiée en raison d'une plus grande efficacité et facilité d'implémentation. On peut noter que les deux approches forment la même enveloppe convexe car les rayons encadrants sont issus de l'intersection des quatre plans.

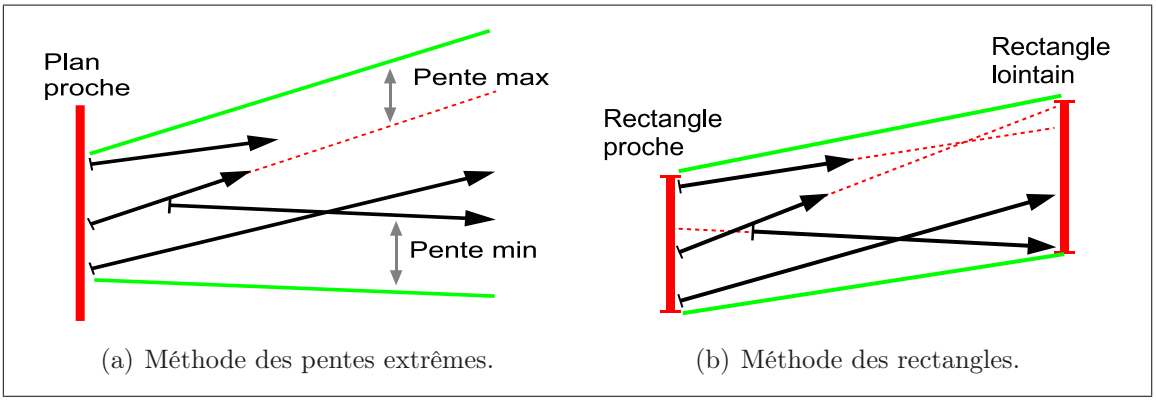
D'un point de vue fonctionnel, la requête d'intersection est étendue pour recevoir d'une part les quatre rayons encadrants et d'autre part le groupe de rayons à traiter.

La nécessité de cohérence est donc reléguée à un traitement spécifique en amont de cette fonction ; ce point est discuté en détails dans le chapitre V à la page 117 traitant des problèmes de divergences. L'utilisation de quatre rayons offre l'avantage d'être directement exploitable avec les instructions SIMD. La requête d'intersection traitant des groupes de rayons a donc directement été implémentée en utilisant les instructions SIMD aussi bien pour les quatre rayons encadrants que pour l'ensemble de N rayons SIMD formant le groupe de $4N$ rayons cohérents.

La construction des quatre rayons encadrants est réalisée en deux temps. Les plans de l'enveloppe sont calculés d'abord, puis utilisés ensuite pour déterminer la direction des rayons. Cette construction doit être très efficace car, l'objectif étant de diminuer le nombre de calculs pour accélérer les traitements, il n'est pas envisageable d'ajouter un temps additionnel par rapport aux rayons simples. Dans un souci de performance et de simplicité, les plans choisis pour former l'enveloppe sont contraints d'être parallèles à un axe. Cette restriction réduit le nombre de variables libres pour modéliser un plan à un angle et une distance par rapport à un axe. Deux approches sont proposées pour parvenir à la construction des quatre rayons encadrants. La première, fondée sur l'utilisation des pentes extrêmes des rayons, reprend une méthode proposée par Boulos *et al.* [11] ; la seconde est dérivée d'une méthode de construction de faisceau proposée par Reshetov [65].

1.2 Méthode des pentes extrêmes

Un axe principal est utilisé pour construire les plans. Tous les rayons doivent alors partager le même signe pour cette direction. En partant de l'hypothèse que le groupe est constitué de rayons cohérents, l'axe majoritaire k_w du premier rayon du groupe est choisi comme axe principal. De part et d'autre de chaque axe secondaire k_u et k_v, deux plans sont positionnés de manière à englober les rayons intérieurs comme présenté sur la figure III.2.



(a) Méthode des pentes extrêmes. (b) Méthode des rectangles.

FIG. III.2 – Méthodes de construction du faisceau encadrant un groupe de rayons.

En considérant pour un rayon donné les projections 2D de son vecteur directeur $D(d_u, d_v, d_w)$ sur les deux plans $\widehat{k_w k_u}$ et $\widehat{k_w k_v}$, on obtient deux vecteurs $U(d_u, 0, d_w)$

et $V(0, d_v, d_w)$. En normalisant par d_w, ces vecteurs se réécrivent $U(d_u/d_w, 0, 1)$ et $V(0, d_v/d_w, 1)$. Les valeurs $\alpha = d_u/d_w$ et $\beta = d_v/d_w$ représentent les pentes du rayon pour chaque axe secondaire. En calculant les pentes minimales et maximales de chaque rayon, on obtient les pentes extrêmes des plans.

Les normales aux plans sont orthogonales aux vecteurs extrêmes suivants :

$$\begin{array}{cc} U_{min}(\alpha_{min}, 0, 1) & U_{max}(\alpha_{max}, 0, 1) \\ V_{min}(0, \beta_{min}, 1) & V_{max}(0, \beta_{max}, 1) \end{array}$$

En veillant à ce que les normales soient orientées vers l'extérieur du groupe, leurs expressions s'écrivent :

$$\begin{array}{cc} NU_{min}(-1, 0, \alpha_{min}) & NU_{max}(1, 0, -\alpha_{max}) \\ NV_{min}(0, 1, -\beta_{min}) & NV_{max}(0, -1, \beta_{max}) \end{array}$$

Disposant des vecteurs normaux aux plans, il ne reste plus qu'à déterminer les distances de ces plans à l'origine pour que les équations de plans soient complètes. Pour cela, les distances maximales sont calculées par intégration de chaque origine de rayon. Une implémentation globale de la méthode est présentée dans la figure III.3. L'implémentation de la construction prend en compte la symétrie des opérations quand le signe de la direction principale est négatif. La fonction prend en entrée un tableau *rays* de *nr* rayons et remplit le tableau n avec les quatre équations de normales. On peut remarquer que lors du calcul des pentes, la normalisation des vecteurs directeurs par d_w ne nécessite pas de division car l'inverse de la direction pour chaque rayon est précalculé. Le précalcul est justifié par le fait que cette réciproque est utilisée de nombreuses fois dans la "vie" d'un rayon, notamment lors du parcours des arbres et des calculs d'intersections rayon/triangle.

```
float d[2][2] = {INF, -INF, INF, -INF};
const int i, m3[5] = {0,1,2,0,1};
const int kw = major_axis(r[0]->d), k[2] = {m3[kw+1], kv[kw+2]}
const float s = r[0]->d[kw] < 0.0f ? -1.0f : 1.0f;
//recherche des pentes min et max
for (i = 0; i < nr; i++) {
  const Ray *r = rays+i;
  for (a = 0; a < 2; a++) {
    const float p = r->d[k[a]] * r->id[kw];
    d[a][MIN] = min(d[a][0], p);
    d[a][MAX] = max(d[a][1], p);
  }
}
//initialisation des normales par orthogonalité
for (j = 0; j < 4; j++) n[j][3] = -INF;
n[0][kw] = s*d[0][0]; n[0][ku] = -s; n[0][kv] = 0;
n[1][kw] = -s*d[0][1]; n[1][ku] = s; n[1][kv] = 0;
n[2][kw] = s*d[1][0]; n[2][ku] = 0; n[2][kv] = -s;
n[3][kw] = -s*d[1][1]; n[3][ku] = 0; n[3][kv] = s;
//distances max aux plans
for (i = 0; i < nr; i++) {
  const Ray *r = rays+i;
  for (j = 0; j < 4; j++) {
    const int ka = k[j >> 1];
    const float o = r->o[kw] * n[j][kw] + r->o[ka] * n[j][ka];
    n[j][3] = max(n[j][3], o);
  }
}
```

FIG. III.3 – Méthode de création d'un groupe de rayons par détermination des pentes extrêmes. On peut noter que cette fonction peut se réécrire en utilisant les instructions SIMD dans les itérations sur les rayons. Dans un souci de présenter essentiellement l'algorithme, seule la version scalaire est fournie.

1.3 Méthode des rectangles

Une autre méthode pour construire les plans est dérivée du performant algorithme de Reshetov [65]. Cet algorithme est dédié au rejet des triangles situés en dehors du *frustum* (pyramide tronquée) formé par les quatre plans. Des simplifications mathématiques évitent le calcul explicite des équations de plans dans la méthode originale ; un réaménagement de la solution pour dégager ces valeurs demeure toutefois simple. Un axe principal pour l'ensemble des rayons est également utilisé ; mais au lieu de calculer les pentes extrêmes des rayons, des rectangles orthogonaux à l'axe principal sont calculés de manière à encadrer tous les rayons. Pour cela il suffit de calculer l'intersection des rayons avec un plan proche P_n et un plan lointain P_f. Reshetov utilise les limites d'une boîte englobante existante dans son approche. Il est possible d'adapter cette méthode en prenant pour un ensemble de rayons quelconques le minimum des origines des rayons dans l'axe principal pour le plan proche. Pour le plan lointain, on peut réciproquement utiliser la valeur maximale des destinations des rayons si ces points existent, ou, dans le cas de rayons infinis ($t_{max} = \infty$) utiliser plus simplement la limite de la boîte englobante du modèle.

Il est donc nécessaire pour chaque rayon de calculer les points d'intersection p_n et p_f avec les plans P_n et P_f tel que :

$$
\begin{aligned}
t_n &= (P_{nw} - O_w)/D_w \\
p_{nu} &= O_u + t_n.D_u \\
p_{nv} &= O_v + t_n.D_v \\
t_f &= (P_{fw} - O_w)/D_w \\
p_{fu} &= O_u + t_f.D_u \\
p_{fv} &= O_v + t_f.D_v
\end{aligned}
$$

Pour chaque plan, les coordonnées des rectangles $Rect_n$ et $Rect_f$ sont alors données par :

$$
\begin{aligned}
Rect_n[s_{n1}, s_{n2}, s_{n3}, s_{n4}] = [\quad & (min(p_{nu}), min(p_{nv}), t_n) \ , \quad (max(p_{nu}), min(p_{nv}), t_n) \ , \\
& (min(p_{nu}), max(p_{nv}), t_n) \ , \quad (max(p_{nu}), max(p_{nv}), t_n) \ ]
\end{aligned}
$$

$$
\begin{aligned}
Rect_f[s_{f1}, s_{f2}, s_{f3}, s_{f4}] = [\quad & (min(p_{fu}), min(p_{fv}), t_f) \ , \quad (max(p_{fu}), min(p_{fv}), t_f) \ , \\
& (min(p_{fu}), max(p_{fv}), t_f) \ , \quad (max(p_{fu}), max(p_{fv}), t_f) \ ]
\end{aligned}
$$

Les normales au plan peuvent être calculées à partir de ces rectangles. En exploitant les simplifications liées aux rectangles alignés, Reshetov montre par exemple que le produit vectoriel $\widehat{s_{n1}s_{n2}} \times \widehat{s_{n1}s_{f1}}$ peut être réalisé en 2 multiplications et 4 soustractions, au lieu de 6 multiplications et 9 soustractions pour un calcul complet. Pour compléter l'équation des plans, les calculs de distances sont obtenus en intégrant les sommets appropriés des rectangles. Reshetov [65] propose un version SIMD très optimisée de cette méthode.

D'un point vue qualitatif, cette méthode permet d'obtenir des plans plus resserrés que la méthode des pentes lorsque des rayons se croisent. Dans ce cas, comme le montre la figure III.2, l'utilisation des rectangles limite le frustum à l'espace réellement utilisé et non selon une projection infinie des rayons, comme le fait la méthode des pentes extrêmes.

1.4 Des plans aux rayons

L'utilisation de quatre rayons encadrants représente un choix de représentation de l'enveloppe convexe d'un groupe de rayons. Pour les construire, il est possible de passer des équations de plans aux équations de rayons. Cependant, avec la méthode des rectangles, le calcul des normales peut être évité puisque les origines des rayons sont directement issus des sommets du rectangle proche $Rect_n$, et que les directions des rayons sont obtenues en soustrayant les coordonnées du rectangle lointain $Rect_f$ aux coordonnées du rectangle proche $Rect_n$.

Avec la méthode des pentes extrêmes, il est nécessaire de calculer des produits vectoriels par couple de plans pour déduire la direction de chaque rayon. Comme les

normales aux plans ont obligatoirement une composante à 0 et une autre composante à 1 (axes k_u et k_v), ce calcul est réalisé avec seulement une multiplication pour chaque direction. Pour déterminer les origines des rayons, de la même manière que Reshetov, Boulos *et al.* utilisent les coordonnées d'un rectangle formé par l'intersection des quatre plans encadrants avec le plan proche (valeur minimale des origines dans l'axe principal) ; ces calculs requièrent cependant plus d'opérations. En introduisant les points du plan proche situé à une distance t_n et défini par $P_n = (0, 0, 1)$ dans les équations de plans NU_{min}, NU_{max}, NV_{min}, NV_{max}, on obtient les coordonnées 2D suivantes :

$$\begin{aligned} left &= -NU_{min}[3] + (NU_{min}[k_w] \times t_n) \\ right &= NU_{max}[3] - (NU_{max}[k_w] \times t_n) \\ bot &= -NV_{min}[3] + (NV_{min}[k_w] \times t_n) \\ top &= NV_{max}[3] - (NV_{max}[k_w] \times t_n) \end{aligned}$$

Ces coordonnées permettent l'initialisation directe des origines des rayons encadrants :

$$\begin{aligned} O_1 &= (left, bot, t_n) \\ O_2 &= (right, bot, t_n) \\ O_3 &= (left, top, t_n) \\ O_4 &= (right, top, t_n) \end{aligned}$$

1.5 Discussion

Dans certaines conditions l'agencement des rayons est cohérent par construction. Les rayons primaires par exemple partagent la même origine et passent par les pixels d'une image. La formation des pixels suit un quadrillage régulier qui implique que tous les rayons situés dans un rectangle de cette image sont inclus dans l'enveloppe convexe formée par les quatre rayons passant par les sommets de ce rectangle. Dans ces conditions, il suffit simplement d'initialiser les rayons encadrants par copie des paramètres des rayons appropriés pour éviter un coût de prétraitement additionnel.

Les méthodes proposées traitent de la construction de groupes de rayons quelconques ; il va de soit que certaines optimisations peuvent être appliquées quand les rayons partagent la même origine ou la même direction.

Pour comparer les deux méthodes présentées, les nombres d'additions, de multiplications et de tests de comparaisons ont été comptabilisés. Le tableau III.1 sur la page suivante présente en détail le compte selon les étapes de calcul pour chaque méthode. Si l'on considère que toutes les opérations ont le même coût, alors la méthode des rectangles prend l'avantage dès que le groupe dépasse deux rayons et que les normales ne sont pas utilisées.

Opérations	Pentes extrêmes			Rectangles		
	$\oplus$	$\otimes$	$<$	$\oplus$	$\otimes$	$<$
Pentes	-	$2N$	$4N$	-	-	-
Rectangles	-	-	-	$6N$	$6N$	$8N$
Initialisation des normales	-	4	-	16	8	-
Distances au plan	$4N$	$8N$	$4N$	4	8	-
Directions des rayons	-	4	-	12	-	-
Origines des rayons	4	4	-	-	-	-
Totaux (avec plan)	$4+4N$	$12+10N$	$8N$	$32+6N$	$16+6N$	$8N$
Totaux (sans plan)	-	-	-	$12+6N$	$6N$	$8N$

Tab. III.1 – Comparaison des méthodes de constructions des groupes de rayons par nombre d'opérations. N représente le nombre de rayons dans le groupe. $\oplus$, $\otimes$ et $<$ représentent respectivement des opérations d'addition, de multiplication et de comparaison. La méthode des rectangles ne requiert pas les étapes d'initialisation des normales et de calculs de distances aux plans pour construire les rayons encadrants. Dans ce cas précis, cette méthode est plus avantageuse.

2 Intersection d'un groupe de rayons avec un triangle

2.1 Problématique

Les groupes de rayons sont implémentés avec le kd-Tree et l'arbre BVH. Les natures opposées de ces structures de recherche impliquent que les algorithmes de traversée diffèrent ; les stratégies sont évoquées dans la section 3 à la page 62 et la section 4 à la page 76. L'algorithme d'intersection groupe/triangle quant à lui est commun aux deux approches.

D'une manière générale, un groupe de rayons est composé d'un faisceau défini par quatre rayons encadrants (un rayon SIMD) et d'un ensemble de rayons SIMD. Lorsque le faisceau rencontre une feuille lors du parcours de l'arbre, alors des calculs d'intersection sont réalisés pour l'ensemble du groupe. En considérant le faisceau, trois cas peuvent se produire : dans le premier cas, le faisceau n'intersecte pas le triangle et on peut conclure la même chose pour tous les rayons du groupe ; dans le second cas, le faisceau intersecte complètement le triangle et on est certain que tous les rayons du groupe l'intersectent également ; dans le dernier cas, l'intersection du faisceau avec le triangle est partielle et aucune information ne peut être exploitée.

Seul le cas de non intersection du faisceau avec un triangle est traité comme technique d'accélération. Ce cas est d'une part beaucoup plus probable que le cas d'intersection complète, et d'autre part évite effectivement tout calcul pour l'ensemble des rayons. En effet, lors du calcul d'intersection rayon/triangle, il est nécessaire de récupérer des informations comme la distance au point d'intersection et les coordon-

nées barycentriques pour chaque rayon. Le cas d'intersection complète du faisceau avec un triangle ne dispense pas de devoir effectuer ces calculs pour chaque rayon du groupe.

Les expérimentations d'intersection groupes/triangles concernent alors essentiellement l'exclusion d'un triangle. Pour détecter cette exclusion, deux techniques sont envisagées selon que l'on se place du point de vue du faisceau ou de celui du triangle. Du point de vue du faisceau, une technique de rejet par plan est possible [65]. Du point de vue du triangle, une technique de rejet par côté est possible [21]. Ces deux techniques sont présentées dans les sections suivantes.

2.2 Rejet par les plans du faisceau

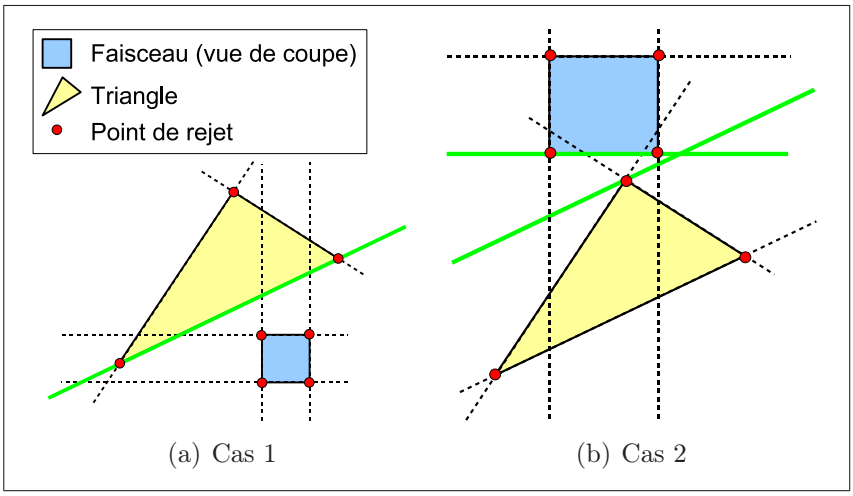


FIG. III.4 – Méthodes de rejet rapide d'un triangle par rapport à un faisceau. Le premier cas (a) montre le rejet du faisceau par un côté du triangle ($\alpha < 0$), tandis que le rejet par plans échoue. Le deuxième cas (b) montre le rejet du triangle par le plan inférieur du faisceau, tandis que le rejet par côté du triangle échoue. Cependant, en étendant l'utilisation des coordonnées barycentriques, il devient possible de rejeter le faisceau ($\alpha > 1$).

La figure III.4 illustre le rejet par plan. Cette technique exclue un triangle si ses trois sommets se trouvent positionnés du côté extérieur d'un des quatre plans du faisceau. Si l'on considère l'équation du plan tel que $N.p + K = 0$, alors le triangle constitué des sommets s_a, s_b et s_c est exclu par ce plan si $\forall s \in \{s_a, s_b, s_c\}$ alors $N.s > -K$. Reshetov [65] présente une version optimisée SIMD de ces tests en tenant compte de la nature alignée des plans.

2.3 Rejet par les côtés du triangle

Une autre façon de calculer l'exclusion d'un triangle pour un groupe de rayons repose sur le rejet des rayons encadrants par les côtés du triangle. Autrement dit,

si les quatre rayons encadrants passent à l'extérieur d'un des côtés du triangle alors le faisceau n'intersecte pas le triangle. Bien que non réalisé, ce test peut être implémenté efficacement en exploitant les coordonnées de Plucker [70]. Dans ce système particulier de coordonnées, il existe un opérateur propice à ce test qui permet de situer une droite par rapport à une autre. Une autre façon de réaliser ce test est d'utiliser les coordonnées barycentriques des points d'intersection des rayons encadrants avec le triangle. Pour parvenir aux mêmes résultats qu'avec les coordonnées de Plucker, il suffit alors de vérifier que les quatre points d'intersection aient leurs coordonnées barycentriques α ou β ou $\gamma < 0$.

En terme d'implémentation, les étapes de calculs sont les mêmes que celle d'un calcul d'intersection rayon SIMD/triangle (voir la section 3.2.2 à la page 49) jusqu'aux calculs des coordonnées barycentriques. En terme d'efficacité, la nécessité de calculer les intersections avec le plan du triangle sont des étapes supplémentaires par rapport à une approche utilisant les coordonnées de Plucker. Cependant, il est possible d'étendre le test des coordonnées barycentriques en excluant également les cas pour lesquels α ou β ou γ sont > 1 et obtenir ainsi un meilleur taux d'exclusion. La figure III.4 montre un cas où, pour la même intersection non valide, le test par coordonnées barycentriques réussit à rejeter le triangle tandis que le test par demi-espace (coordonnées de Plucker) échoue.

2.4 Masquage des triangles

Les expérimentations présentées dans les sections suivantes (voir la section 3 sur la page suivante) montrent que la traversée du faisceau peut être décorrélée de la traversée du groupe de rayons. Pour bénéficier des triangles rejetés par un faisceau, il est alors nécessaire de mémoriser les triangles exclus pour un usage différé par le groupe de rayons. En remarquant que le nombre de triangles dans une feuille est très souvent $\leq$ à 32, un masque codé sur un entier de 32 bits est utilisé pour marquer les triangles exclus. Pour supporter les rares cas où le nombre de triangles serait trop grand, ceux situés après les 32 premiers ne peuvent pas être exclus. L'usage de ce masque offre l'avantage d'être insignifiant en coût mémoire et en temps de calculs.

2.5 Discussion

La figure III.4 présente les différents types d'exclusion. On peut remarquer que selon la technique utilisée, certains cas d'exclusion ne sont pas identifiés. Ces différences s'expliquent essentiellement pas le nombre de demi-espaces exploités par les différentes techniques :

- rejet par plans : 4 demi-espaces ;
- rejet par côtés avec utilisation des coordonnées de Plucker : 3 demi-espaces ;
- rejet par côtés avec utilisation des coordonnées barycentriques : 6 demi-espaces.

Il semble normal que le taux d'exclusion soit lié au nombre de demi-espaces permettant de rejeter le faisceau ou le triangle. La technique de rejet par côtés avec utilisation des coordonnées barycentriques (shaft culling) est exclusivement employée dans

les expérimentations qui suivent. Les tableaux III.4 et III.11 montrent les bénéfices tirés de l'utilisation du masquage de triangles.

3 Groupes de rayons avec le kd-Tree

3.1 Problématique

Les approches de regroupement de rayons diffèrent selon la structure de recherche utilisée. Pour l'arbre BVH, le faisceau représenté par les rayons encadrants est propagé en même temps que les rayons du groupe. Pour le kd-Tree, le traitement est réalisé en deux temps. Dans un premier temps le faisceau seul est propagé afin de récupérer le sous-ensemble visité de l'arbre. La définition même du sous-ensemble dépend de la méthode d'exploration utilisée. Ce sous-ensemble est ensuite exploité indépendamment par chaque rayon du groupe. Cette différence tient essentiellement du fait que le kd-Tree est plus sensible que l'arbre BVH à l'utilisation des groupes de rayons, et que selon les résultats d'exploration du faisceau, il est possible d'envisager des requêtes d'intersection classiques ou groupées. Ces points sont évoqués plus en détail dans le chapitre V à la page 117 concernant les rayons divergents.

La problématique principale est de déterminer, avec le meilleur rapport temps/-qualité, un dénominateur commun : un sous-ensemble de noeuds dans l'arbre pour lequel tous les rayons partagent l'ordre de traversée. Si la méthode d'exploration minimise ce dénominateur commun, la faible factorisation des étapes de traversée et des calculs d'intersection apporte un gain de performance négligeable. A l'inverse, si la méthode d'exploration tend à chercher le dénominateur commun optimal, qui peut ne pas exister, alors la pénalité occasionnée par une recherche longue et infructueuse peut dégrader les performances.

L'approche retenue pour traiter les groupes de rayons avec le kd-Tree dissocie donc le parcours du faisceau (enveloppe du groupe de rayons) du parcours des rayons. Trois expérimentations ont été réalisées pour le parcours du faisceau. La première reprend les travaux de Reshetov *et al.* [66] sur la recherche d'un point d'entrée commun, que ce soit un noeud ou une feuille. La seconde s'apparente aux travaux de Dmitriev *et al.* [21] qui récupère la liste de toutes les feuilles communes visitées par un faisceau, si et seulement si cette liste est commune à tous les rayons. La dernière expérimentation récupère un sous-ensemble de l'arbre avec les noeuds pour lesquels les rayons sont divergents et les feuilles communes. Ces trois expérimentations sont présentées en suivant après avoir montré comment propager d'une manière générale un faisceau dans un kd-Tree.

3.2 Intersection d'un faisceau de quatre rayons avec le plan séparateur

De par leur construction, tous les rayons d'un groupe y compris les rayons encadrants, partagent le même signe dans la direction principale du faisceau. Pour

simplifier les implémentations, la même hypothèse est faite également pour les deux autres directions. Cette contrainte n'est pas vraiment pénalisante car cela revient à dissocier huit grands ensembles de rayons à partir desquels sont construits les groupes. De plus, la nature alignée des plans séparateurs dans le kd-Tree (ou les boîtes pour l'arbre BVH) entraîne plus probablement la subdivision d'un groupe dont les signes de direction diffèrent. Dans la quête du plus grand dénominateur commun, il est préférable d'effectuer la séparation en huit octants préalablement.

Afin de propager le faisceau dans le kd-Tree, il est nécessaire de disposer d'opérateurs conservatifs qui permettent de discerner les trois cas suivants lors de chaque passage dans un noeud :

- *proche* : le faisceau traverse exclusivement le fils proche ;
- *lointain* : le faisceau traverse exclusivement le fils lointain ;
- *commun* : le faisceau traverse complètement les deux fils ;

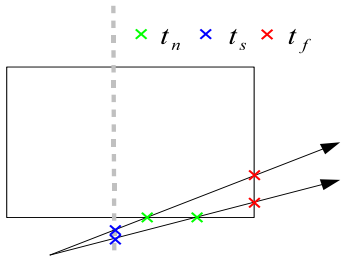
Si aucun de ces trois cas n'est détecté, alors le cas spécial de *division* du faisceau est déduit.

Pour parvenir à dissocier ces trois cas, deux approches peuvent être utilisées. La première approche est une extension des rayons SIMD et reprend la notion de segments de rayons tandis que la deuxième approche utilise un algorithme d'intervalle arithmétique [66, 11].

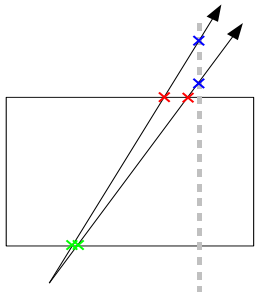
3.2.1 Segments de rayons

La figure III.5 illustre les trois cas de parcours cohérents et le cas de division suivant l'utilisation de segments de rayons. Selon les distances d'entrée tn et de sortie tf des rayons, et la distance d'intersection au plan séparateur ts, on peut affirmer que si pour tous les rayons :

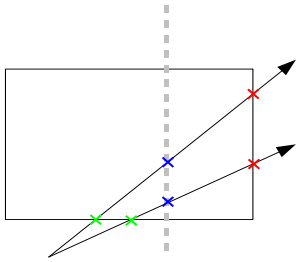
- $ts_i > tf_i$ alors on se situe dans le cas *proche* ;
- $ts_i < tn_i$ alors on se situe dans le cas *lointain* ;
- les deux tests précédents sont faux, alors on se situe dans le cas *commun*.



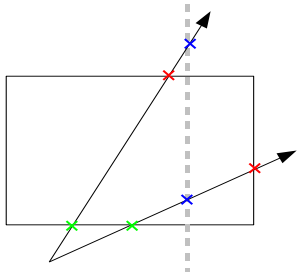
(a) $\forall i\ ts_i < tn_i$: cas *lointain*



(b) $\forall i\ ts_i > tf_i$: cas *proche*



(c) $\forall i\ tn_i \leq ts_i\ et\ ts_i \leq tf_i$: cas *commun*



(d) Sinon : cas *division*

FIG. III.5 – Parcours d'un faisceau par segment dans un kd-Tree selon les différents cas de traversée (a) *lointain*, (b) *proche*, (c) *commun* et (d) *division*.

La figure III.6 montre l'implémentation de ce test dans le contexte du parcours d'un faisceau. Les vecteurs SIMD $left4$ et $right4$ contiennent les valeurs inverses des deux premiers tests. Le troisième test vérifie une égalité entre $left4$ et $right4$ qui ne peut être vrai que si $left4$ et $right4$ sont vrais.

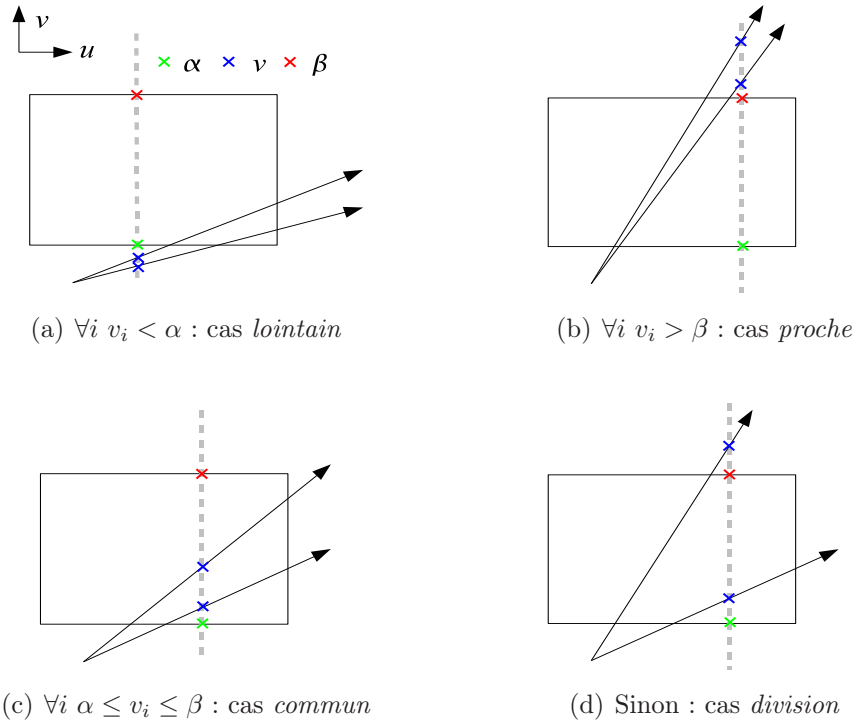
```
const int a = split_axis(n);
const __m128 s4 = _set1_ps(n->split_location);
//distance au plan séparateur
const __m128 ts4 = _mul_ps(_sub_ps(s4, r4->o4[a]), r4->id4[a]);
const __m128 left4 = _cmpnlt_ps(ts4, tn4);
const __m128 right4 = _cmpngt_ps(ts4, tf4);
if (!_movemask_ps(left4)) ... //lointain
else if (!_movemask_ps(right4)) ... //proche
else if (!_movemask_ps(_xor_ps(left4, right4)) ... //commun
else ... //division
```

FIG. III.6 – Détection des différents cas de traversée dans un kd-Tree pour un faisceau en utilisant les rayons encadrants.

3.2.2 Utilisation de l'intervalle arithmétique

L'implémentation de la propagation du faisceau par intervalle arithmétique requiert plus de calculs mais ne nécessite pas de maintenir à jour les distances tn et

tf. La figure III.7 illustre les différents cas de propagation avec cet algorithme. Une interprétation géométrique de ce calcul est de vérifier comment les rayons encadrants passent par le rectangle représenté par l'intersection de la boîte et du plan séparateur. Si tous les rayons passent à l'intérieur de ce rectangle alors le cas *commun* est trouvé. Si tous les rayons passent par un des quatre côtés du rectangle alors les cas *proche* ou *lointain* sont trouvés, en fonction des signes de direction des rayons.



(a) $\forall i\ v_i < \alpha$: cas *lointain*

(b) $\forall i\ v_i > \beta$: cas *proche*

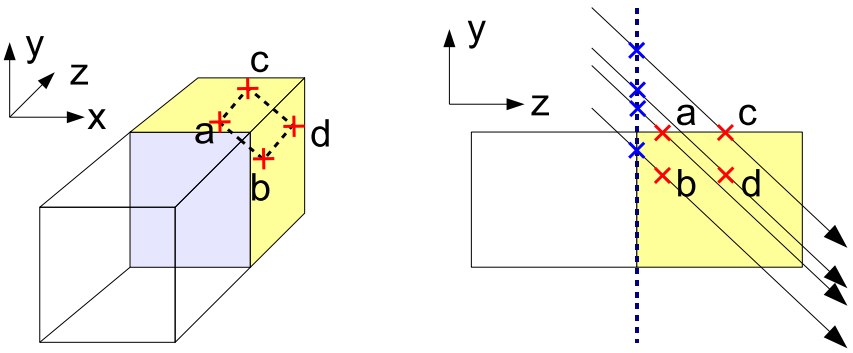
(c) $\forall i\ \alpha \leq v_i \leq \beta$: cas *commun*

(d) Sinon : cas *division*

FIG. III.7 – Parcours d'un faisceau par intervalle arithmétique dans un kd-Tree selon les différents cas de traversée (a) *lointain*, (b) *proche*, (c) *commun* et (d) *division*.

La figure III.9 montre une implémentation concrète de cet algorithme en fonction du placement des points d'intersection par rapport au rectangle et des signes de direction. On peut remarquer que cette méthode demande plus de calculs, car réalisée en deux dimensions, et nécessite de plus l'accès aux limites de la boîte. Benthin [6] montre comment ces limites peuvent être calculées incrémentalement lors du parcours du faisceau.

Si la propagation d'un faisceau est continuée après un cas de *division* alors seul le calcul par intervalle arithmétique peut fournir des résultats corrects. La figure III.8 illustre un cas où l'algorithme par segment détecte un cas *lointain* alors que le faisceau présente réellement un cas de *division*. Cependant, pour arriver dans ce cas où les points d'entrée des rayons encadrants sont situés sur des plans distincts, le faisceau doit auparavant être passé par un cas de *division* bien détecté. On utilisera alors l'un ou l'autre algorithme selon que l'on souhaite continuer la propagation d'un faisceau après un cas de *division* ou non.



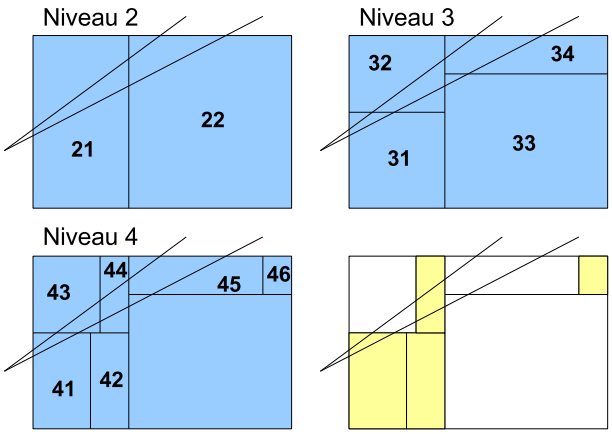
(a) Vue depuis l'origine du faisceau. (b) Vue de coupe de la boîte.

FIG. III.8 – Mauvaise interprétation possible du parcours du faisceau si utilisation des segments après un cas de *division*. Bien que le cas *lointain* soit détecté (b), il reste possible de placer des rayons entre a et b (a) qui ne suivent pas le même chemin que le faisceau.

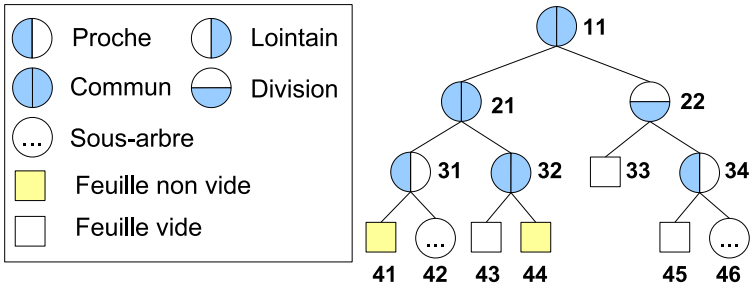
```
const int a = split_axis(n);
const __m128 s4 = _set1_ps(n->split_location);
//distance au plan separateur
const __m128 ts4 = _mul_ps(_sub_ps(s4, r4->o4[a]), r4->id4[a]);
//intersection 2D : pu, pv
const int ku = m3[a+1], kv = m3[a+2];
const __m128 pu4 = _add_ps(r4->o4[ku], _mul_ps(ts4, r4->d4[ku]));
const __m128 pv4 = _add_ps(r4->o4[kv], _mul_ps(ts4, r4->d4[kv]));
int in_left, in_right, in_bot, in_top;
//differents cas de traversee
in_left = _movemask_ps(_cmpge_ps(pu, _set1_ps(bb[0][ku])));
if (!in_left) ... //s[ku] ? proche : lointain
in_right = _movemask_ps(_cmple_ps(pu, _set1_ps(bb[1][ku])));
if (!in_right) ... //s[ku] ? lointain : proche
in_bot = _movemask_ps(_cmpge_ps(pv, _set1_ps(bb[0][kv])));
if (!in_bot) ... //s[kv] ? proche : lointain
in_top = _movemask_ps(_cmple_ps(pv, _set1_ps(bb[1][kv])));
if (!in_top) ... //s[kv] ? lointain : proche
//in_left = in_right = in_bot = in_top = 0xF
if ((in_left & in_right & in_bot & in_top) == 0xF) ... //commun
else ... //division
```

FIG. III.9 – Détection des différents cas de traversée dans un kd-Tree pour un faisceau en utilisant les intervalles arithmétiques.

3.3 Recherche d'un point d'entrée commun



(a) Vue schématique.



(b) Vue arborescente.

FIG. III.10 – Parcours d'un faisceau dans une arborescence. (a) montre une vue schématique décomposée selon chaque niveau de l'arbre. (b) montre les différents cas de traversée du faisceau pour chaque noeud de l'arbre.

Le plus petit dénominateur commun pour le parcours d'un faisceau consiste à trouver un seul point d'entrée commun à tous les rayons du groupe. Afin de réduire efficacement le nombre d'étapes de traversée, ce point d'entrée doit être situé le plus profondément possible dans l'arborescence. Il peut s'agir alors d'une feuille, si l'on détermine que le faisceau passera uniquement par celle-ci, ou d'un noeud pour lequel le faisceau bifurque. Il faudra dans ce dernier cas lever les ambiguïtés de traversées pour chaque rayon. Il est également possible qu'aucun point d'entrée ne soit trouvé évitant ainsi toutes requêtes d'intersections ultérieures.

Pour trouver cet unique point d'entrée Reshetov *et al.* [66] proposent de propager le faisceau dans le kd-Tree à la manière d'un rayon SIMD, en profondeur d'abord puis en largeur, et de ranger dans une pile, au passage, tous les noeuds pour lesquels le faisceau bifurque. Par rapport aux cas de traversée présentés précédemment, une bifurcation cumule le cas *commun* et le cas de *division*. La première feuille rencontrée devient un point d'entrée potentiel. A partir de ce moment, la pile des bifurcations est gelée car aucune autre bifurcation ne pourra être trouvée dans la hiérarchie. Il

est alors nécessaire de continuer la traversée du faisceau en dépilant les bifurcations. Si lors de cette seconde étape une nouvelle feuille est trouvée, alors la dernière bifurcation dépilée devient le nouveau point d'entrée. La figure III.11 montre une version simplifiée de cet algorithme.

En considérant une application de cet algorithme avec l'arbre de la figure III.11(b), les noeuds 11 et 21 (*commun*) sont empilés comme des bifurcations. Le noeud 31 (*proche*) conduit directement à la première feuille visitée 41 qui devient un point d'entrée potentiel. Le sous-arbre 42 est alors ignoré. La dernière bifurcation 21 est dépilée et le parcours continue dans le noeud 32 (*commun*). Le fils proche qui est une feuille vide est ignoré et le fils lointain qui est une feuille non vide valide le noeud 21 en tant que nouveau point d'entrée. La première bifurcation 11 est alors dépilée et mène au noeud 22 (*division*). Le fils proche 33 est ignoré (feuille vide) et le parcours continue sur le noeud 34 (*proche*) qui mène uniquement à une feuille vide 45. Il n'y a plus alors aucun noeud sur la pile, le noeud 21 demeure le point d'entrée.

```
Node *n = tree->root, *bn, *sn[KDTREE_MAX_DEPTH];
int sn_size = 0, sb_size = 0;
while (1) {
  //traversée des noeuds
  while (!is_leaf(n)) {
    const int c = traversal_case(n, r4);
    if (c == FAR) n = right_child(n);
    else if (c == NEAR) n = left_child(n);
    else { //c == COMMON ou c == DIVISION
      sn[sn_size++] = n; //empilement du noeud parent
      if (list_size(nlist) == 0) sb_size++; //empilement d'une bifurcation
      n = left_child(n);
    }
  }
  if (!is_empty(n)) { //feuille non vide
    const int mask = shaft_culling(n, r4); //masque les triangles exclus
    if (mask) {
      if (list_size(nlist) == 0) list_set(nlist, n, mask); //1er point d'entrée
      else list_set(nlist, bn, NO_MASK); //bifurcation devient point d'entrée
      if (sb_size == 0) break;
      sn_size = sb_size; //pile ramenée à la dernière bifurcation
    }
  }
  if (sn_size == 0) break;
  if (sn_size == sb_size) bn = sn[--sb_size]; //dépilement bifurcation
  node = right_child(sn[--sn_size]); //dépilement fils lointain
}
return list_size(nlist);
```

FIG. III.11 – Fonction de recherche d'un unique point d'entrée.

Pour être générique avec les approches présentées dans les sections suivantes, le point d'entrée est mémorisé dans une liste *nlist*. Chaque élément de cette liste comporte une référence au noeud trouvé et un masque des triangles exclus pour le faisceau (voir la section 2.4 à la page 61). Les fonctions de manipulations de cette liste *list_set*(), *list_add*() et *list_size*() permettent respectivement de définir le premier élément de la liste (et de le créer s'il n'existe pas), d'ajouter un nouvel élément et de récupérer le nombre d'éléments de la liste. La fonction *traversal_case*() retourne

les différents cas de traversée (voir la section 3.2 à la page 62) selon la position du faisceau par rapport au plan séparateur.

On peut remarquer que la pile *sn*, liée à un parcours en profondeur d'abord et largeur après, est partagée par les noeuds restant à visiter et les bifurcations ; en effet, les premiers éléments de cette pile sont les mêmes à la différence près que les bifurcations sont les noeuds parents des noeuds (lointains) restants à visiter. Comme il est possible de déduire les fils à partir d'un noeud parent et non l'inverse, seules les bifurcations sont empilées. Deux pointeurs de pile *sn_size* et *sb_size* sont utilisés et incrémentés à chaque empilement ; cependant, le pointeur de pile des bifurcations *sb_size* n'est plus incrémenté dès que la première feuille a été trouvée.

3.4 Recherche des feuilles

A l'opposé de la recherche d'un unique point d'entrée, Dmitriev *et al.* [21] proposent de recueillir l'ensemble des feuilles visitées par le faisceau à condition que tout le trajet du faisceau soit cohérent. Cette dernière propriété est assurée si aucun cas de *division* n'est détecté durant la traversée. Le résultat de la fonction de traversée renvoie donc soit une liste de N feuilles pour $N \geq 0$, ou soit un code d'erreur pour un parcours de faisceau incohérent. Contrairement à l'approche précédente, le temps passé à la recherche des feuilles peut être un temps additionnel sans bénéfice pour le traitement ultérieur des rayons. Une amélioration possible de cette technique, mais non implémentée, est de mémoriser l'ensemble des feuilles communes et la première bifurcation, si elle existe. La figure III.12 montre le code source simplifié de la recherche des feuilles.

```
Node *n = tree->root, *sn[KDTREE_MAX_DEPTH];
int sn_size = 0;
while (1) {
  //Traversée des noeuds
  while (!is_leaf(n)) {
    const int c = traversal_case(n, r4);
    if (c == FAR) n = right_child(n); //lointain
    else if (c == NEAR) n = left_child(n); //proche
    else if (c == COMMON) { //commun
      sn[sn_size++] = right_child(n);
      n = left_child(n);
    }
    else return -1; //sortie rapide si division
  }
  if (!is_empty(n)) {//feuille non vide
    const int mask = shaft_culling(n, r_x4); //masque les triangles exclus
    if (mask) list_add(nlist, n, mask); //nouvelle feuille
  }
  if (sn_size == 0) break;
  node = sn[--sn_size]; //noeud suivant...
}
return list_size(nlist);
```

FIG. III.12 – Fonction de recherche de toutes les feuilles

Dans la figure III.11(b), cet algorithme parcourt l'arbre en suivant les noeuds 11 (*commun*), 22 (*commun*) et 31 (*proche*) pour arriver à la première feuille 41 ajoutée

dans la liste. Le prochain noeud visité 32 (*commun*) mène d'abord à une feuille vide 43 ignorée et une feuille non vide 44 ajoutée dans la liste. Ensuite le noeud 22 (*division*) amène l'algorithme à se terminer promptement car le parcours devient incohérent. Cet exemple présente donc un cas où la recherche des feuilles échoue.

3.5 Suppression du sous-arbre commun

Une approche intermédiaire est proposée entre la recherche unique d'un point d'entrée et la recherche exhaustive des feuilles. Cette approche supprime le sous-arbre commun à l'ensemble du faisceau en collectant l'ensemble des feuilles non vides et des noeuds incohérents (cas de *division*). Seuls les calculs d'intersection avec les feuilles non vides et le parcours des noeuds incohérents resteront à charge pour les rayons du groupe. La figure III.13 présente cet algorithme. En comparant cette méthode avec la recherche exhaustive de feuilles, les étapes de traversée sont identiques jusqu'à la traversée du noeud 22 *division* (voir la figure III.11(b)) qui dans ce cas est ajouté à liste des noeuds.

```
Node *n = tree->root, *sn[KDTREE_MAX_DEPTH];
int sn_size = 0;
while (1) {
  //Traversée des noeuds
  while (!is_leaf(n)) {
    const int c = traversal_case(n, r4);
    if (c == FAR) n = right_child(n); //lointain
    else if (c == NEAR) n = left_child(n); //proche
    else if (c == COMMON) { //commun
      sn[sn_size++] = right_child(n);
      n = left_child(n);
    }
    else break; //arrêt de la traversée si division
  }
  if (!is_leaf(n)) list_add(nlist, n); //nouveau noeud
  else if (!is_empty(n)) { //feuille non vide
    const int mask = shaft_culling(n, r_x4); //masque les triangles exclus
    if (mask) list_add(nlist, n, mask); //nouvelle feuille
  }
  if (sn_size == 0) break;
  node = sn[--sn_size]; //noeud suivant...
}
return list_size(nlist);
```

FIG. III.13 – Fonction de recherche des noeuds *incohérents* et des feuilles non vides.

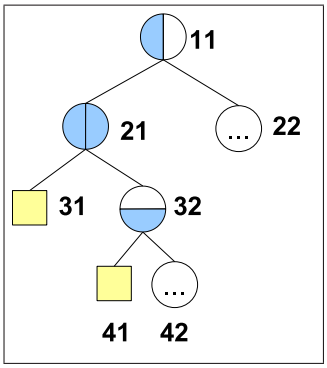


FIG. III.14 – Arborescence de traversée.

Contrairement à la recherche d'un point d'entrée unique, les noeuds de type *Commun* ne font pas partie de l'ensemble des noeuds qui peuvent être recensés. Comme leur traversée est cohérente pour tous les rayons, il est inutile de les traiter à nouveau pour chaque rayon du groupe. La figure III.14 montre un exemple dans lequel la recherche d'un point d'entrée unique conduit à trouver le noeud 21 (*commun*) tandis que la méthode de suppression du sous-arbre récupère directement la feuille 31 et le noeud 32 (*division*). On peut noter ici que la méthode de recherche des feuilles échoue à nouveau étant donné le cas de *division*.

3.6 Résultats

Afin de comparer les différentes méthodes de traversée du groupe de rayons, un ensemble d'expérimentations a été réalisé sur une dizaine de modèles comportant des spécificités différentes (voir la section A à la page 190). Toutes les mesures ont été recueillies lors du rendu de 1024^2 rayons primaires pour la vue par défaut des modèles. Le tableau III.2 montrent les performances brutes obtenues en terme de millions de rayons par seconde. Différentes tailles de groupes de rayons ont été testées pour chaque stratégie de traversée. Pour l'ensemble des stratégies, les tailles extrêmes de groupes de rayons (16 ou 1024 rayons) donnent les moins bonnes performances. Le groupe de 64 rayons offre le meilleur compromis.

Si l'on compare les stratégies de traversée, la recherche des feuilles est la moins performante tandis que la recherche du sous-arbre donne toujours les meilleurs résultats. La recherche des feuilles souffre essentiellement de la pénalité occasionnée par une recherche infructueuse. Par rapport à la recherche des points d'entrée, la recherche du sous-arbre est plus efficace en moyenne d'un facteur x1,2. Cette différence s'explique notamment par la capacité de recherche du sous-arbre à aller plus loin dans l'exploration de l'arbre et ainsi de réduire efficacement le nombre d'étapes de traversées pour les rayons (voir le tableau III.5).

	Mr/s											
Méthode	Point d'entrée				Feuilles				Sous-arbre			
# rayons	16	64	256	1024	16	64	256	1024	16	64	256	1024
Armadillo	3,8	4,5	4,7	4,5	4,2	4,3	4,3	4,1	4,5	**4,8**	4,6	4,3
Buddha	3,6	4,2	4,3	4,2	4,1	4,2	4,2	4,1	4,2	**4,5**	4,4	4,2
Bunny	4,6	5,5	5,7	5,5	5,1	5,5	5,3	5,1	5,7	**6,2**	5,9	5,4
Clipper	2,8	3,1	3,1	2,9	2,7	2,8	2,8	2,6	3,1	**3,3**	3,2	2,9
Conference	4,1	4,7	4,7	4,3	4,2	4,4	4,2	3,9	4,9	**5,9**	5,0	4,9
ERW6	8,7	10,5	10,4	9,1	7,9	8,6	8,0	7,2	10,7	**13,1**	12,1	9,9
Fairy	3,0	3,5	3,5	3,3	2,8	3,0	2,9	2,8	3,5	**3,9**	3,7	3,4
Soda	3,2	4,0	4,4	4,3	3,6	4,1	4,2	4,1	3,2	5,4	**5,6**	5,0
Sully	4,2	4,9	4,9	4,6	4,0	4,2	4,1	3,9	4,7	**5,5**	5,1	4,6
Villa Digy	3,7	4,5	4,6	4,4	3,9	4,2	4,1	4,0	5,1	**6,5**	6,1	5,3
Moyenne	4,2	4,9	5,0	4,7	4,2	4,5	4,4	4,2	5,0	**5,9**	5,6	5,0

TAB. III.2 – Millions de requêtes d'intersection exécutées par seconde pour un rendu de 1024x1024 rayons primaires. Les résultats sont dissociés pour les différentes stratégies de recherche d'un dénominateur commun, et pour différentes tailles de groupes de rayons. Les meilleures performances sont obtenues avec la recherche d'un sous-arbre pour des groupes composés de 64 rayons.

Le tableau III.3 compare les efficacités des rayons SIMD et des groupes de 64 rayons par rapport aux rayons simples. Les rayons SIMD offrent un facteur d'accélération assez stable autour de la moyenne de x2,5. Les performances des groupes de 64 rayons sont plus mitigées, car avec une accélération moyenne de x3,6 les performances varient entre x1,9 (Buddha) et x6 (ERW6). Le faible gain de performance obtenu avec le modèle *Buddha* s'explique par la spécificité de ce modèle constitué de nombreux petits triangles. Pour ce modèle, un groupe de rayons diverge tôt dans la hiérarchie de traversée pour à terme avoir très peu de rayons en commun qui intersectent un même triangle. Cette divergence limite alors les possibilités de factorisation des étapes de traversée et des calculs d'intersection rayon/triangle. Le modèle ERW6 est exactement l'inverse puisque sa géométrie est composée de quelques centaines de grands triangles. En dehors de ces deux cas extrêmes qui ne constituent pas la réalité des modèles utilisés fréquemment, l'accélération apportée par les groupes de 64 rayons se situe entre x3 et x4,5.

# rayons	Mr/s			Accélération	
	1	4	64	4/1	64/1
Armadillo	1,9	4,2	4,8	x2,2	x2,5
Buddha	2,3	4,1	4,5	x1,7	x1,9
Bunny	2,1	5,3	6,2	x2,5	x3,0
Clipper	1,1	2,6	3,3	x2,3	x2,9
Conference	1,3	3,8	5,9	x2,8	x4,4
ERW6	2,2	6,4	13,1	x2,9	x6,0
Fairy	1,1	2,8	3,9	x2,6	x3,6
Soda	1,6	4,2	5,4	x2,7	x3,4
Sully	1,4	3,9	5,5	x2,8	x3,9
Villa Digy	1,4	4,1	6,5	x2,8	x4,5
Moyenne	1,7	4,1	5,9	x2,5	x3,6

TAB. III.3 – Millions de requêtes d'intersection exécutées par seconde pour un rendu de 1024x1024 rayons primaires. Comparaison des performances des groupes de 64 rayons (avec sous-arbre) par rapport à des rayons simples et des rayons SIMD.

L'accélération apportée par un groupe de rayons tient dans la réduction du nombre d'étapes de traversée par la recherche du sous-arbre commun ainsi que la réduction du nombre de calculs d'intersection par le masquage des triangles. Cependant, comme le montre le tableau III.4, l'apport du masquage des triangles est faible et peut même être défavorable. Malgré une réduction effective du nombre de tests d'intersection de plus de 20%, le bénéfice sur le temps de calcul est en dessous de 10% dans le meilleur cas. On peut considérer que le nombre de tests d'intersection est assez stable avec ou sans regroupement de rayons. L'efficacité réelle provient de la réduction du nombre d'étapes de traversées comme le présente le tableau III.5.

Modèles	Sans masquage	Avec masquage			
	I_{ref} : # tests	# tests	Point d'entrée	Feuilles	Sous-arbre
Conference	1 957	I : intersection	1 692	1 495	1 383
		M : masquage	211	56	73
		$(I+M)/I_{ref}$	97%	79%	74%
		accélération	x1,01	x1,03	x1,08
Fairy	3 399	I : intersection	2 273	2 573	2 399
		M : masquage	285	68	100
		$(I+M)/I_{ref}$	75%	78%	74%
		accélération	x1,03	x1,02	x1,05
Soda Hall	1 365	I : intersection	1 358	1 361	1 152
		M : masquage	247	22	83
		$(I+M)/I_{ref}$	118%	101%	91%
		accélération	x0,95	x0,97	x0,99

TAB. III.4 – Nombre de tests d'intersection (en milliers) selon le masquage ou non des triangles. Les tests d'intersection (I) sont réalisés par les rayons et les tests de masquage (M) sont réalisés par les faisceaux. Malgré une réduction effective du nombre global de tests d'intersection, le masquage des triangles apporte relativement peu à la performance.

En comparant les nombre d'étapes de traversée selon différentes tailles de groupes de rayons (voir le tableau III.5), on constate que le plus petit nombre d'étapes, soit la meilleure factorisation, est atteinte pour les groupes de 64 rayons. Ce résultat peut être comparé aux performances brutes obtenues, et confirme que l'accélération obtenue est directement liée à la réduction du nombre d'étapes de traversées. On peut remarquer que les faisceaux et les rayons travaillent de pair. Pour les petits groupes de rayons, le faisceau génère un sous-arbre commun important réduisant ainsi l'arbre restant à explorer pour les rayons. A l'inverse, pour les grands groupes de rayons, la faible exploration du faisceau confronté à une divergence rapide, laisse une grande partie de l'arbre à explorer aux rayons.

Modèles	Type de traversée	# traversées selon le nombre de rayons					
		1	4	16	64	256	1024
Conference	R : rayon	40,6	10,5	1,8	2,8	4,1	5,5
	F : faisceau	-	-	2,9	0,6	0,1	0,1
	$G : R + F$	40,6	10,5	4,7	3,4	4,2	5,6
	G_1/G_N	100%	26%	12%	**8%**	10%	14%
Fairy	R : rayon	44,7	11,9	3,3	4,6	5,8	7,0
	F : faisceau	-	-	3,1	0,6	0,1	0,1
	$G : R + F$	44,7	11,9	6,4	5,2	5,9	7,1
	G_1/G_N	100%	27%	14%	**12%**	13%	16%
Soda Hall	R : rayon	35,5	9,4	2,4	3,7	5,0	6,2
	F : faisceau	-	-	7,4	1,3	0,2	0,1
	$G : R + F$	35,5	9,4	9,8	4,9	5,2	6,3
	G_1/G_N	100%	26%	28%	**14%**	15%	18%

TAB. III.5 – Nombre d'étapes de traversée (en millions) selon la taille des groupes de rayons. Les étapes de traversée des faisceaux (F) et des rayons (R) sont dissociées. Les rayons SIMD réduisent par un facteur proche de 4 le nombre d'étapes de traversée, et les groupes de 64 rayons par un facteur supérieur à 7.

3.7 Discussion

Trois méthodes différentes ont été présentées pour rechercher un sous-ensemble de noeuds communs à un groupe de rayons. La recherche d'un unique point d'entrée et la recherche exhaustive des feuilles sont issues des travaux de Reshetov [66] et de Dmitriev *et al.* [21]. Dans les approches originales, les auteurs exploitent des objets terminaux pour arrêter au plus tôt le parcours du faisceau. Un objet terminal est un objet complètement opaque qui occupe toute la taille d'une feuille. Si une feuille de ce type est traversée complètement par le faisceau, alors il est inutile d'aller plus loin. L'efficacité de leur méthode, selon les auteurs, dépend fortement de la présence de ces objets terminaux.

Pour des raisons pratiques, cette notion n'a pas été implémentée dans nos expérimentations ; d'une part, la détection de ces objets est une tâche délicate qui augmente le temps de prétraitement d'un modèle ; d'autre part, l'opacité d'une surface peut dépendre soit de paramètres qui peuvent évoluer indépendamment de la géométrie, ou soit d'une texture. L'utilisation d'imposteurs, qui sont des surfaces rectangulaires dont la visibilité en chaque point dépend d'une texture, peut par exemple conduire à des résultats erronés d'occlusion s'ils sont classés comme des objets terminaux.

En prenant en compte ces considérations, la recherche du sous-arbre que nous proposons, est l'approche la plus efficace. Elle s'affranchit simplement des limitations des autres techniques en supprimant la totalité des noeuds visités en *commun* par le faisceau. Le traitement des rayons SIMD se limite alors à visiter les feuilles non vides, ainsi que les noeuds dont le chemin de traversée est resté indéterminé. Cette

méthode est également plus simple à mettre en oeuvre que la recherche d'un unique point d'entrée, qui nécessite l'utilisation d'une pile spéciale pour les bifurcations.

L'approche multi-passes de parcours du kd-Tree offre la possibilité de mémoriser les résultats de la première passe (la liste des noeuds) pour un usage différé et répété. Tant qu'il est possible de lancer des rayons inclus dans les chemins formés par les faisceaux, il est possible d'exploiter la liste des noeuds. Cette propriété est exploitée pour des calculs d'ombrage présentés dans la section 1.4 à la page 97.

Il est intéressant de comparer les performances de regroupements de rayons entre le kd-Tree et l'arbre BVH. Le chapitre suivant présente maintenant les groupes de rayons avec l'arbre BVH ainsi que les performances obtenues.

4 Groupes de rayons avec l'arbre BVH

4.1 Problématique

Avec l'arbre BVH, l'approche retenue pour propager le faisceau et le groupe de rayons se déroule dans une seule et même étape. La stratégie utilisée est directement inspirée de celle proposée par Wald *et al.* [77]. Il ne s'agit plus alors de trouver un dénominateur commun mais de propager simultanément le faisceau et l'ensemble des rayons. Une autre différence fondamentale par rapport au kd-Tree est caractérisée par la propagation d'un faisceau qui continue même pour un cas de *division* ; ceci a pour conséquence directe d'amener certains rayons du groupe à suivre alors un chemin qu'ils n'auraient pas suivi individuellement. Le danger de réaliser alors trop de calculs additionnels est minimisé avec la topologie de l'arbre BVH qui tolère, beaucoup mieux que le kd-Tree, cette divergence. Cette tolérance est expliquée en partie par une plus faible profondeur d'arborescence et la présence d'espaces vides plus grands dans un arbre BVH.

Afin de réaliser rapidement les tests de traversée du faisceau, une stratégie en trois étapes [77] est utilisée. Les tests d'intersections sont, quant à eux, évités pour les rayons quand le faisceau n'intersecte pas le triangle par la technique de *shaft culling* (voir la section 2.3 à la page 60).

Cette section présente dans un premier temps la stratégie de traversée en trois étapes. Ensuite, deux méthodes de rejet rapide d'une boîte par un faisceau — méthodes utilisées lors la seconde étape de la stratégie — sont discutées. Enfin, quelques adaptations par rapport à l'approche originale sont montrées dans le but de diminuer le nombre de calculs.

4.2 Stratégie de traversée en trois étapes

Contrairement à l'approche de propagation du faisceau pour le kd-Tree, la stratégie de traversée pour l'arbre BVH utilise aussi bien le faisceau modélisé par ses rayons encadrants que les rayons internes du groupe. La propagation du faisceau

dans la structure effectue au plus les trois tests suivants pour valider l'intersection d'un groupe de rayons avec une boîte :

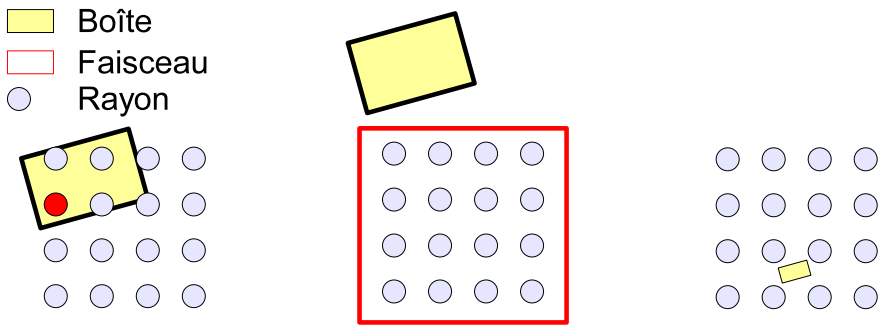
- *Entrée rapide* : si le rayon actif du groupe intersecte la boîte alors le faisceau traverse la boîte ;
- *Sortie rapide* : si le faisceau n'intersecte pas la boîte alors la boîte est rejetée ;
- *Premier entrée* : si un rayon du groupe intersecte la boîte alors le faisceau traverse la boîte sinon la boîte est rejetée ;

Entrée rapide. A chaque fois qu'un rayon intersecte une boîte alors celui-ci est marqué comme "rayon actif". En remarquant que ce rayon a une forte probabilité d'intersecter également les boîtes filles, celui-ci est utilisé avant tous les autres tests pour valider la traversée d'une boîte pour l'ensemble du groupe de rayons. On peut noter que le marqueur de rayon actif n'est pas global à toute la traversée mais valable uniquement pour le sous-arbre en cours. En cas de bifurcation, ce marqueur est empilé avec le noeud lointain afin d'être restauré quand ce noeud sera traité.

Sortie rapide. Suite au premier test, le rejet complet de la boîte est envisagé pour tout le groupe de rayons. Pour cela un test rapide mais conservatif est exécuté ; les détails d'implémentation sont expliqués en suivant.

Premier entrée. En dernier lieu, un test d'intersection est réalisé individuellement entre chaque rayon du groupe et la boîte. Dès qu'un rayon intersecte la boîte, celui-ci est marqué comme rayon actif et les autres tests sont suspendus. La traversée de la boîte est alors validée. Si aucun rayon n'intersecte la boîte alors celle-ci est finalement rejetée.

Il est bien sûr possible de remplacer cette stratégie en trois étapes par un test exact d'intersection du faisceau avec la boîte. Cependant, l'exécution de ce test, plus complexe, est moins rapide que les deux premières étapes cumulées. En considérant la troisième étape qui peut être coûteuse si l'ensemble des rayons est évalué, elle offre l'avantage, par rapport à un test de faisceau, d'éviter les petites boîtes qui passeraient entre les différents rayons, économisant ainsi des calculs d'intersections rayons/triangle inutiles. Ce cas est illustré dans la figure III.15.



(a) Entrée rapide : intersection avec le dernier rayon actif valide.

(b) Sortie rapide : rejet de la boîte par le faisceau.

(c) Premier entrée : aucun rayon n'intersecte la boîte.

FIG. III.15 – Stratégie de traversée en trois étapes : (a) entrée rapide, (b) sortie rapide et (c) premier entrée.

4.3 Rejet rapide d'une boîte pour un faisceau

La stratégie de traversée s'appuie principalement sur deux fonctions pour valider chaque étape. La première et la troisième étape utilisent une fonction d'intersection rayon/boîte. Cette fonction est celle présentée dans la section 3.1.1.2 à la page 39 à la différence près qu'elle est réalisée pour un rayon SIMD. Le résultat de cette fonction est alors valide si un des quatre rayons intersecte la boîte. La seconde étape de traversée nécessite une fonction de rejet de la boîte par le faisceau. Bien qu'un faisceau soit modélisé par un rayon SIMD, il n'est pas possible d'exploiter le résultat inverse de la fonction rayon/boîte, car le fait qu'aucun des quatre rayons ne traverse la boîte n'implique pas que le faisceau ne la traverse pas également. Ce problème ressemble à celui évoqué dans la figure III.8 de la section 3.2.1 à la page 64. Il est donc nécessaire de disposer d'un opérateur fiable de rejet d'une boîte pour un faisceau donné.

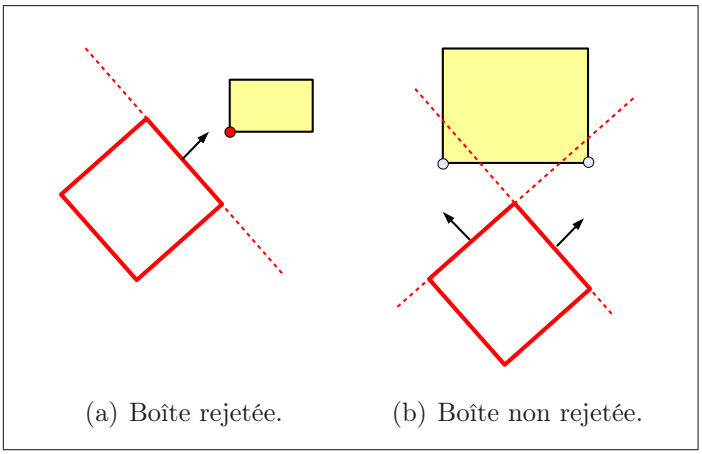
4.3.1 Rejet des sommets de la boîte par les plans du faisceau

Une approche classique dite de *frustum culling* consiste à rejeter une boîte par les plans définis par un faisceau. Assarson *et al.* [4] proposent une implémentation optimisée en limitant les tests pour chacun des plans à un sommet particulier de la boîte. Pour chaque plan du faisceau, seul le sommet potentiellement le plus proche du plan est utilisé. Ce sommet est facile à déterminer car il ne dépend que du signe de la normale au plan. Une boîte peut être modélisée par deux sommets extrêmes : soit $s0$ celui ayant les plus petites coordonnées et $s1$ celui ayant les plus grandes coordonnées. Les coordonnées du sommet sn le plus proche du plan P ayant pour

équation $N.p + K = 0$ sont :

$$\begin{aligned} sn_x &= \text{si } N_x > 0 \text{ alors } s0_x \text{ sinon } s1_x \\ sn_y &= \text{si } N_y > 0 \text{ alors } s0_y \text{ sinon } s1_y \\ sn_z &= \text{si } N_z > 0 \text{ alors } s0_z \text{ sinon } s1_z \end{aligned}$$

Pour un plan donné, il suffit d'intégrer le sommet sn dans l'équation du plan P pour vérifier si celui-ci se situe à l'extérieur ou l'intérieur du plan. Bien que ces calculs soient très rapides, la technique de *frustum culling* est mieux adaptée quand la taille du faisceau est plus grande que celle des boîtes. Dans le cas inverse, cette méthode détectera moins efficacement le rejet de la boîte comme illustré dans la figure III.16, reportant ainsi plus de calculs dans la troisième étape de la stratégie de traversée.



(a) Boîte rejetée. (b) Boîte non rejetée.

FIG. III.16 – Technique de rejet de la boîte par *frustum culling*. Cette technique montre des limites (b) quand la boîte est grande par rapport au faisceau.

4.3.2 Rejet de la boîte par distances extrêmes

Une méthode utilisant les segments de rayons [66] permet de rejeter plus efficacement les boîtes. Pour l'appliquer, les distances d'entrée tn et de sortie tf sont calculées pour les quatre rayons du faisceau avec les six plans de la boîte. La base de cet algorithme est la même que celle du calcul d'intersection rayon/boîte, sauf qu'au lieu de vérifier individuellement la cohérence de chaque segment de rayons, les distances minimales et maximales par plan de tous les rayons sont utilisées. A partir de ces valeurs, il est possible de déduire le rejet si pour chaque plan $\widehat{XY}$, $\widehat{XZ}$ et $\widehat{ZX}$ notés d'une manière générale $\widehat{UV}$ les inéquations suivantes sont vérifiées :

$$min(tn_u) > max(tf_v) \text{ ou } min(tn_v) > max(tf_u)$$

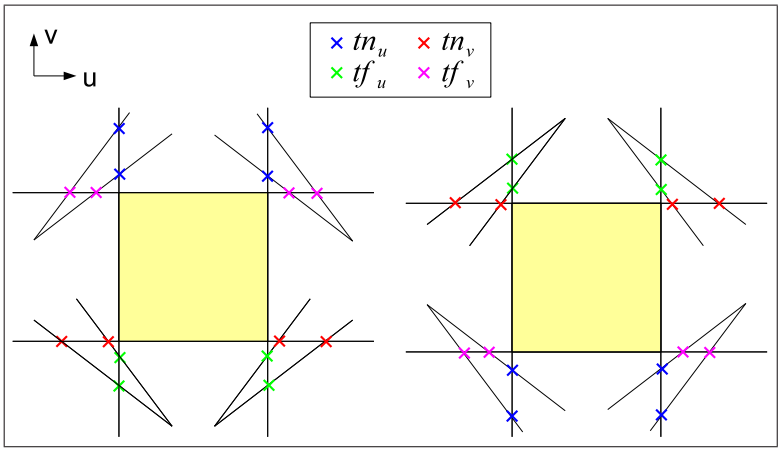


FIG. III.17 – Rejet de la boîte par utitlisation des distances extrêmes selon les huit directions principales du faisceau.

La figure III.17 illustre les différents cas possibles pour chacun des trois plans. La figure III.18 présente une implémentation SIMD de ce test de rejet rapide.

```
#define _T4(A, B) _mul_ps(_sub_ps(bb4[B][A], r4->o4[A]), r4->id4[A])
const float min_tin_x = _hmin4(_T4(0, 0)), max_tout_y = _hmax4(_T4(1, 1));
if (min_tin_x <= max_tout_y) { //cohérence en x^y
  const float min_tin_y = _hmin4(_T4(1, 0)), max_tout_x = _hmax4(_T4(0, 1));
  if (min_tin_y <= max_tout_x) { //cohérence en x^y
    const float min_tin_z = _hmin4(_T4(2, 0)), max_tout_z =_hmax4(_T4(2, 1));
    if (min_tin_x <= max_tout_z && min_tin_z <= max_tout_x && //cohérence en x^z
      min_tin_y <= max_tout_z && min_tin_z <= max_tout_y) //cohérence en y^z
      return 0; //boîte non rejetée
  }
}
return 1; //boîte rejetée
```

FIG. III.18 – Fonction SIMD de rejet rapide d'une boîte. Les fonctions *_hmin4* et *_hmax4* sont des fonctions dites "horizontales" qui donnent la valeur scalaire minimale (ou maximale) d'un vecteur SIMD de quatre réels.

Comme le montre les résultats d'expérimentations du tableau III.6, l'utilisation des distances extrêmes permet d'avoir une meilleure efficacité de rejet des boîtes.

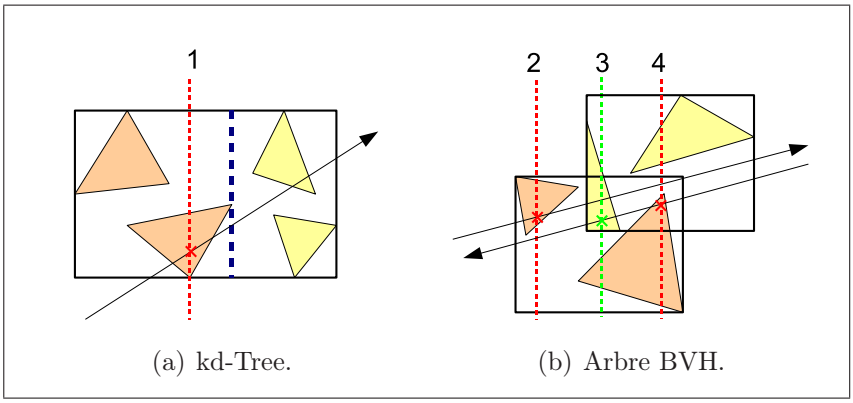
Modèles	# tests faisceau/boîte	# rejets par plans	# rejets par distances extrêmes
Conference	916	310 (34%)	352 (38%)
Fairy	1 153	417 (36%)	447 (39%)
Soda Hall	1 076	313 (29%)	388 (36%)

TAB. III.6 – Comparaison des efficacités de rejets des boîtes, pour un rendu de 1024x1024 rayons primaires avec des groupes de 64 rayons, selon l'utilisation des plans ou des distances extrêmes. Les nombres sont exprimés en milliers d'opérations. La méthode de rejet par les distances extrêmes offre les meilleures résultats pour les modèles présentés.

4.4 Adaptations

4.4.1 Intersection rayon/boîte en fonction de l'axe dominant

Contrairement au kd-Tree, l'arbre BVH ne garantit pas un ordre de traversée strict car les boîtes d'un arbre BVH peuvent se chevaucher. La figure III.19 illustre ce problème en montrant la possibilité pour le kd-Tree d'arrêter au plus tôt la traversée d'un rayon et l'impossibilité pour l'arbre BVH de réaliser la même opération. Cependant dans le deuxième exemple avec l'arbre BVH, le point d'intersection dans la boîte la plus proche se situe avant le point d'entrée de la boîte lointaine. Bien que l'arbre BVH ne comporte pas de plan séparateur, pour chaque boîte un axe dominant selon lequel les boîtes filles sont le plus éloignées est défini. Lors de la traversée, cet axe dominant est exploité afin de choisir l'ordre de parcours des boîtes filles selon la direction du rayon. Cela permet de réduire le nombre de tests d'intersection, car une occlusion détectée dans la boîte proche aura pour conséquence de réduire la distance maximale du rayon t_{max} qui est utilisée ensuite dans la fonction d'intersection rayon/boîte pour la boîte lointaine. Si cette distance maximale est plus petite que la distance d'entrée dans la boîte lointaine alors celle-ci sera effectivement rejetée.



(a) kd-Tree. (b) Arbre BVH.

FIG. III.19 – Bénéfice de l'ordre de traversée pour le kd-Tree (a). Pour l'arbre BVH (b), bien que l'impact 2 soit le plus proche le long du rayon, le chevauchement des boîtes ne garantit pas un ordre de traversée strict. L'impact 3, trouvé dans la boîte la plus proche, montre qu'il est possible de trouver une intersection plus proche (impact 4) dans la boîte lointaine.

Pour aller plus loin dans cette stratégie, la fonction d'intersection rayon/boîte a été modifiée afin de commencer les tests de rejet par l'axe dominant en priorité. Pour rappel, ce test est utilisé lors des étapes *Entrée Rapide* et *Premier entrée* de la stratégie de traversée. La figure III.20 montre la mise en place de cette stratégie pour un rayon SIMD. Les distances des points d'entrée $tn4$ et de sortie $tf4$ sont initialement définies avec le segment du rayon $r4 \rightarrow tmin4$ et $r4 \rightarrow tmax4$. Ces distances sont ensuite mises à jour pour chaque axe en commençant par l'axe dominant de la boîte. Dès que les distances deviennent invalides ($tn4 > tf4$) alors la fonction retourne un cas de non intersection. Ainsi, un rayon occulté par une surface incluse dans une boîte proche, engendrera moins d'itérations dans la fonction rayon/boîte pour la boîte lointaine (voir le tableau III.7).

```
const int m3[5] = {0,1,2,0,1};
int i, a;
__m128 tn4 = r4->tmin4, tf4 = r4->tmax4;
//itération sur les axes en commençant par un axe dominant
for (i = 0, a = split_axis; i < 3; i++, a = m3[++split_axis]) {
  //distances au plan "a"
  tn4 = _max_ps(tn4, _mul_ps(rx_4->id4[a], _sub_ps(bb4[0][a], rp->o4[a])));
  tf4 = _min_ps(tf4, _mul_ps(rx_4->id4[a], _sub_ps(bb4[1][a], rp->o4[a])));
  //sortie rapide si non intersection
  if (!_movemask_ps(_cmpngt_ps(tn4, tf4))) return 0;
}
//intersection
return 1;
```

FIG. III.20 – Fonction d'intersection SIMD rayon/boîte adaptée pour commencer par un axe dominant.

Modèles	# tests rayon/boîte	Choix du 1^{er} axe	# rejets par axe		
			1^{er} axe	$2^{ème}$ axe	$3^{ème}$ axe
Conference	1 898	x	587 (31%)	396 (21%)	360 (19%)
		dominant	875 (46%)	305 (16%)	163 (09%)
Fairy	2 493	x	841 (34%)	499 (20%)	340 (14%)
		dominant	969 (39%)	429 (17%)	283 (11%)
Soda Hall	2 871	x	984 (34%)	436 (15%)	743 (26%)
		dominant	1 397 (49%)	475 (17%)	290 (10%)

TAB. III.7 – Nombre de rejets (en milliers) rayon/boîte pour un rendu de 1024x1024 rayons primaires. Différents niveaux de rejets sont dissociés selon l'axe qui permet d'exclure le rayon et la boîte. Selon le premier axe utilisé, arbitrairement l'axe x ou volontairement l'axe dominant de la boîte, la répartition des rejets diffère. En choisissant l'axe x, un tiers des tests peuvent être arrêtés dès le premier axe, tandis qu'avec l'axe dominant de la boîte, près de la moitié des tests bénéficie d'un arrêt rapide. On peut remarquer que le nombre de rejets total est le même quel que soit le choix du premier axe, et que seule la répartition entre les différents niveaux de rejet varie.

4.4.2 Retrait des rayons occultés pour les requêtes d'intervisibilité

Les requêtes d'intersection peuvent être de natures différentes. Dans le cas général, seule la recherche de la plus proche surface est analysée. Cependant, la requête d'intervisibilité entre deux points est également très répandue et mérite des optimisations particulières. En effet, dès qu'un rayon du groupe est occulté, il devient inutile par la suite de réaliser des tests d'intersection rayon/boîte ou rayon/triangle car seule l'information d'occlusion suffit.

Pour prendre en compte cet avantage, la liste des rayons d'un groupe est implémentée sous la forme d'une liste chaînée. Ainsi, dès qu'une occlusion est détectée, ce rayon est retiré de la liste réduisant ainsi le nombre de calculs (voir le tableau III.8). On peut remarquer que cette approche peut tout aussi bien être implémentée en utilisant un masque des rayons occultés. Cependant, le test additionnel pour vérifier la valeur de ce masque est plus coûteux que l'utilisation de la liste des rayons. En effet, le retrait du rayon de la liste évite ce test et permet de ne travailler qu'avec l'ensemble des rayons actifs.

Modèles	Occlusion	# traversées	# intersections
Conference	sans retrait	1 011	1 334
	avec retrait	736	617
	avec/sans	73%	46%
Fairy	sans retrait	1 483	1 753
	avec retrait	1 286	1 175
	avec/sans	87%	67%
Soda Hall	sans retrait	1 322	1 447
	avec retrait	945	897
	avec/sans	71%	62%

TAB. III.8 – Nombre d'étapes de traversée et de calculs d'intersection (en milliers) pour un rendu de 1024x1024 rayons d'ombre. Le retrait des rayons occultés réduit de manière très significative aussi bien le nombre d'étapes de traversée que le nombre de calculs d'intersection.

4.5 Résultats

Le tableau III.9 présente les performances brutes obtenues avec les groupes de rayons pour l'arbre BVH. Les résultats montrent le nombre de requêtes d'intersection (en millions) exécutées par seconde pour un rendu de 1024x1024 rayons primaires. Le jeu de tests est le même que celui utilisé pour le kd-Tree (voir la section 3.6 à la page 71). Les meilleures performances sont obtenues pour des groupes constitués de 64 ou 256 rayons. L'accélération décline ensuite pour des groupes plus grands. Cette courbe de performances "en cloche" est due à un équilibre qui doit être tenu entre un nombre de rayons suffisamment important pour factoriser au mieux les calculs, et pas trop grand, pour éviter une divergence excessive du faisceau. Cette divergence se traduit par un léger accroissement du nombre d'étapes de traversée (voir le tableau III.12) et par une grande augmentation du nombre de calculs d'intersection rayon/triangle (voir le tableau III.13).

	Mr/s			
# rayons	16	64	256	1024
Armadillo	5,0	**5,6**	3,7	1,5
Buddha	**4,4**	4,0	2,1	0,8
Bunny	6,9	**10,7**	9,3	4,9
Clipper	3,1	**4,5**	3,2	1,3
Conference	4,2	8,2	**8,4**	4,6
ERW6	8,9	19,1	**24,8**	17,4
Fairy	3,3	**5,8**	5,3	2,6
Soda	3,7	**6,8**	6,7	3,4
Sully	2,4	5,1	**5,7**	3,3
Villa Digy	2,5	5,8	**7,6**	4,4
Moyenne	4,5	7,6	**7,7**	4,4

TAB. III.9 – Millions de requêtes d'intersection exécutées par seconde pour un rendu de 1024x1024 rayons primaires. Les résultats sont dissociés pour différentes tailles de groupes de rayons. Les meilleures performances sont obtenues pour des groupes composés de 64 ou 256 rayons.

En comparant les performances des rayons SIMD aux simples rayons (voir le tableau III.10), on peut remarquer une très bonne accélération moyenne de x3,6. L'accélération des groupes de 64 rayons varie entre x2,8 (Buddha) et x20,5 (Vila Digy) avec une moyenne de x10,5. Les raisons de ces variations s'expliquent, comme pour le kd-Tree, par la géométrie des modèles propices ou non à la factorisation des calculs.

	Mr/s			Accélération	
# rays	1	4	64	4/1	64/1
Armadillo	1,1	3,7	5,6	x3,4	x5,3
Buddha	1,4	4,1	4,0	x2,8	x2,8
Bunny	1,1	3,9	10,7	x3,7	x10,1
Clipper	0,5	1,6	4,5	x3,4	x9,7
Conference	0,5	2,0	8,2	x3,7	x15,5
ERW6	1,1	4,1	19,1	x3,7	x17,1
Fairy	0,5	1,7	5,8	x3,7	x12,9
Soda	0,5	2,0	6,8	x3,7	x13,0
Sully	0,3	1,0	5,1	x3,5	x17,7
Villa Digy	0,3	1,1	5,8	x3,8	x20,5
Moyenne	0,7	2,5	7,6	x3,6	x10,5

TAB. III.10 – Millions de requêtes d'intersection exécutées par seconde pour un rendu de 1024x1024 rayons primaires. Comparaison des performances des groupes de 64 rayons par rapport à des rayons simples et des rayons SIMD.

Contrairement au kd-Tree, l'apport du masquage de triangles est efficace avec

l'arbre BVH. Le tableau III.11 montre pour les trois modèles présentés un gain de performances toujours favorable et supérieur à x1,25 pour les modèles *Conference* et *Fairy*. Ce gain est obtenu par une réduction efficace de 23% à 48% du nombre de calculs d'intersection.

	Sans masquage	Avec masquage	
Modèles	I_{ref} : # tests	# tests	
Conference	2 988	I : intersection	1 334
		M : masquage	229
		$(I+M)/I_{ref}$	52%
		accélération	x1,25
Fairy	2 594	I : intersection	1 753
		M : masquage	237
		$(I+M)/I_{ref}$	77%
		accélération	x1,35
Soda Hall	2 176	I : intersection	1 447
		M : masquage	200
		$(I+M)/I_{ref}$	76%
		accélération	x1,05

TAB. III.11 – Nombre de tests d'intersection (en milliers) selon le masquage ou non des triangles. Les tests d'intersection (I) sont réalisés par les rayons et les tests de masquage (M) sont réalisés par les faisceaux. La réduction du nombre de tests d'intersection est supérieure à 23% pour les modèles présentés et apporte un facteur d'accélération allant jusqu'à x1,35.

On peut remarquer que les facteurs d'accélération des groupes de rayons sont très importants pour l'arbre BVH. Sur les dix modèles présentés, sept bénéficient d'une accélération supérieure à x9. Ce gain s'explique par une large réduction du nombre d'étapes de traversée comme le montre le tableau III.12 sur la page suivante. Pour les groupes constitués de plus de 16 rayons, le nombre d'étapes est réduit par un facteur supérieur à 20.

		# traversées selon le nombre de rayons					
Modèles	# rayons	1	4	16	64	256	1024
Conference	T : # traversées	51,3	13,0	3,4	1,0	0,5	0,5
	T_1/T_N	100%	25%	6,6%	2,0%	0,9%	0,9%
Fairy	T : # traversées	59,4	15,4	4,3	1,5	0,9	1,2
	T_1/T_N	100%	26%	7,2%	2,5%	1,6%	2,0%
Soda Hall	T : # traversées	54,2	14,1	3,9	1,3	0,8	1,0
	T_1/T_N	100%	26%	7,2%	2,4%	1,5%	1,8%

Tab. III.12 – Nombre d'étapes de traversée (en millions) selon la taille des groupes de rayons. Les rayons SIMD réduisent par un facteur proche de 4 le nombre d'étapes de traversée, et les groupes de 64 rayons par un facteur supérieur à 20 pour les groupes constitués de plus de 16 rayons.

Le tableau III.13 montre le nombre de calculs d'intersection selon la taille des groupes de rayons. Une seule réduction effective du nombre d'intersections, de l'ordre d'un tiers, est observée pour les rayons SIMD. Contrairement aux étapes de traversée, et bien que le masquage de triangles soit utilisé, le nombre de calculs d'intersections ne réduit pas ou peu avec l'utilisation des groupes de rayons. *Conference* est le seul modèle qui bénéficie d'une légère réduction pour les groupes de 16 et 64 rayons ; cette particularité peut être mise en rapport avec une bonne efficacité de masquage des triangles (voir le tableau III.11).

		# intersections selon le nombre de rayons					
Modèles	# rayons	1	4	16	64	256	1024
Conference	I : # intersections	7,5	2,1	1,4	1,6	2,9	6,6
	I_1/I_N	100%	28%	18%	21%	38%	87%
Fairy	I : # intersections	5,1	1,5	1,5	2,0	4,2	11,2
	I_1/I_N	100%	29%	29%	39%	82%	219%
Soda Hall	I : # intersections	4,2	1,2	1,2	1,6	3,4	8,8
	I_1/I_N	100%	30%	29%	39%	81%	208%

Tab. III.13 – Nombre de tests d'intersections (en millions) selon la taille des groupes de rayons. Les rayons SIMD réduisent par un facteur proche de 3 ces tests. Ce facteur de réduction est maintenu pour les groupes d'une taille $\geq$ à 64 rayons. Pour les groupes plus grands, la proportion de rayons inactifs dans une feuille entraîne un accroissement de ces tests pouvant doubler par rapport aux rayons simples.

4.6 Discussion

Cette section a présenté une implémentation de la requête d'intersection pour un groupe de rayons avec l'arbre BVH. La stratégie de propagation utilisée s'appuie sur les travaux de Wald *et al.* [77]. Celle-ci propage simultanément le faisceau et les rayons du groupe dans l'arbre. Deux adaptations sont proposées afin de réduire les

tests d'intersection rayons/boîtes et d'intersection de rayons/triangles. La première débute le test d'intersection rayon/boîte par l'axe dominant de la boîte. Cet axe, issu de la construction de l'arbre, permet un rejet plus efficace de la boîte dès le premier test. La deuxième adaptation optimise les calculs d'intervisibilité en enlevant dans un groupe les rayons occultés. Ces retraits permettent aussi bien de réduire les tests de traversées que les tests d'intersection.

Les résultats montrent que les groupes de rayons avec l'arbre BVH offrent un facteur d'accélération très important par rapport aux rayons simples. Cette accélération est essentiellement due à une grande réduction du nombre d'étapes de traversée. Contrairement à la stratégie de traversée du kd-Tree, des rayons peuvent suivre des chemins qu'ils n'auraient pas empruntés individuellement. Cependant, grâce au masquage efficace des triangles, le nombre de calculs d'intersection rayon/triangle reste raisonnable pour des groupes de moins de 1024 rayons.

Conclusion

Ce chapitre a présenté des techniques pour lancer des rayons par groupe. En faisant l'hypothèse de disposer d'un ensemble de rayons cohérents, nous avons montré comment créer rapidement l'enveloppe convexe qui englobe tous les rayons. Nous avons retenu une méthode de construction de cette enveloppe qui exploite les rectangles formés par les points d'entrée et de sortie des rayons dans la boîte englobante du modèle. Cette méthode est à la fois rapide, et permet d'avoir un faisceau resserré sur l'ensemble des rayons. Pour modéliser le faisceau, nous avons choisi d'utiliser quatre rayons encadrants qui permettent directement de réutiliser les méthodes SIMD d'intersection rayon/boîte et rayon/triangle.

Les groupes de rayons ont été intégrés dans les deux structures de recherche étudiées : le kd-Tree et l'arbre BVH. Les méthodes de propagation des groupes de rayons diffèrent alors selon la structure utilisée. Pour le kd-Tree, une approche en deux passes est réalisée. Lors de la première passe, seul le faisceau est propagé dans l'arbre pour collecter un sous-ensemble des noeuds et des feuilles. A partir de ce sous-ensemble, chaque rayon du groupe est exécuté individuellement, évitant ainsi de reparcourir les parties de l'arbre communes. En comparant plusieurs approches pour récupérer ce sous-ensemble, nous avons privilégié une technique qui supprime le sous-arbre commun aux différents rayons par rapport à d'autres techniques qui soit, recherchent un unique point d'entrée [66], ou bien collectent l'ensemble des feuilles non vides [21]. L'efficacité globale des regroupements de rayons avec le kd-Tree permet alors un facteur d'accélération moyen de x3,6 par rapport aux rayons simples.

L'exécution des groupes de rayons avec l'arbre BVH propage le faisceau et les rayons dans un même temps. La stratégie de propagation s'appuie sur un test d'intersection faisceau/boîte réalisé en trois étapes [77] ; nous avons apporté deux adaptations à cette stratégie. Les première et troisième étapes de cette stratégie utilisent un test d'intersection rayon/boîte que nous avons adapté pour rejeter plus rapide-

ment les boîtes. Ce test vérifie le rejet selon les trois axes en commençant d'abord par l'axe qui a permis de séparer les deux boîtes filles lors de la construction de l'arbre. Par l'utilisation de cet axe dominant, la probabilité de rejet pour le premier axe passe d'un tiers à près de la moitié des tests. La seconde adaptation permet d'accélérer les requêtes d'intervisibilité, comme par exemple les rayons d'ombrage, en enlevant les rayons dès qu'ils sont occultés.

Les groupes de rayons pour l'arbre BVH offrent une facteur d'accélération moyen de x10,5 par rapport aux rayons simples. On peut noter que cette accélération est bien supérieure à celle apportée par les groupes de rayons pour le kd-Tree. Cependant, ces valeurs sont comparées respectivement aux rayons simples de chaque arbre ; la comparaison des performances brutes ne reflète pas un tel écart même si l'arbre BVH reste tout de même avantagé par les groupes de rayons. Le chapitre 1 à la page 118 compare en détails ces performances et présente une solution pour tirer partie d'un antagonisme entres les deux structures qui ont des points forts et des points faibles opposés.

On peut s'interroger sur la possibilité d'interchanger les méthodes de propagation des faisceaux entre le kd-Tree et l'arbre BVH. Bien que nous n'ayons pas tester ces solutions, nous pensons que les méthodes actuelles sont les plus adaptées aux différentes structures de recherche. Il serait possible pour le kd-Tree, lors de la recherche du sous-arbre, d'évaluer dans le même temps que la propagation du faisceau, les rayons du groupe. Cependant, la stabilité du nombre de tests d'intersections rayons/surfaces lors de l'exécution de rayons SIMD ou de groupes de rayons de différentes tailles, prouve que le fait de différer l'exécution des rayons n'est pas pénalisant. D'autre part, dans le contexte des calculs d'intervisibilité, la section 1.4 à la page 97 montre qu'il est intéressant de mémoriser le résultat de la propagation du faisceau pour l'exploiter ultérieurement par tous rayons qui passeraient dans ce faisceau. Plus généralement, le chapitre suivant montre comment tirer partie des groupes de rayons afin d'accélérer les calculs d'intervisibilités.

Chapitre IV

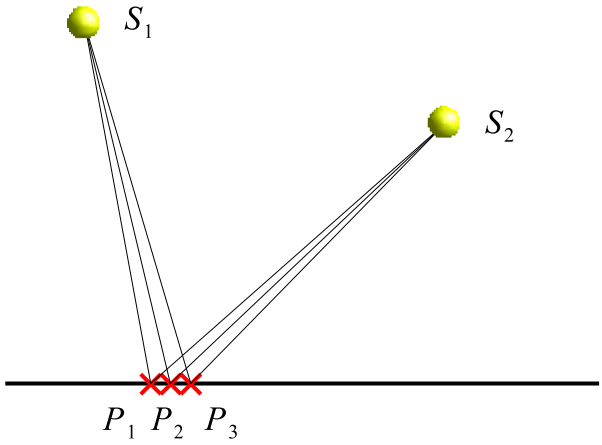
Accélération des calculs d'intervisibilité à origine commune

Introduction

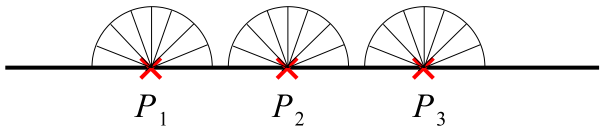
Le chapitre précédent a montré d'une manière générale la performance obtenue avec des groupes de rayons cohérents. Ce chapitre propose d'exploiter la cohérence existante pour une application bien spécifique mais omniprésente dans les applications de la requête d'intersection : l'intervisibilité pour un groupe de rayons à origine commune.

L'intervisibilité peut être modélisée par une fonction $iv(p_a, p_b)$ qui indique de manière binaire la visibilité entre deux points p_a et p_b. Si l'on doit résoudre plusieurs cas d'intervisibilité entre les paires disjointes $\{p_a, p_b\}$ tels que $p_a \in A$ et $p_b \in B$, il devient intéressant d'exploiter des requêtes d'intersection groupées. Cependant, pour que l'exécution de ces groupes soit efficace, les ensembles A et B doivent être individuellement cohérents. Si la taille de l'un de ces ensembles A ou B est réduite à un seul point, alors la probabilité de construire des groupes de rayons cohérents est grande. Ce dernier cas de figure est étudié dans ce chapitre.

Les fonctions $iv(p_a, p_b)$ et $iv(p_b, p_a)$ étant équivalentes, les ensembles A et B peuvent alors représenter indifféremment l'ensemble des points d'origine ou l'ensemble des points destination. L'ensemble de taille un est choisi pour représenter l'origine commune. Ce choix est induit par les méthodes de traversée des groupes de rayons car il permet de retarder la divergence des rayons, voire de l'éviter si des occlusions sont trouvées rapidement.



(a) Ombrage de sources ponctuelles.



(b) Occlusion ambiante.

FIG. IV.1 – Intervisibilités à origine commune pour l'ombrage de sources ponctuelles (a), et l'occlusion ambiante (b).

Bien que des similarités existent entre tous les domaines d'application, il est difficile de proposer une solution générale sur le thème des groupes de rayons à origine commune. Deux cas d'application dans la synthèse d'image ont donc été choisis afin de mettre en valeur ces regroupements particuliers (voir la figure IV.1) : l'ombrage d'une source ponctuelle [85] et le calcul de l'occlusion ambiante [86]. Ces deux cas s'opposent sur les quatre points suivants :

- *Angle d'ouverture.* L'occlusion ambiante requiert un angle ouverture extrêmement large puisqu'il s'agit d'une évaluation hémisphérique de l'intervisibilité pour un point. Cet angle est beaucoup plus resserré pour les sources ponctuelles.
- *Origine des rayons.* L'origine des rayons correspond dans un cas à la position de la source ponctuelle et dans l'autre cas au point d'évaluation de l'occlusion ambiante. Dans le dernier cas, le point d'évaluation est toujours situé au niveau de la géométrie ce qui implique une arborescence plus complexe de la structure de recherche en ce point, et donc une moins bonne cohérence des groupes de rayons.
- *Direction des rayons.* Pour l'ombrage d'une source ponctuelle, la direction des rayons est imposée par les points à éclairer. Cette contrainte implique de ne pas maîtriser ces directions et donc éventuellement de devoir les réorganiser pour optimiser la cohérence. Pour l'occlusion ambiante, le choix de ces directions peut être réalisé au préalable dans un but de cohérence.
- *Portée des rayons.* La portée de l'effet de l'occlusion ambiante est limitée à un rayon prédéfini. Pour l'ombrage d'une source ponctuelle, cette portée est

contrainte par les points à évaluer et est, par expérience, toujours plus grande. Cette différence est importante car la divergence augmente généralement avec la portée des rayons.

Nous pensons que ces différences font que ces deux cas permettent de couvrir un nombre important de variantes de la requête d'intervisibilité pour un groupe de rayons à origine commune. Les expérimentations et solutions apportées sont présentées dans les sections suivantes.

1 Ombrage d'une source ponctuelle

1.1 Problématique

La requête d'intervisibilité est très sollicitée en image de synthèse pour vérifier l'éclairage d'une source lumineuse. Dans l'algorithme classique du lancer de rayons, elle permet de déduire l'illumination directe d'un point en lançant des rayons d'ombre vers chaque source [85]. Le nombre de rayons dépend alors de la résolution de l'image, ainsi que du nombre de sources qui peut atteindre plusieurs milliers quand elles sont utilisées pour simuler de l'illumination globale, comme par exemple avec les sources de lumières virtuelles (VPL) [41]. Une simulation correcte peut alors nécessiter de lancer plusieurs dizaines de millions de rayons d'intervisibilité qu'il convient d'accélérer par des groupes de rayons.

Par nature, la requête d'intervisibilité est plus rapide que la requête d'intersection car l'exécution est arrêtée dès la première intersection trouvée. Cette opportunité permet d'ajouter une optimisation en mémorisant la dernière surface occultante [28]. Ainsi, il est possible d'éviter l'exécution de la requête d'intervisibilité si une intersection est préalablement vérifiée avec la dernière surface occultante. Bien que cette technique apporte un gain de performances certain, elle reste limitée aux rayons "ombrants" par opposition aux rayons "éclairants". Sur ce dernier point, le regroupement de rayons peut apporter un gain de performances additionnel.

Dans nos expérimentations, les points d'impacts issus des rayons primaires constituent l'ensemble de points de destination vers lesquels sont exécutées les requêtes groupées d'intervisibilité. Etant donné que les rayons primaires sont aisément groupés en formant des rectangles de pixels connexes, les points d'impact issus de ces rayons sont généralement cohérents. Ces groupes de points d'impact sont directement utilisés dans une première expérimentation pour former le faisceau d'ombre. Dans une seconde expérimentation, une technique de réorganisation des points d'impact est présentée pour réduire l'angle d'ouverture des faisceaux. Dans une dernière expérimentation, les rayons d'ombre exploitent une liste de points d'entrée précalculée lors de la propagation d'un faisceau dans le kd-Tree (voir la section 3 à la page 62).

1.2 Réutilisation d'un groupe de points

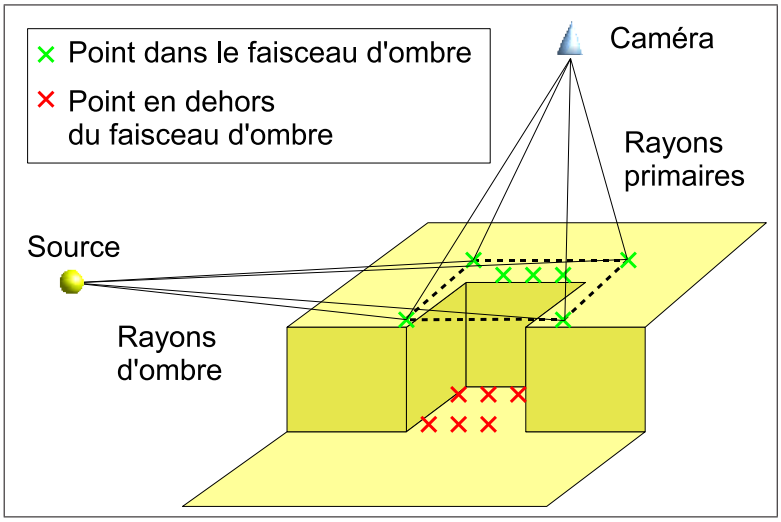


FIG. IV.2 – Points d'impacts situés en dehors du faisceau d'ombre formé par la réutilisation directe des rayons encadrants primaires.

L'approche classique consiste à réutiliser le groupe de points d'impacts formé par les rayons primaires. Parce qu'il est impossible de réutiliser les points d'impacts des rayons encadrants du faisceau primaire comme le montre la figure IV.2, le faisceau d'ombre doit être reconstruit. Les techniques des pentes extrêmes ou des rectangles (voir les sections 1.2 à la page 54 et 1.3 à la page 56) peuvent être utilisées à cette fin en itérant sur chaque point d'impact. Cependant, par l'exploitation de l'origine commune des rayons, il est plus simple et plus rapide de construire le faisceau d'ombre à partir de la boîte englobante formée par les points d'impact primaires.

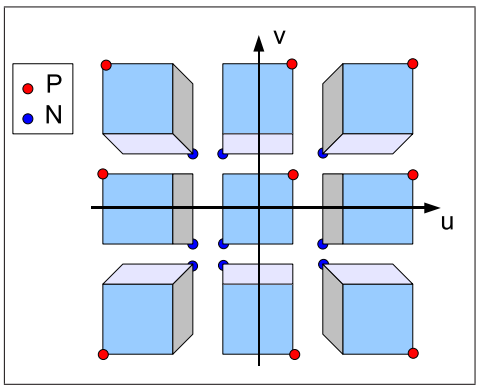


FIG. IV.3 – Vue depuis la source ponctuelle des extrémités P et N de la boîte engloblante des points d'impacts, utiles à la construction du faisceau d'ombre.

En se plaçant du point de vue de la source lumineuse et selon la direction principale du faisceau d'ombre, la figure IV.3 illustre les différents positionnements possibles de cette boîte par rapport aux directions secondaires U et V du faisceau. Deux

sommets particuliers de la boîte, nommés N et P, sont utiles à la construction. Le point P est le sommet le plus éloigné de l'axe principal et situé en avant plan, réciproquement le sommet N est le sommet le plus proche de l'axe principal et situé en arrière plan. En considérant les sommets extrêmes $s0$ et $s1$ de la boîte englobante, l'axe principal du faisceau W, les axes secondaires U et V et l'origine commune des rayons O, alors les points P et N ont pour coordonnées :

$$\begin{aligned}
P_u &= si\ s1_u \geq O_u\ alors\ s1_u\ sinon\ s0_u \\
P_v &= si\ s1_v \geq O_v\ alors\ s1_v\ sinon\ s0_v \\
P_w &= s0_w \\
N_u &= si\ s0_u < O_u\ alors\ s1_u\ sinon\ s0_u \\
N_v &= si\ s0_v < O_v\ alors\ s1_v\ sinon\ s0_v \\
N_w &= s1_w
\end{aligned}$$

En projetant le point P sur l'arrière plan $\widehat{UV}$, on obtient un point P' qui associé au point N forme un rectangle ; ce dernier définit alors les limites du faisceau d'ombre. En retranchant les quatre sommets du rectangle à l'origine commune, on obtient directement les directions des quatre rayons formant le faisceau d'ombre. Par rapport aux méthodes présentées dans la section 1 à la page 53, celle-ci requiert moins d'opérations.

1.3 Réorganisation des rayons d'ombre

1.3.1 Principe

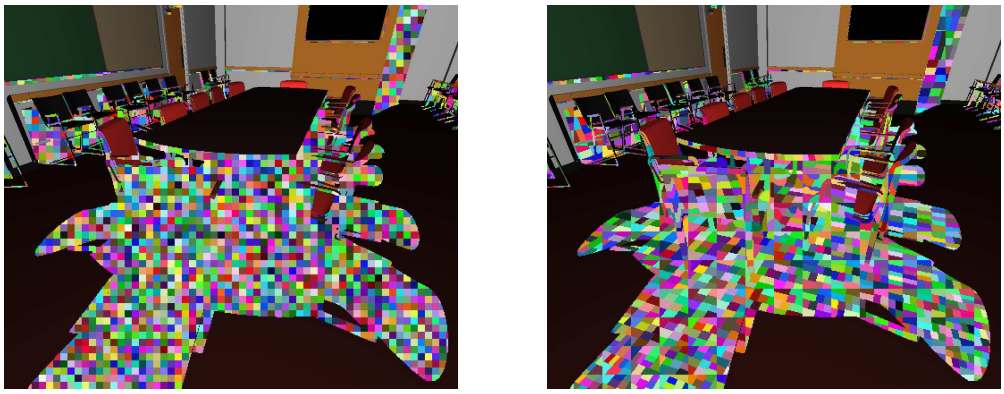
En observant la figure IV.2, on s'aperçoit que la cohérence issue des rayons primaires est dégradée quand les points d'impacts sont situés sur des surfaces décalées. Cette situation est particulièrement remarquable quand la source lumineuse est orthogonale à la direction des rayons primaires, car les faisceaux d'ombre résultants sont alors excessivement ouverts. Pour remédier à cela, une approche a été expérimentée pour réorganiser les points d'impacts afin de former des faisceaux plus cohérents. La difficulté principale est de parvenir à une réorganisation efficace rapidement.

Afin de faciliter la réorganisation spatiale, une projection 2D de tous les points d'impacts est utilisée. Celle-ci est réalisée sur les faces d'un cube virtuel centré sur la source lumineuse. La face de projection choisie dépend de l'axe majoritaire et du signe de la direction des rayons d'ombres. Chaque face collecte un ensemble de points référencés par leurs coordonnées 2D p_u et p_v sous la forme d'une liste chaînée. Soit d la direction d'un rayon d'ombre issu d'un point d'impact, W son axe majoritaire et U et V ses axes secondaires, les coordonnées du point projeté p sont :

$$\begin{aligned}
p_u &= d_u/d_w \\
p_v &= d_v/d_w
\end{aligned}$$

A partir des points projetés, une arbre binaire 2D est construit pour chaque face. Pour que cette construction soit rapide et ne représente pas un temps additionnel au traitement des rayons d'ombres, l'algorithme de répartition des points est simplifié. Les points sont séparés récursivement selon leurs coordonnées p_u ou p_v par rapport à la ligne séparatrice médiane de la plus grande dimension. La construction s'arrête lorsqu'un nombre de points minimal, correspondant à une taille de groupe optimale, est atteint dans les feuilles de l'arbre.

Chaque feuille de l'arbre représente alors un faisceau d'ombre. Pour que le faisceau ne soit pas surdimensionné, les limites extrêmes des points sont finalement calculées afin d'obtenir un rectangle ajusté. Les sommets de ce rectangle permettent de former directement les quatre rayons encadrants du faisceau. La figure IV.4 montre la formation des faisceaux d'ombre après leur réorganisation.



(a) Réutilisation des groupes primaires. (b) Réorganisation des rayons.

FIG. IV.4 – Représentation des faisceaux d'ombre en fausses couleurs selon la réutilisation des groupes prrimaires (a), ou la réorganisaiton des rayons (b).

1.3.2 Résultats

Des expérimentations de la réorganisation des rayons d'ombre ont été réalisées avec l'arbre BVH. Le tableau IV.1 compare les résultats de différents regroupements de rayons selon les nombres d'étapes de traversée, les nombres de calculs d'intersection et les angles d'ouverture des faisceaux. On constate que les groupes de 64 rayons, issus de la simple réutilisation des groupes primaires, diminuent efficacement le nombre d'étapes de traversée par rapport aux rayons SIMD (entre 9% et 22%). Par contre, le nombre de calculs d'intersection n'est pas toujours meilleur. Dans les pires cas, mesurés pour les modèles *Fairy* et *Soda Hall*, il augmente d'un facteur supérieur à x1,3, mais est accompagné d'une réduction très efficace du nombre d'étapes de traversée, supérieure à x10.

Modèles	# rayons	4	64	64^{reorg}
Conference	# traversées	9,9	1,7 (17%)	0,9 (9%)
	# intersections	1,2	1,3 (111%)	1,1 (88%)
	angle moyen	-	3,4°	0,9°
Fairy	# traversées	22,0	1,9 (9%)	1,5 (7%)
	# intersections	2,0	2,6 (133%)	2,3 (116%)
	angle moyen	-	0,3°	0,1°
Soda Hall	# traversées	14,4	1,2 (9%)	1,1 (7%)
	# intersections	0,8	1,2 (154%)	0,9 (116%)
	angle moyen	-	0,8°	0,3°
Sully	# traversées	8,4	0,8 (10%)	0,5 (7%)
	# intersections	0,9	0,7 (86%)	0,5 (62%)
	angle moyen	-	0,3°	0,2°
Villa Digy	# traversées	16,3	3,6 (22%)	1,9 (11%)
	# intersections	0,8	0,7 (86%)	0,6 (76%)
	angle moyen	-	7,0°	1,4°

TAB. IV.1 – Données statistiques issues de l'exécution des rayons d'ombrage avec l'arbre BVH, selon l'utilisation de la technique de réorganisation, pour un rendu de 1024^2 pixels. Les modèles *Fairy* et *Soda Hall* comportent deux sources lumineuses et les autres modèles qu'une seule. Les nombres d'étapes de traversée et de calculs d'intersection sont exprimés en millions. La réorganisation des points d'ombrage permet de réduire le nombre d'étapes de traversée par un facteur 2 quand l'angle moyen d'ouverture des faisceaux d'ombre est supérieur à 1 degré (modèles *Conference* et *Villa Digy*).

La réorganisation des rayons d'ombre permet de réduire significativement aussi bien le nombre d'étapes de traversées que le nombre de calculs d'intersection par rapport aux groupes de rayons. Cette réduction provient d'une meilleure localité spatiale des rayons comme le prouve la diminution des angles d'ouverture des faisceaux. On peut remarquer que le nombre d'étapes de traversée est réduit de moitié pour les modèles *Conference* et *Villa Digy* qui ont une ouverture initiale supérieure à 1 degré.

Modèles	# rayons ombrages	4	64	64^{reorg}			64/4	$64^{reorg}/4$
Confererence	1 048	394	155	142	(26	+ 116)	x2,55	**x2,78**
Fairy	2 059	880	296	299	(49	+ 250)	**x2,97**	x2,94
Soda Hall	2 032	523	169	195	(57	+ 138)	**x3,09**	x2,68
Sully	724	334	93	93	(22	+ 71)	**x3,60**	**x3,60**
Villa Digy	1 034	615	220	169	(30	+ 139)	x2,80	**x3,63**

TAB. IV.2 – Temps d'exécution (en ms) des rayons d'ombrage, pour un rendu de 1024^2 avec l'arbre BVH, selon différents types de regroupements de rayons et l'utilisation de la réorganisation. Les groupes de 64 rayons issus de la réutilisation des groupes primaires permettent d'avoir un facteur d'accélération supérieur à x2,55 par rapport aux rayons SIMD. Ces performances sont améliorées par la réorganisation des points d'ombrage quand l'angle d'ouverture moyen des faisceaux est supérieur à 1 degré.

En observant le tableau IV.2, on peut remarquer que la réutilisation des groupes de rayons primaires apporte un facteur d'accélération supérieur à x2,55 par rapport aux rayons SIMD. Contrairement aux données statistiques, l'exécution des groupes de rayons réorganisés n'est pas toujours profitable. Il n'y a pas de gain pour le modèle *Fairy*, le gain est négligeable pour les modèles *Fairy* et *Sully*, et intéressant pour les modèles *Conference* et *Villa Digy*. On peut cependant remarquer que ces deux derniers modèles présentent une ouverture angulaire moyenne du faisceau particulièrement importante, et que la réorganisation des rayons permet alors une réelle diminution du nombre d'étapes de traversées, qui se traduit par de meilleurs temps de calculs. Pour les autres modèles, des réductions significatives mais moins prononcées ne sont pas compensées par le temps additionnel pris par la réorganisation des rayons. En ne considérant que les temps d'exécution des requêtes d'intersection sans la réorganisation, on constate que les performances sont toujours meilleures.

Bien que la réorganisation des rayons ne puisse pas être appliquée à coup sûr, cette technique peut être réservée aux faisceaux les moins cohérents. Cependant, comme la réorganisation des rayons ne peut être efficace que si l'ensemble des points d'ombrage est important, la réorganisation d'un faisceau seul n'a pas de sens. Une variation possible serait d'analyser les angles d'ouverture de plusieurs faisceaux proches, issus par exemple d'un découpage de l'image en rectangles, et de ne réorganiser ce sous-ensemble que si au moins un des faisceaux présente un angle d'ouverture supérieur à un angle donné.

1.4 Mémorisation d'une liste de points d'entrée

1.4.1 Principe

Les méthodes précédentes se sont concentrées sur l'exploitation instantanée des faisceaux d'ombre. Cependant, en considérant le cas où une source ponctuelle demeure immobile durant plusieurs générations de rayons, alors on peut remarquer que

des rayons d'ombres de la génération courante vont parcourir des chemins communs aux rayons de la génération précédente. Il est alors intéressant d'exploiter cette cohérence temporelle en mémorisant le trajet réalisé par des faisceaux d'ombre dans le but de le réutiliser un grand nombre de fois. En dissociant le parcours du faisceau du parcours des rayons, la requête d'intervisibilité d'un groupe de rayons implémentée avec le kd-Tree 3 à la page 62 permet de mémoriser le sous-ensemble de noeuds restant à parcourir pour tous les rayons qui passeraient dans ce faisceau. Une implémentation de cette stratégie a été mise en place pour calculer et mémoriser les trajets de faisceaux d'ombre.

De manière similaire à la technique de réorganisation des rayons, un cube virtuel est positionné autour d'une source ponctuelle. Chaque face de ce cube comporte non plus un arbre binaire, mais une série de $N_{grilles}$ grilles régulières. Chaque cellule de ces grilles permet de délimiter un faisceau d'ombre originaire de la source ponctuelle et passant par ses quatre coins ; elle mémorise alors le sous ensemble de noeuds issu de la propagation du faisceau la caractérisant.

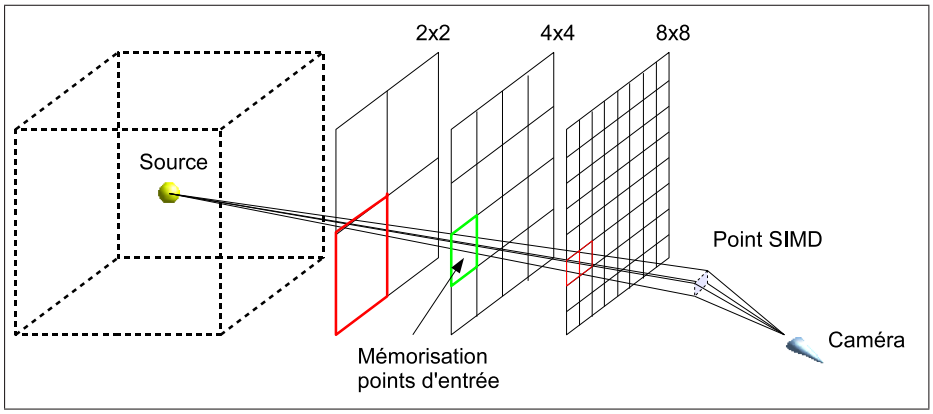


FIG. IV.5 – Mémorisation des points d'entrée du faisceau d'ombre dans la grille la plus fine.

Pour exploiter ce cache de noeuds, chaque point d'impact issu des rayons primaires est considéré indépendamment, la notion de groupe n'importe plus. Le point est projeté, selon la même technique que celle utilisée pour la réorganisation des rayons d'ombres, sur une des faces du cube. Comme l'illustre la figure IV.5, chaque face comporte $N_{grilles}$ grilles dont la grille la plus fine est d'une résolution R^2_{max}. Les $N_{grilles}-1$ grilles restantes ont des résolutions de plus en plus petites divisées par deux successivement. Cette topologie en puissance de 2 est volontairement choisie pour trouver très rapidement la cellule la plus fine pouvant contenir le point d'impact. En effet, les points d'impact sont de type SIMD, il est alors nécessaire que les quatre points simples soient situés dans la même cellule. Pour que le sous-ensemble de noeuds soit le mieux adapté, seule la cellule la plus fine pouvant contenir les quatre points est considérée. La figure IV.6 illustre comment récupérer la cellule pour un rayon d'ombre donné, et le tableau IV.3 montre l'occupation mémoire et le nombre d'étapes de traversée selon différentes variations des paramètres $N_{grilles}$ et R_{max}.

	$R_{max} = 64$		$R_{max} = 128$		$R_{max} = 256$		$R_{max} = 512$		$R_{max} = 1024$	
$N_{grilles}$	#T	Ko	#T	Ko	#T	Mo	#T	Mo	#T	Mo
1	3,6	47	3,7	173	4,4	0,6	5,7	1,7	6,7	2,6
2	3,3	59	3,2	219	3,5	0,7	4,3	2,3	5,6	4,3
4	3,2	64	2,8	235	2,7	0,8	2,8	2,5	4,3	5,1
8	-	-	-	-	**2,3**	**0,8**	2,3	2,5	2,4	5,2

TAB. IV.3 – Evolution du nombre d'étapes de traversée (en millions) et de l'occupation mémoire des multiples-grilles, lors de la mémorisation des points d'entrée, pour un rendu de 1024^2 pixels avec le modèle *Conference*. En augmentant la résolution R_{max} des grilles ainsi que le nombre de grilles $N_{grilles}$, le nombre d'étapes de traversée diminue en raison d'une meilleure cohérence des faisceaux, et l'occupation mémoire augmente en raison d'un nombre plus important de cellules créées. Un compromis doit donc être trouvé entre efficacité et occupation mémoire, qui est dans ce cas obtenu pour $R_{max} = 256$ et $N_{grilles} = 8$.

```
GridCell *cell = NULL;
int l, ml;
//Projection 2D
const __m128 pu4 = _mul_ps(r4->d4[ku], r4->id4[kw]),
const __m128 pv4 = _mul_ps(r4->d4[kv], r4->id4[kw]);
//Discretisation selon la resolution de la plus grande grille
const __m128i hr4 = _set1_ps(0.5f * RMAX);
const __m128i xi4 = _cvttps_epi32(_mul_ps(_add_ps(pu4, ONE4), hr4));
const __m128i yi4 = _cvttps_epi32(_mul_ps(_add_ps(pv4, ONE4), hr4));
//Comparaison des quatres points selon x et y
const __m128i cmp_xyi4 = _or_si128(
  _xor_si128(_shuffle_epi32(xi4, 0), xi4),
  _xor_si128(_shuffle_epi32(yi4, 0), yi4));
const int cmp_xy = cmp_xyi4[1] | cmp_xyi4[2] | cmp_xyi4[3];
//Aucune correspondance dans la grille la plus large ?
if (cmp_xy & (0xFFFFFFFF << (NG-1))) return NULL;
//Boucle jusqu'à atteindre la plus petite resolution
for (l = NG-1, ml = 1 << (NG-2); l && !(cmp_xy & ml); l--, ml >>= 1);
//Recuperation de la cellule commune dans la grille la plus fine trouvée
cell = get_cell(l, xi4[0] >> l, yi4[0] >> l);
//Calcul paresseux de la liste des points d'entrée pour cette cellule
if (!is_computed(cell)) compute_cell(cell);
return cell;
```

FIG. IV.6 – Fonction de recherche de la cellule la plus adaptée pour un rayon d'ombre. Cette cellule doit être la plus petite contenant les quatre rayons simples.

On peut remarquer que la fonction *compute_cell* chargée de calculer le faisceau d'ombre pour une cellule donnée n'est appelée que si au moins un rayon la traverse. Ceci permet de limiter de manière conséquente le nombre de calculs de faisceaux dans le cas où l'ensemble des points d'impacts est très localisé par rapport à la source ponctuelle. Pour trouver la cellule la plus adaptée, les coordonnées projetées p_u et p_v sont converties en entiers x et y correspondant à des indices d'accès à la cellule de la grille la plus fine. Par comparaison et combinaison binaires, la concordance des quatre couples d'indices entre eux est codée dans la variable *cmp_xy*. La position

du premier bit à 1 dans le sens poids fort vers poids faible permet de trouver le meilleur niveau de grille. Par exemple, une valeur binaire de 00000010 pour le codage de *cmp_xy* indique une incohérence à partir du deuxième niveau, et donc que le troisième niveau sera choisi.

1.4.2 Résultats

La mémorisation des points d'entrée a été expérimentée avec le kd-Tree pour différentes tailles de groupes de rayons comme le présentent les tableaux IV.4 et IV.5. Les paramètres N_{grille} et R_{max} ont été déterminés pour chaque modèle de manière à offrir le meilleur compromis entre temps de calculs et occupation mémoire. Ce paramétrage mérite d'être automatisé en considérant par exemple la distance de la source lumineuse par rapport au modèle. D'une manière générale, avec N_{grille} défini à 8 et R_{max} compris entre 256 et 1024, les meilleurs résultats sont obtenus. En considérant le nombre d'étapes de traversée, les groupes de 64 rayons permettent une réduction de cette quantité de moitié par rapport aux rayons SIMD. En mémorisant des points d'entrée, et en les utilisant avec des rayons SIMD, le nombre d'étapes de traversée peut être réduit jusqu'à un tiers. Cette amélioration s'explique en partie par le nombre d'étapes de traversée évitées par la recherche de points d'entrée précalculée, mais aussi par l'utilisation de points d'entrée mieux adaptés aux trajets réels des rayons (voir la figure IV.5).

Modèles	4	64		4^{mem}	
Conference	7,0	3,2	(46%)	2,3	(34%)
Fairy	17,8	9,7	(54%)	9,5	(53%)
Soda Hall	13,6	7,1	(52%)	5,8	(43%)
Sully	4,3	2,1	(50%)	1,8	(43%)
Villa Digy	9,3	4,8	(51%)	3,2	(35%)

TAB. IV.4 – Nombre d'étapes de traversée (en millions) lors de l'exécution des rayons d'ombrage avec le kd-Tree, selon la mémorisation des points d'entrée, pour un rendu de 1024^2 pixels. L'exploitation des points d'entrée pour des rayons SIMD ajoute, au premier niveau de réduction apporté par les groupes de 64 rayons, une amélioration significative en divisant jusqu'à un facteur 3 le nombre d'étapes de traversée par rapport aux rayons SIMD.

En terme de performances, les groupes de 64 rayons d'ombre permettent d'apporter un facteur d'accélération allant jusqu'à un facteur x1,32 pour le modèle *Conference*. En considérant l'utilisation des points d'entrée mémorisés, on peut distinguer deux résultats. En première remarque, le gain apporté, par rapport aux groupes de 64 rayons, est relativement faible et légèrement défavorable pour le modèle *Sully*. Ce gain est d'une part limité par le temps d'accès aux grilles qui est de l'ordre de 20 ms pour 1 million de rayons, et d'autre part pénalisé par un manque de localité temporelle. En effet, le fait de ne pas enchaîner directement la recherche des

points d'entrée et leur traversée doit certainement nécessiter un rechargement de la mémoire cache.

Modèles	# rayons ombrages	4	64	4^{mem}	64/4	4^{mem}/4
Confererence	1 048	190	143	142 (19 + 123)	x1,32	**x1,34**
Fairy	2 059	456	422	418 (31 + 387)	x1,08	**x1,09**
Soda Hall	2 032	289	282	273 (34 + 240)	x1,02	**x1,06**
Sully	724	107	92	99 (11 + 88)	**x1,17**	x1,08
Villa Digy	1 034	211	164	153 (19 + 133)	x1,29	**x1,38**

TAB. IV.5 – Temps d'exécution (en ms) des rayons d'ombrage, pour un rendu de 1024^2 avec le kd-Tree, selon différents types de regroupements de rayons et la mémorisation des points d'entrée. Les groupes de 64 rayons et les rayons SIMD exploitant les points d'entrée mémorisés ont un facteur d'accélération quasiment équivalent par rapport aux rayons SIMD classiques.

En seconde remarque, on peut comparer la stratégie de mémorisation des points d'entrée par rapport aux rayons SIMD. Dans ce cas de figure, l'accélération apportée est quasiment équivalente à celle des groupes de 64 rayons. Ces résultats rendent la stratégie intéressante car elle permet de porter la performance de rayons SIMD à celle des groupes de rayons, en récupérant simplement pour chaque point d'ombrage les points d'entrée mémorisés. En d'autres termes, il n'est pas nécessaire de construire un faisceau, ni de disposer d'un ensemble de rayons SIMD cohérent, pour bénéficier d'un facteur d'accélération équivalent à ceux des groupes de rayons. De plus, cette technique peut être utilisée aussi bien pour des rayons SIMD que pour des rayons simples, et aussi bien pour des points d'ombrage localisés qu'isolés. Le mécanisme général s'intègre de manière transparente pour chaque source ponctuelle dont on souhaite rapidement déterminer l'intervisibilité. Bien sûr, l'occupation mémoire est à prendre en compte, mais on peut raisonnablement utiliser cette approche pour plusieurs dizaines de sources ponctuelles.

1.5 Discussion

Dans cette section, nous avons présenté plusieurs expérimentations pour accélérer les calculs d'intervisibilité à origine commune, via l'utilisation de groupes de rayons. Nous avons retenu comme cas de test l'ombrage de sources ponctuelles, qui est un type de calcul très sollicité en synthèse d'images par lancer de rayons. Nous avons montré dans un premier temps comment réutiliser la cohérence issue d'un groupe de rayons primaires en reconstruisant rapidement le faisceau d'ombrage par l'utilisation de la boîte englobante des points à évaluer. Ces groupes de rayons apportent alors un facteur d'accélération très intéressant variable selon l'utilisation du kd-Tree ou de l'arbre BVH. Dans un second temps, nous avons cherché à réorganiser les points d'ombrage pour améliorer la cohérence des groupes de rayons, et enfin nous avons montré comment réutiliser la recherche des points d'entrée dans le kd-Tree.

La réorganisation des rayons d'ombrage montre une réduction des angles d'ouvertures des faisceaux qui se traduit, si l'on considère l'arbre BVH, par une réduction du nombre d'étapes de traversée et du nombre de calculs d'intersection. La performance n'est cependant améliorée que pour les modèles dont les faisceaux ont un angle d'ouverture supérieur à 1 degré. Pour une exploitation immédiate et efficace, cette technique doit être utilisée avec un garde-fou par rapport aux seuls faisceaux incohérents. Nous pensons que le temps de réorganisation des rayons n'est pas optimal et que quelques efforts d'optimisations additionnels permettraient de réduire son temps de calcul de moitié. De plus, si l'on considère seulement les temps d'exécution des groupes réorganisés, la technique est toujours avantageuse.

La mémorisation des points d'entrée avec le kd-Tree permet, lors de l'exécution de rayons d'ombrage, de réutiliser plusieurs fois des sous-ensemble de noeuds restant à visiter (voir la section 3.5 à la page 70). Cette technique permet aux rayons SIMD exploitant ces points d'entrée mémorisés, d'avoir un facteur d'accélération équivalent aux groupes de rayons. Cela permet d'apporter par ailleurs de la performance pour des points d'ombrages isolés et nous pensons, bien que nous ne l'avons pas expérimenté, que les rayons simples peuvent également bénéficier d'un facteur d'accélération intéressant.

La technique de mémorisation des points d'entrée exploite des grilles disposées autour des sources ponctuelles. Ces structures limitent, par leur occupation mémoire qui est de l'ordre du Mo, cette technique à un nombre raisonnable de quelques dizaines de sources ponctuelles. On peut noter qu'il est aisé d'étendre cette structure en ajoutant la référence du dernier triangle ombrant dans chaque cellule pour bénéficier d'un cache de triangle directionnel [28]. Si l'on s'autorise à créer des grilles très fines, on peut envisager de vérifier la présence de surfaces qui occultent complètement une cellule par la technique inverse de shaft-culling (voir la section 2.3 à la page 60) ; dans les cas favorables tout calcul d'intersection serait alors évité.

Le calcul de l'occlusion ambiante est un autre cas de test de l'intervisibilité à origine commune. Les expérimentations pour accélérer les groupes de rayons pour ce type de d'intervisibilité est présenté dans la section suivante.

2 Occlusion ambiante

2.1 Problématique

A l'opposé du calcul d'intervisibilité entre un point et une source ponctuelle, l'occlusion ambiante évalue, d'une manière générale, l'accessibilité de la lumière en un point. Son utilisation est dérivée de la notion "d'obscurance" proposée dans les travaux de Zhukov *et al.* [86] dont l'idée initiale était d'apporter un modèle d'illumination directe, plus réaliste que le traditionnel modèle de Phong [56], mais moins complexe que les calculs d'illumination globale. L'obscurance w est définie pour un

point x et une direction n comme suit :

$$w(x,n) = \frac{1}{\pi}\int_{\Omega} \rho(d(x,w))\lfloor n.\omega \rfloor d\omega$$

Dans la fonction w, d représente la distance du premier impact pour un rayon ayant pour origine x et direction w. ρ est une fonction qui normalise la distance d entre 0 et 1. L'occlusion ambiante oa est dérivée de l'obscurance sous la forme :

$$oa(x,n) = \frac{1}{\pi}\int_{\Omega} iv(x,w,\sigma)\lfloor n.\omega \rfloor d\omega$$

Dans cette reformulation, la notion de distance est remplacée par la requête d'intervisibilité binaire iv. Il est important de noter que cette requête est limitée à une distance maximale σ, introduite pour l'utilisation de l'occlusion ambiante dans la production cinématographique [45], afin d'atténuer la lumière ambiante en fonction de la présence d'objets proches.

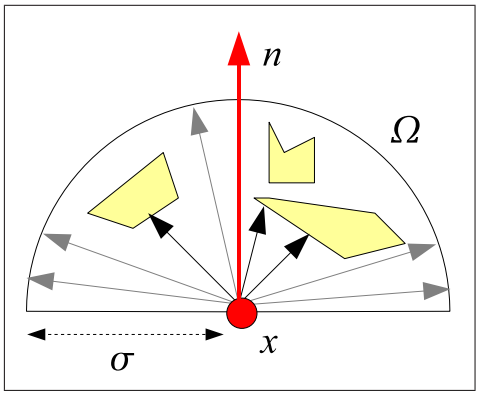


FIG. IV.7 – Evaluation hémisphérique de l'occlusion ambiante en un point.

La résolution de l'occlusion ambiante consiste à intégrer l'intervisibilité de toutes les surfaces, par rapport à x, situées dans un champ hémisphérique autour de n (voir la figure IV.7). Pour cela, il est possible d'échantillonner selon la méthode de Monte Carlo cet hémisphère avec des requêtes d'intervisibilité à portée limitée ; du nombre de ces requêtes dépend la précision obtenue. Au delà du calcul de l'occlusion ambiante, ce type d'intervisibilité hémisphérique à portée limitée est très souvent utilisée notamment dans les calculs d'illumination globale.

La combinatoire, entre le nombre de points dont l'illumination doit être déterminée et le nombre de requêtes d'intervisibilité utilisées pour la précision souhaitée, est toujours très grande. Il est donc intéressant d'optimiser ces requêtes en exploitant la cohérence spatiale. Contrairement à l'approche des sources ponctuelles dans laquelle on exploite essentiellement une cohérence de directions, on a dans ce cas une cohérence de localisations : tous les points d'origines et destinations (issus du bornage de la distance de recherche) des requêtes sont dans un espace restreint.

Cette section présente alors différentes expérimentations afin d'accélérer le calcul de l'occlusion ambiante en exploitant la localité spatiale des calculs. Dans un premier

temps, l'occlusion ambiante n'est considérée que comme un cas de test pour valider un ensemble plus large d'utilisations potentielles. L'accélération des calculs d'intervisibilité hémisphérique à portée limitée est particulièrement étudiée avec différentes stratégies de regroupement. Ensuite, une méthode de détection de zones absentes de surfaces est présentée afin d'éviter tout calcul d'intervisibilité dans un périmètre donné. Enfin, le cas particulier de l'occlusion ambiante est considéré pour proposer une technique d'approximation rapide.

2.2 Calcul de l'intégrale en un point

Le calcul de l'occlusion ambiante en un point x nécessite d'exécuter un nombre N_{oa} de requêtes d'intervisibilité dans différentes directions, de manière à couvrir l'hémisphère de visibilité. Etant donné le fort angle d'ouverture entre les rayons, différents types de regroupements ont été expérimentés de manière à trouver le meilleur nombre de rayons par groupe en fonction de N_{oa}. Les différentes tailles de groupes N_{groupe} envisagées sont :

- 1 avec des rayons simples,
- 4 avec des rayons SIMD et,
- de 16 à 64 pour des groupes de rayons SIMD.

De manière à pouvoir simplement regrouper les rayons, la structure d'un hémicube, centré sur le point x et orienté selon la normale n, est utilisée pour générer les rayons. La face centrale de cet hémicube comporte une grille de résolution ${R_{grille}}^2$, déterminée de manière à approcher au mieux le nombre de requête N_{oa} tel que :

$$R_{Grille} \backsimeq \sqrt{\frac{N_{oa}}{3}}, \text{ pour } R_{Grille} > 0 \text{ et } R_{Grille} \; mod \; \sqrt{N_{groupe}} = 0$$

Chaque cellule permet de générer aléatoirement un rayon. Elles sont regroupées, pour une même face, par groupe de quatre pour constituer des rayons SIMD, qui sont eux même regroupés pour construire des groupes de rayons plus grands. Les rayons encadrants des faisceaux ne sont pas calculés mais déduits directement des coins des cellules extrêmes.

Le tableau IV.6 montre les performances brutes des différents regroupements de rayons selon la densité d'échantillonnage de l'occlusion ambiante. Les groupes de 64 rayons sont construits de manière à générer d'abord et si possible des ensembles de 64 rayons, puis de 16 et enfin de 4 pour les rayons restants. Les performances montrent que les rayons SIMD sont peu sensibles à l'angle d'ouverture des faisceaux, car même avec 48 rayons utilisés pour l'échantillonnage hémisphérique, le facteur d'accélération reste de x2,9 par rapport aux rayons simples. Les groupes de rayons sont davantage pénalisés par cet écart angulaire car les évaluations de moins de 200 rayons apportent peu ou pas d'accélération. En augmentant le nombre de rayons jusqu'à 3072, l'accélération des groupes de rayons devient très intéressante avec un facteur supérieur à x3,9 par rapport aux rayons simples. Bien qu'un tel nombre de rayons soit surdimensionné pour échantillonner l'occlusion ambiante, ce gain peut

être transposé à toute technique plus réaliste qui nécessite d'évaluer de manière dense les intervisibilités autour d'un point.

Pour un nombre de rayons moins important mais plus adapté au cas de l'occlusion ambiante, le regroupement de rayons n'est pas une technique très efficace, bien qu'il existe de la cohérence. La section suivante montre comment exploiter la cohérence de ces rayons dans le cas particulier où l'évaluation de l'occlusion ambiante est réalisée dans un espace vide de surfaces.

Modèles	R_{grille}	1	4	64	4/1	64/1
Conference	4 (48)	0,74	2,17	2,15	x2,9	x2,9
	8 (192)	0,78	2,36	2,92	x3,0	x3,7
	16 (768)	0,82	2,36	4,07	x2,9	x5,0
	32 (3072)	0,84	2,44	5,01	x2,9	x6,0
Fairy	4 (48)	0,52	1,37	1,38	x2,6	x2,6
	8 (192)	0,56	1,53	1,62	x2,7	x2,9
	16 (768)	0,60	1,65	2,10	x2,8	x3,5
	32 (3072)	0,63	1,79	2,43	x2,8	x3,9

TAB. IV.6 – Performances du calcul de l'intégrale avec l'arbre BVH, en millions de rayons par seconde, selon la taille des groupes de rayons. Les rayons SIMD apportent un facteur d'accélération par rapport aux rayons simples assez constant quelle que soit la densité d'échantillonnage, puisqu'il est compris entre x2,6 et x3,0. Les groupes de rayons apportent également un facteur d'accélération qui devient intéressant pour des évaluations de plus de 200 rayons.

2.3 Détection des zones vides

2.3.1 Boîte de visibilité

Le paragraphe précédent a présenté des méthodes afin d'exploiter la cohérence pour le groupe de rayons issu de l'évaluation d'un point. L'intérêt est désormais porté sur la cohérence entre deux points proches. Bien qu'il soit possible d'espacer les points de calculs, car l'occlusion ambiante est un phénomène qui évolue lentement, seule l'approche classique qui consiste à calculer cette valeur en tout point est considérée pour l'instant.

Par définition, les requêtes d'intervisibilité pour l'occlusion ambiante sont limitées à une distance maximale σ par rapport au point à évaluer. Cette distance peut être modélisée par le rayon de l'hémisphère de visibilité ; toute surface située en dehors de cet hémisphère n'est pas à considérer. Entre deux points proches, le différentiel de volume de visibilité entre les hémisphères respectifs est faible. Pour profiter de ce partage de l'espace, il est intéressant de mutualiser la zone de visibilité pour un ensemble de points afin de ne vérifier cette visibilité qu'une seule fois (voir la figure IV.8). Deux cas de figures peuvent subvenir : la zone est vide de toute surface, ou contient au moins une surface. Dans le second cas, il est envisageable de récupérer

l'ensemble des surfaces et de ne calculer que des requêtes d'intersections rayons/-triangles évitant ainsi des parcours d'arbre. Cependant le nombre de surfaces peut être important, et le bénéfice de la complexité logarithmique de recherche perdu. Bien qu'il puisse exister un seuil en dessous duquel le calcul de toutes les intersections peut être favorable, seul le premier cas des zones vides a été expérimenté.

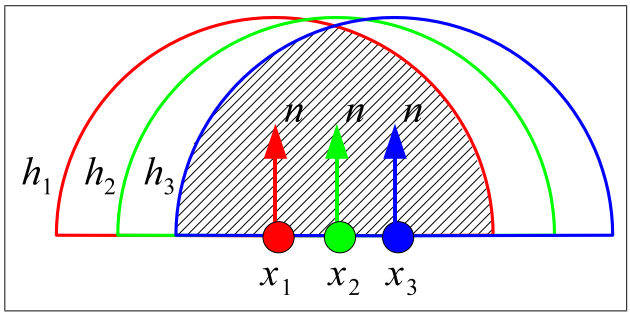


FIG. IV.8 – Volume de visibilité partagé par les hémisphères de trois points proches.

Afin de détecter les zones vides, une requête spéciale de visibilité dans une boîte a été développée avec l'arbre BVH. Cette requête peut tout aussi bien être implémentée avec le kd-Tree, mais l'arbre BVH génère par nature la construction d'espaces vides plus grands. Les paramètres de cette requête sont une boîte et une équation de plan P_v. La boîte traduit le volume dans lequel la présence de surface est recherchée, et sa taille est fonction de σ. P_v est le plan qui passe par un point x de normale n, il sert à caractériser le demi-espace visible depuis x.

Pour déterminer la visibilité de la boîte B_v, celle-ci est propagée dans l'arbre BVH en fonction de son intersection avec les noeuds. En cas d'intersection, deux tests additionnels sont réalisés afin de vérifier sa position par rapport au plan P_v selon la technique expliquée dans la section 4.3.1 à la page 78. Si le noeud est situé complètement en-dessous de P_v alors il est rejeté, et la propagation de la boîte de visibilité est poursuivie. Si le noeud est situé complètement au-dessus de P_v alors la propagation est arrêtée et la requête renvoie un cas d'occlusion. Pour chaque noeud visible, la boîte englobante de chaque triangle est testée vis à vis de la boîte de visibilité B_v, ainsi que ses sommets par rapport au plan P_v. Si une intersection est trouvée, alors la requête de visibilité est interrompue immédiatement, sinon elle continue jusqu'à la traversée complète de l'arbre. En considérant les sommets extrêmes $e0$ et $e1$ de la boîte englobante B_e d'un noeud ou d'un triangle, le test de rejet avec la boîte de visibilité B_v, dont les sommets extrêmes sont $v0$ et $v1$, est le suivant :

$$\begin{aligned} Rejet(Bv, Be) = \\ & e0_x > v1_x \text{ ou } v0_x > e1_x \text{ ou} \\ & e0_y > v1_y \text{ ou } v0_y > e1_y \text{ ou} \\ & e0_z > v1_z \text{ ou } v0_z > e1_z \text{ ou} \\ & n.ef + k < 0 \end{aligned}$$

Le dernier test correspond au rejet de la boîte Be par son sommet ef, le plus loin par

rapport au plan P_v d'équation $n.p + k = 0$. Les différents cas de rejet sont illustrés dans la figure IV.9.

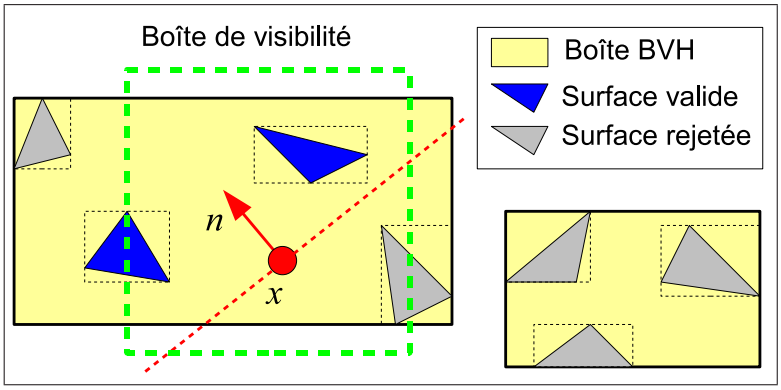


FIG. IV.9 – Principe du rejet entre une boîte de visibilité et une boîte englobante (noeud ou surface).

2.3.2 Mémorisation des zones vides dans un octree

Disposant d'une requête de visibilité dans une boîte, il est nécessaire de réfléchir à l'utilisation de cette boîte : quelle dimension et quel positionnement pour un ensemble de points donnés. Plutôt que de définir ces paramètres à partir d'un ensemble de points, une approche inverse a été choisie en construisant les boîtes de visibilité, a priori, selon la structure d'un arbre octal (voir la figure IV.10). Cette approche a l'avantage de réutiliser le résultat des requêtes de visibilité pour tous points tant que la géométrie n'évolue pas.

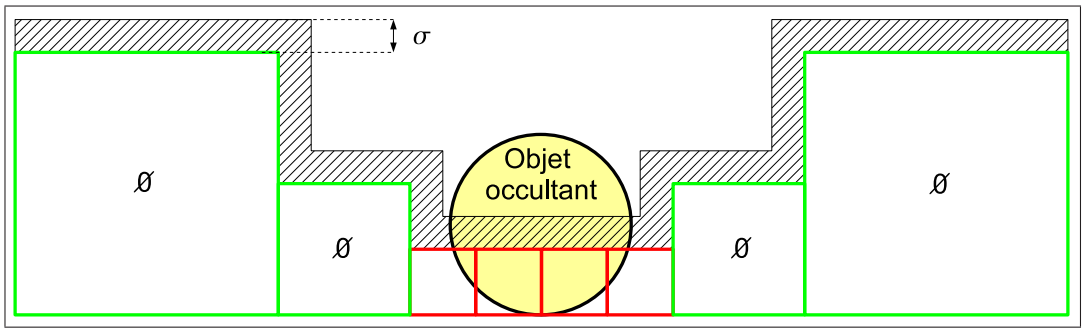


FIG. IV.10 – Discrétisation des volumes vides selon une distance maximale de recherche σ.

Pour initier la construction de l'arbre octal, un cube représentant sa racine est déterminé de telle manière qu'il englobe toute la géométrie. L'arbre est ensuite raffiné de façon paresseuse en fonction des points d'occlusion ambiante à évaluer. Pour chaque point, l'arbre octal est parcouru récursivement jusqu'à la feuille le contenant. A partir de celle-ci, une requête de visibilité est exécutée en utilisant, pour paramètres, la dimension de la feuille étendu de 2σ, et l'équation de plan passant par le

point. L'extension de la dimension permet de s'assurer que tout point situé dans la feuille se situe dans un rayon de visibilité $\leq \sigma$. En cas d'absence de surface, le parcours de l'arbre octal est interrompu et l'occlusion ambiante est considérée comme nulle. Dans le cas inverse, la feuille est transformée en noeud et subdivisée en huit sous-feuilles. L'algorithme précédent est alors répété pour la sous-feuille contenant le point. Le processus de raffinement est arrêté si la dimension d'une feuille est $< 2\sigma$. Dans ce cas, l'évaluation de l'occlusion est déterminée selon le calcul de l'intégrale présentée précédemment (voir la section 2.2 à la page 104).

Deux aménagements permettent de limiter le nombre de requêtes de visibilité pour une boîte par rapport à l'approche générale présentée. D'une part, sans considérer le plan P_v comme élément de rejet de surfaces, on peut remarquer que si toutes les surfaces présentes dans une boîte de visibilité B_v sont coplanaires selon un plan P_c, alors par définition aucune surface ne peut être située au dessus de P_c. En considérant dans ce cas que tous les points de calculs de l'occlusion ambiante sont positionnés sur des surfaces, alors tous les points appartiennent forcément à P_c. Par conséquent, si toutes les surfaces d'une boîte de visibilité sont coplanaires alors il est certain que tout calcul d'occlusion ambiante sera nul. Pour mettre en place cette observation, la requête de visibilité pour une boîte est modifiée afin de distinguer les trois cas suivants (par ordre de priorité) :

- cas de *coplanarité* : présence exclusive de surfaces coplanaires ;
- cas de *vide* : absence de surfaces visibles selon B_v et P_v ;
- cas de *occlusion* : au moins une surface visible ;

Bien que les deux premiers cas permettent d'éviter le calcul d'occlusion ambiante pour un point x, le cas de *coplanarité*, mémorisé comme un état dans une feuille de l'arbre octal, permet d'éviter tout type requête de visibilité par la suite.

Le deuxième aménagement consiste en une optimisation, puisqu'il s'agit de mémoriser la dernière feuille visitée, les paramètres B_v et P_v, et le résultat de la dernière requête de visibilité. Ainsi, lors de l'évaluation du point suivant, son appartenance à la boîte B_v mémorisée est d'abord vérifiée. Dans l'affirmative, le parcours de l'arbre octal est repris directement à partir de la feuille associée, et si le nouveau point est situé sur le plan P_v mémorisé, alors le résultat de la dernière requête de visibilité est directement réutilisé.

	$\sigma = 100$	$\sigma = 50$	$\sigma = 20$
# évaluations	262	262	262
# cache même noeud	235 (90%)	227 (87%)	221 (84%)
# cache même plan	160 (61%)	118 (45%)	85 (32%)
# requêtes zones vides	75 (29%)	109 (42%)	136 (52%)
# traversées Octree	412	487	555
# traversées BVH	1 826	2 981	3 854
# intersections BVH	138	176	263

TAB. IV.7 – Données statistiques (en milliers) de la requête de détection de zones vides pour un rendu de 512^2 pixels, et une évaluation en tout point de l'occlusion ambiante. Moins de la moitié des évaluations aboutissent à une exécution réelle de la requête de visibilité grâce à l'utilisation de caches pour le dernier noeud de l'arbre octal accédé, et le dernier plan utilisé. Le cache de dernier noeud permet de limiter en moyenne à 2 étapes de traversée dans l'arbre octal par point d'évaluation. Le nombre de traversées et de calculs d'intersections de la boîte de visibilité dans l'arbre BVH augmente quand σ diminue, en raison de plus petites tailles de boîtes de visibilité.

Le tableau IV.7 présente différentes données statistiques recueillies lors de l'exécution de la fonction de détection des zones vides. La mise en place du cache d'appartenance au dernier noeud accédé de l'arbre octal, ainsi qu'au dernier plan évalué, permet de limiter le nombre effectif de détections entre 29% et 52% par rapport au nombre de points évalués. La réduction du rayon de visibilité σ génère des noeuds de plus petites tailles dans l'arbre octal, en raison du critère de subdivision, et augmente ainsi le nombre d'étapes de traversée et de calculs d'intersections.

2.3.3 Résultats

Le tableau IV.8 présente l'apport de la détection des zones vides pour l'évaluation de l'occlusion ambiante en tout point. En considérant les temps de calculs T_{oa} sans la détection des zones vides, on constate que les performances augmentent quand le rayon de visibilité σ diminue. Cette évolution est simplement due au fait qu'un rayon court s'exécute plus rapidement qu'un rayon plus long, car il génère moins d'étapes de traversée et de calculs d'intersections. En considérant le rapport $N_{oa}^{\emptyset}/N_{oa}$, on peut remarquer également que la quantité d'espace trouvé augmente quand σ diminue. Dans les meilleurs cas, plus de deux tiers des points d'évaluation sont situés dans des zones absentes de surfaces pouvant apporter de l'occlusion ambiante.

En termes de performances, l'accélération des temps de calculs est liée à la quantité d'espaces vides trouvés et à la densité d'échantillonnage. Pour le plus grand rayon de visibilité ($\sigma = 100$), les 10% d'espaces vides détectés ne suffisent pas à apporter un gain significatif. Pour le plus petit rayon de visibilité ($\sigma = 20$), les performances sont accélérées par un facteur x1,71 à x1,9. Plus la densité d'échantillonnage est élevée et plus la détection des espaces vides est profitable.

	$R_{Grille} = 4$			$R_{Grille} = 8$		
	$\sigma = 100$	$\sigma = 50$	$\sigma = 20$	$\sigma = 100$	$\sigma = 50$	$\sigma = 20$
T_{oa}	5,6	4,7	4,2	20,3	17,7	16,0
$T_{oa}^{\emptyset}$	5,5	3,6	2,4	20,0	13,0	8,4
$T_{oa}/T_{oa}^{\emptyset}$	x1,02	x1,29	x1,72	x1,01	x1,36	x1,90
N_{oa}	12,7	12,7	12,7	50,4	50,4	50,4
$N_{oa}^{\emptyset}$	11,4	6,3	3,9	45,2	25,1	15,6
$N_{oa}^{\emptyset}/N_{oa}$	90%	50%	31%	90%	50%	31%

TAB. IV.8 – Temps de calculs (en secondes), et nombres de rayons lancés (en millions), pour l'évaluation de l'occlusion ambiante en tout point pour le rendu du modèle *Soda Hall* en 512^2 pixels, avec ou sans détection des zones vides. La proportion d'espace vide trouvé est directement en rapport inverse avec le rayon de visibilité. Pour $\sigma = 20$, deux tiers des points ne sont pas évalués, ce qui résulte en une accélération des calculs entre x1,72 et x1,9 selon la densité d'échantillonnage.

Bien que la détection des espaces vides apporte un gain certain, les temps de calculs de l'occlusion ambiante restent supérieurs à la seconde si celle-ci est réalisée en tout point. Il est donc intéressant de voir comment réduire le nombre de points d'évaluation tout en gardant une approximation correcte.

2.4 Approximation de l'occlusion ambiante

2.4.1 Travaux précédents

Parce que l'occlusion ambiante est un phénomène basse fréquence, les valeurs calculées entre deux points proches varient faiblement. De nombreux travaux se sont appuyés sur cette observation pour en approximer et/ou interpoler les valeurs.

Pour parvenir au temps réel, certaines approches précalculent l'occlusion ambiante dans une grille autour d'objets rigides [44, 48]. Lors d'un rendu de type GPU, les cellules de ces grilles sont ensuite accédées pour chaque point, et une interpolation est réalisée sur les valeurs d'occlusion ambiante précalculées. Bien que ces approches permettent d'atteindre le temps réel (plus de 20 images par seconde) et le support dynamique d'objets rigides, les performances sont linéairement dépendantes du nombre d'objets différents. De plus, chaque objet nécessite un temps de précalcul variable selon la complexité de l'objet pouvant atteindre plusieurs minutes.

Une autre approche consiste à éviter le calcul de visibilité en ne calculant que des facteurs de formes [12] entre des disques orientés. Ces derniers sont positionnés sur chaque sommet de la géométrie, et leur orientation et leur taille sont fonction des surfaces partageant le sommet. Des facteurs de forme sont calculés entre chaque sommet. Pour éviter une complexité en $O(N^2)$, une approximation est réalisée pour les sommets lointains en les regroupant sur un disque plus grossier. L'avantage de cette approche est d'éviter toute requête d'intervisibilité ; cependant l'efficacité et

la précision de cette méthode restent très liées à la façon dont est modélisée la géométrie, étant donné que l'occlusion ambiante est interpolée entre les sommets. De plus, la performance temps réel est limitée sur GPU (carte graphique) à quelques dizaines de milliers de surfaces. D'autres implémentations de ce même algorithme [83] montrent que les performances sur CPU sont divisées par un facteur cinq, réduisant d'autant la taille maximale des modèles exploitables en temps réel.

Bikker [8] propose de calculer l'occlusion ambiante de manière éparse et grossière puis d'interpoler les résultats. Une grille 2D est construite sur le plan de l'image dans laquelle plusieurs pixels sont regroupés par cellule. Seuls les pixels situés aux coins des cellules sont calculés avec 16 requêtes d'intervisibilités. Les pixels situés à l'intérieur des cellules sont interpolés en fonction des valeurs aux coins si aucune discontinuité en terme de normales, de distances et de différences de valeur d'occlusion ambiante n'est trouvée. En cas de discontinuité, la grille est raffinée de manière hiérarchique jusqu'à atteindre des cellules de 2x2 pixels. Bien que cette approche soit intégrable dans un rendu interactif, le nombre de requêtes utilisées pour échantillonner l'occlusion ambiante semble faible. De plus, une approche par interpolation de l'image pose des problèmes si l'on considère des effets secondaires de réflexion ou réfraction, et ne permet pas la réutilisation des résultats pour les images suivantes, dans la mesure où la géométrie reste inchangée.

L'approche choisie pour l'approximation de l'occlusion ambiante diffère de celles présentées, et s'inspire de celle proposée par Greger [27] qui mémorise l'illumination globale dans les cellules d'une grille. Disposant d'une structure pour mémoriser les zones vides, l'arbre octal est étendu afin de mémoriser en plus l'occlusion ambiante dans les feuilles. Cette approche est présentée en suivant.

2.4.2 Mémorisation de l'occlusion ambiante

La construction de l'arbre octal présentée précédemment, est guidée par le calcul des points d'occlusion ambiante. Pour chaque point x, la feuille de l'arbre contenant ce point est retrouvée, et si aucune surface n'est visible dans un rayon autour de cette feuille, alors le calcul d'occlusion ambiante est considéré comme nul. Dans le cas inverse, l'approche précédente réalise un calcul classique (voir la section 2.2 à la page 104) pour le point donné. L'approche nouvelle est de précalculer et de mémoriser l'intervisibilité sphérique des sommets du cube représenté par la feuille. La valeur finale au point x est alors approximée en intégrant les intervisibilités, en fonction de n, pour chaque sommet, puis en effectuant une interpolation trilinéaire des différentes occlusions ambiantes obtenues.

Cependant, pour que l'interpolation ne soit pas trop grossière, un paramètre additionnel vient piloter la construction de la grille. Ce paramètre représente la distance projetée δ de N_{pixels} pixels au point x. Plus N_{pixels} est grand et plus l'approximation sera grande et les calculs accélérés. Ainsi, une feuille ne peut être exploitée que si sa dimension est $\leq \delta$; le raffinement de l'arbre octal est alors modifié afin de continuer la subdivision d'une feuille non vide tant que sa dimension est $> \delta$.

Pour éviter tout calcul inutile, l'intervisibilité sphérique n'est évaluée pour les sommets d'une feuille que si un point y est localisé. Ce calcul est réalisé de la même manière que la technique présentée dans la section 2.2 à la page 104, en remplaçant l'hémicube par un cube et en alignant ce dernier aux axes. Chaque cellule des grilles, constituant les faces de ce cube, mémorise alors une valeur binaire d'intervisibilité. Etant donné que les sommets des cubes sont partagés entre plusieurs feuilles, un système d'indexation est mis en place pour accéder au sommet selon ses coordonnées discrétisées converties en identifiant unique. Un gestionnaire de pages permet ensuite d'adresser l'ensemble des sommets possibles pour une occupation mémoire relativement faible. Le partage des sommets permet de réduire par un facteur 3 le nombre de sommets calculés.

2.4.3 Interpolation

L'interpolation de l'occlusion ambiante est réalisée sur l'ensemble des sommets visibles dans le demi-espace défini par le point x et sa normale n. Pour chaque sommet visible, il est nécessaire de calculer une valeur d'occlusion ambiante locale en fonction de n. Pour cela, une intégration doit être réalisée en cumulant toutes les intervisibilité précalculées dont la direction est située dans l'hémisphère de visibilité de x. Comme cette seule opération peut s'avérer coûteuse, l'occlusion ambiante pour n est également mémorisée dans les grilles du sommet. Chaque cellule d'une grille contient alors une information de visibilité et une information d'occlusion ambiante pour la direction discrétisée. Bien que l'information de visibilité ne nécessite qu'un seul bit de codage, l'occlusion ambiante requiert 4 octets pour un réel simple occasionnant ainsi une occupation mémoire assez conséquente.

Une optimisation limite le calcul de l'occlusion ambiante aux axes principaux et son codage sur un octet. Lors du calcul d'intervisibilité sphérique, six calculs d'occlusion ambiante sont directement effectués et mémorisés pour les six directions axiales. Etant donné que la valeur de l'occlusion ambiante est bornée entre 0 et 1, celle-ci peut être quantifiée sur 8 bits, générant une imprécision acceptable de 4.10^{-3}. Pour déduire l'occlusion ambiante d'un sommet, les signes et coefficients de sa normale n sont utilisés pour choisir et pondérer les six valeurs précalculées. Cette approximation permet de réduire considérablement l'occupation mémoire, et également d'améliorer les performances en raison d'une meilleure localité d'accès.

Seuls les sommets situés dans l'hémisphère de visibilité du point x sont utilisés. En considérant les huit sommets s_{ijk} d'une feuille, où i, j, k sont des indices d'accès comme illustré sur la figure IV.11, la fonction de visibilité u d'un sommet est déterminée telle que $u(s_{ijk}, n) = n.s_{ijk} + k > 0$. L'occlusion ambiante finale est alors obtenue par interpolation trilinéaire sur les valeurs d'occlusion ambiante aux

sommets oa_{ijk} selon les calculs suivants :

$$\begin{aligned}
\alpha &= (x_x - s000_x)/(s111_x - s000_x) \\
\beta &= (x_y - s000_y)/(s111_y - s000_y) \\
\gamma &= (x_z - s000_z)/(s111_z - s000_z) \\
c_{ijk} &= (\text{si } i > 0 \text{ alors } \alpha \text{ sinon } 1 - \alpha) \times \\
&\quad (\text{si } j > 0 \text{ alors } \beta \text{ sinon } 1 - \beta) \times \\
&\quad (\text{si } k > 0 \text{ alors } \gamma \text{ sinon } 1 - \gamma)
\end{aligned}$$

$$oa(x, n) = \frac{\sum_{ijk} u(s_{ijk}, n) \times c_{ijk} \times oa_{ijk}}{\sum_{ijk} u(s_{ijk}, n) \times c_{ijk}}$$

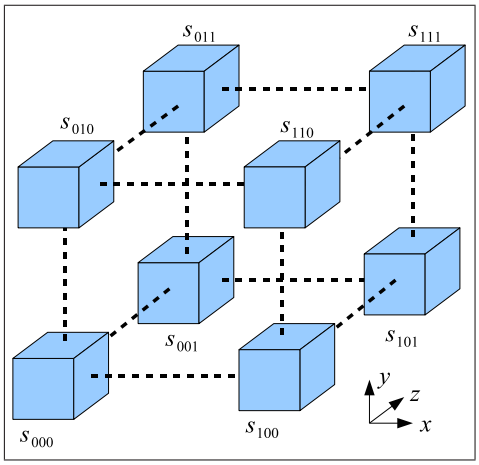


FIG. IV.11 – Disposition des sommets dans la feuille d'un arbre octal.

2.4.4 Résultats

Dans le tableau IV.9, les résultats de l'approximation ambiante sont comparés à l'évaluation en tout point, en faisant varier le paramètre N_{pixels} de 4 à 64. La première ligne reprend donc les résultats présentés dans le tableau IV.8 avec la détection des espaces vides. Les temps de calculs sont accélérés par un facteur x3 à x50 selon le niveau d'approximation défini. Ce gain de performances est naturellement lié au nombre de sommets évalués qui diminue à mesure que N_{pixel} augmente.

		$R_{Grille} = 4$			$R_{Grille} = 8$		
N_{pixels}	# sommets	$\sigma = 100$	$\sigma = 50$	$\sigma = 20$	$\sigma = 100$	$\sigma = 50$	$\sigma = 20$
1	-	5 523	3 647	2 436	20 337	17 684	16 014
8	29 913	1417	1 086	971	4 195	3 635	3 314
16	9 551	500	408	342	1 537	1 293	1 129
32	3 042	197	158	139	568	463	403
64	850	160	70	59	211	151	133

TAB. IV.9 – Temps d'initialisation (en ms) de l'approximation de l'occlusion ambiante pour le rendu en 512^2 pixels du modèle *Soda Hall*. Le premier niveau d'approximation pour $N_{pixels} = 8$ permet de réduire par un facteur 3 le temps de calculs de l'occlusion ambiante en tout point (avec détection des espaces vides). Pour des approximations plus élevées, les temps d'initialisation se limitent à quelques centaines de millisecondes.

Bien qu'il soit hors de nos propos de juger de cette qualité d'"approximation, on peut considérer, que visuellement, les paramètres $N_{pixel} = 16$ et $R_{grilles} = 8$ offrent un bon compromis entre qualité d'image et temps de calculs. Ces temps correspondent au temps de génération de la première image avec la construction de l'arbre octal et l'évaluation des sommets. Si l'on considère une animation avec ces mêmes paramètres, après un temps d'initialisation de 1,5 s, les temps de calculs pour les images suivantes diminuent significativement et restent compris entre 50 ms et 200 ms (voir la figure IV.12). La construction paresseuse de l'arbre octal permet alors d'évaluer interactivement le différentiel de sommets pour chaque image.

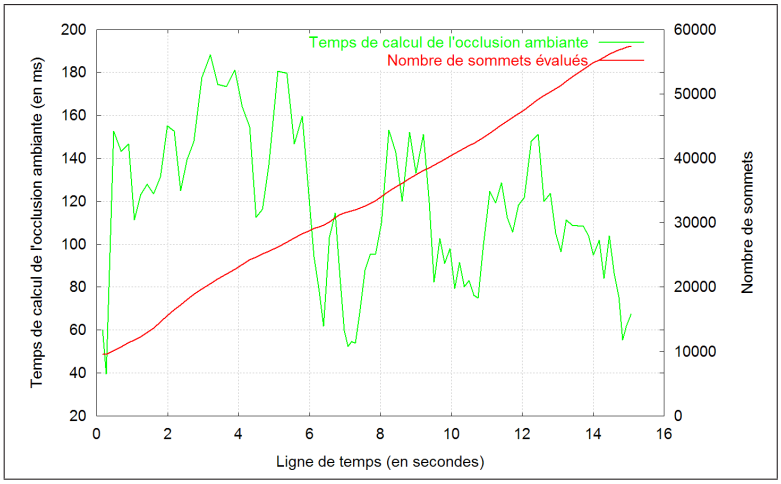


FIG. IV.12 – Temps de calcul (en ms) de l'occlusion ambiante, et nombre total de sommets calculés, au cours d'une animation de 15 secondes pour le rendu du modèle *Soda Hall* en 512^2 pixels.

2.5 Discussion

Le calcul de l'occlusion ambiante est un cas de test intéressant pour mesurer la capacité des groupes de rayons à supporter des calculs d'intervisibilité omnidirectionnels en un point. Nous avons montré que l'apport de performance ne devenait intéressant qu'à partir d'un échantillonnage de plus de 200 rayons par point d'évaluation. En dessous de cette limite, il est préférable d'utiliser des rayons SIMD ; ces derniers présentent alors un fort facteur d'accélération supérieur à x2,6 par rapport aux rayons simples, quelle que soit la densité d'échantillonnage.

En considérant l'évaluation de l'occlusion ambiante en tout point, nous avons montré qu'il est possible d'éviter tout calcul pour les zones ne comportant pas de surface située dans le rayon de visibilité. En associant d'une part la recherche de zones vides dans un volume défini par une boîte, et d'autre part le contrôle de cette recherche par la structure d'un arbre octal construit paresseusement, nous avons mesuré, dans un cas favorable mais réaliste, que deux tiers des points d'évaluation peuvent être ignorés permettant une accélération par x2 des performances. Cette approche originale, bien que différente des stratégies de regroupement des rayons, exploite les mêmes principes de factorisation des calculs pour un ensemble de rayons cohérents. De plus, cette approche peut s'utiliser en complément des groupes de rayons si l'on considère des échantillonnages plus denses.

Cependant, malgré la détection des zones vides, les temps de calculs en tout point demeurent supérieurs à la seconde pour le rendu d'une image de faible résolution (512^2). Nous avons alors modifié l'arbre octal pour mémoriser dans les sommets de ces feuilles, un calcul d'occlusion ambiante sphérique. La valeur de l'occlusion ambiante en un point est alors interpolée avec les sommets visibles. Pour contrôler l'approximation résultante de cette interpolation, les noeuds de l'arbre octal sont sélectionnés de manière à avoir une dimension inférieure à N_{pixels}. La solution générale permet, pour un niveau d'approximation correcte, d'initialiser rapidement les premières valeurs d'occlusion ambiante et de naviguer ensuite interactivement, tout en évaluant le différentiel de valeurs, à plus de 5 images/s.

Conclusion

Dans ce chapitre, nous avons traité du problème spécifique des calculs d'intervisibilité à origine commune. Deux cas de tests utilisés dans la synthèse d'image, et présentant de nombreuses différences, ont été étudiés pour exploiter au mieux cette particularité : l'ombrage direct des sources ponctuelles et l'occlusion ambiante.

Pour les sources ponctuelles, nous avons présenté en premier lieu une méthode qui permet de construire rapidement un faisceau d'ombre à origine commune, en exploitant les deux sommets extrêmes de la boîte englobante des points à ombrer. La réutilisation des points issus des intersections des groupes de rayons primaires, permet d'apporter un facteur d'accélération, par rapport aux rayons SIMD, toujours favorable mais variable selon l'utilisation du kd-Tree ou de l'arbre BVH.

Pour aller plus loin par rapport à la simple réutilisation des groupes de rayons primaires, nous avons montré comment réorganiser les points d'ombrage pour construire des faisceaux d'ombre plus cohérents. La réorganisation de ces points permet alors une réduction effective du nombre d'étapes de traversée, ainsi que du nombre de calculs d'intersection. Cependant, le temps additionnel pris par la réorganisation (qui reste perfectible), limite l'utilisation de cette technique aux faisceaux d'ombre les moins cohérents dont l'angle d'ouverture est supérieur à 1 degré.

Une approche originale a montré qu'il était possible de mémoriser les résultats de la recherche des points d'entrée avec le kd-Tree (voir la section 3 à la page 62), dans une structure multi-grilles construite autour de la source ponctuelle. Cette approche, par l'exploitation des points d'entrée mémorisés, permet alors d'amener les performances d'exécution des rayons SIMD à celles des groupes de 64 rayons, sans imposer les contraintes de cohérence, ni de quantité de points à ombrer.

Dans le cas de l'intervisibilité omnidirectionnelle à portée limitée, représentée par l'occlusion ambiante, le regroupement de rayons n'apporte un facteur d'accélération significatif que pour une densité d'échantillonnage supérieure à 200 rayons par point d'évaluation. Nous avons montré qu'il était possible, pour des densités d'échantillonnage inférieures, d'exploiter la cohérence existante entre les différents points d'évaluation par la détection de zones vides. Une requête de visibilité spécialisée est alors utilisée pour vérifier la présence de surfaces dans un volume donné représenté par une boîte. L'utilisation couplée de cette requête originale de visibilité, guidée par la construction d'un arbre octal, permet une accélération effective des performances par un facteur x2.

En considérant que l'occlusion ambiante peut être approximée sans grande perte de qualité d'image, nous avons remplacé l'évaluation en tout point de l'occlusion ambiante par les seules évaluations des sommets des feuilles d'un arbre octal. Cette approximation contrôlée par un paramètre de précision, permet alors de naviguer interactivement, à plus de 5 images/s, dans un modèle constitué de plus 2 millions de triangles.

Ce chapitre a montré quelques limitations des regroupements de rayons, dans une moindre mesure avec l'incohérence de certains faisceaux d'ombre, et dans une mesure plus importante avec l'échantillonnage omnidirectionnel en un point. De plus, les différentes techniques ont été présentées en partie avec l'arbre BVH ou avec le kd-Tree. Le chapitre suivant, analyse plus en détail, les différentes performances obtenues selon le regroupement de rayons et le type de structure de recherche utilisé dans le contexte d'un rendu en synthèse d'images exploitant aussi bien les rayons primaires, les rayons d'ombrages, les rayons secondaires et l'occlusion ambiante.

Chapitre V

Traitement des rayons divergents

Introduction

Le regroupement de rayons est une technique d'accélération de la requête d'intersection dont l'efficacité est dépendante de la cohérence des rayons. Il est difficile de déterminer a priori la limite entre cohérence et incohérence. Celle-ci dépend bien sûr de l'organisation des rayons, mais aussi de la géométrie traitée et également de la capacité de la structure de recherche à tolérer de larges groupes de rayons. Franchir cette limite peut faire basculer les performances de l'accélération à la dégradation. La dégradation peut être minime si elle se limite au temps de construction de la structure additionnelle utilisée pour traiter les groupes de rayons, ou conséquente si l'exécution du groupe de rayons génère des chemins de traversée et des calculs d'intersections plus importants que l'exécution des rayons individuels.

Une approche prudente se limiterait à n'utiliser que des rayons simples. Cependant il est intéressant de traiter au mieux tout type de rayons en tirant partie, quand cela est possible, des capacités d'accélération des groupes de rayons. Selon la nature du problème physique traité, on sait par expérience quels types de rayons sont par construction cohérents et ceux susceptibles de ne pas l'être. Ainsi dans les techniques de rendus par lancer de rayons, les rayons primaires bénéficient toujours d'une forte cohérence et les rayons d'illumination globale sont par exemple connus pour être trop divergents. Deux approches différentes sont proposées pour traiter le problème de la divergence des rayons. La première approche est empirique, et tient compte de la nature connue des rayons pour exploiter les points forts des structures d'accélération au regard des rayons simples et des groupes de rayons. La deuxième approche est une réflexion plus optimiste, qui remet en cause la limite empirique fixée entre cohérence et incohérence, et propose une méthode de regroupement automatique de rayons.

1 Couplage des structures de données

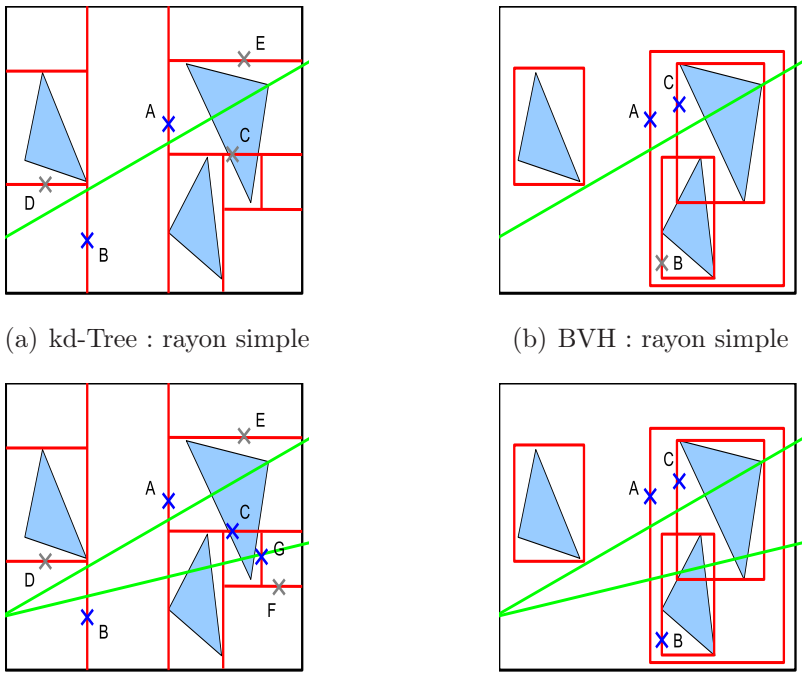
1.1 Problématique

La cohérence est un critère indispensable pour exploiter efficacement les groupes de rayons. Dans le contexte de la synthèse d'image par lancer de rayons, les rayons secondaires et d'une manière plus générale les rayons omnidirectionnels sont de nature divergente. Cette divergence n'assure pas une cohérence suffisante des rayons et rend l'exploitation des groupes de rayons difficiles ; l'utilisation de rayons individuels devient alors indispensable.

Disposant de deux structures de recherche, le kd-Tree et l'arbre BVH, il est intéressant de comparer leurs performances vis à vis des rayons individuels et des groupes de rayons. Afin d'évaluer leurs performances, une analyse de leur capacité à supporter des groupes de rayons est en premier lieu présentée. Ensuite, pour confirmer cette analyse, des expérimentations sont réalisées selon différentes tailles de groupes de rayons. Ces résultats permettent la définition d'une stratégie de couplage des deux structures de recherche, qui est enfin évaluée et validée avec le rendu de différents modèles.

1.2 Comparaison des structures de données

Les deux structures de recherche retenues pour comparer les performances des groupes de rayons sont le kd-Tree et l'arbre BVH. Celles-ci ont été sélectionnées car elles présentent une différence fondamentale sur la nature de construction qui respectivement, subdivise l'espace, ou sépare les surfaces (voir la figure V.1). Une analyse théorique de ces structures est présentée en suivant.



(a) kd-Tree : rayon simple

(b) BVH : rayon simple

(c) kd-Tree : groupe de rayons

(d) BVH : groupe de rayons

FIG. V.1 – Etapes de traversée pour le kd-Tree et l'arbre BVH. Les croix identifient les plans séparateurs évalués pour le kd-Tree, et les boîtes évaluées pour l'arbre BVH. Dans le cas du kd-Tree, 5 plans séparateurs doivent être évalués lors de la traversée d'un rayon simple (a) et 7 plans séparateurs pour le groupe de rayons (c). Dans le cas de l'arbre BVH, seules 3 boîtes sont évaluées aussi bien pour le rayon simple (b) que pour le groupe de rayons (d). Cet exemple simple illustre pourquoi l'arbre BVH est moins sensible aux larges groupes de rayons.

1.2.1 Analyse du kd-Tree

Les éléments suivants résument les points forts et points faibles du kd-Tree dans le contexte de la performance de la requête d'intersection.

Points forts :

- *Faible coût de traversée.* Durant la propagation d'un rayon dans le kd-Tree, il suffit de tester l'intersection d'un rayon avec le plan séparateur d'un noeud pour déterminer le ou les fils traversés. Etant donné que les plans séparateurs sont des plans alignés aux axes, ce test est exécuté très rapidement.
- *Peu de triangles dans les feuilles.* Grâce à sa capacité à subdiviser l'espace, le raffinement d'un kd-Tree est suffisamment fin pour limiter à un petit nombre, les triangles répartis dans les feuilles (généralement moins de 3).
- *Traversée ordonnée.* Quand l'intersection la plus proche a été trouvée dans une feuille, la traversée de l'arbre peut être interrompue immédiatement. L'ordre de traversée ordonné garantit qu'aucune autre intersection plus proche ne pourra être trouvée dans la partie de l'arbre non explorée.

Points faibles :
- *Arbre profond.* Pour permettre une bonne subdivision de l'espace, le kd-Tree doit positionner de nombreux plans séparateurs. Cette abondance de plans est un frein à l'utilisation de larges groupes de rayons car la probabilité de rencontrer des bifurcations est grande.
- *Triangles partagés.* Lors de chaque subdivision de l'espace par un plan, les triangles intersectant ce plan doivent être référencés dans les deux sous-espaces. Cette duplication de références est opérée de nombreuses fois au cours de la construction du kd-Tree, et génère une occupation mémoire conséquente (voir le tableau V.1).

1.2.2 Analyse du BVH

De la même manière que pour le kd-Tree, les points forts et points faibles de l'arbre BVH sont résumés.

Points forts :
- *Rejet rapide des grands espaces vides.* L'arbre BVH englobe la géométrie au plus prêt, dans les trois dimensions, en une seule opération de séparation. Lors d'un choix de traversée, un rayon est susceptible de parcourir les deux boîtes filles, une de ces deux boîtes filles ou aucune des deux. Le dernier cas représente souvent des grands espaces vides (voir le tableau V.1) qui limitent rapidement la propagation des rayons.
- *Petit arbre.* Contrairement au kd-Tree, les espaces vides ne sont pas identifiés en tant que tels mais déduits par l'absence d'intersections avec une boîte. Pour cette raison, l'arbre BVH est moins profond que le kd-Tree, ce qui est une propriété très intéressante pour supporter efficacement les groupes comportant de nombreux rayons.
- *Faible consommation mémoire.* Puisque l'arbre BVH est petit et que les références de surfaces sont uniques dans chaque feuille, son occupation mémoire est naturellement réduite par rapport au kd-Tree.

Points faibles :
- *Coût de traversée élevé.* Afin de déterminer les boîtes filles traversées, deux tests d'intersection rayon/boîte doivent être réalisés pour l'arbre BVH là où on un seul test d'intersection rayon/plan suffit pour le kd-Tree. Le nombre d'opérations nécessaires, par niveau de traversée, est alors entre 2 à 6 fois plus important.
- *Traversée non ordonnée.* Etant donné que les boîtes d'un arbre BVH peuvent se chevaucher, il est impossible d'établir un ordre de traversée strict, et ainsi d'arrêter la traversée au plus tôt dès qu'une intersection est trouvée.

1.2.3 Discussion

Cette analyse des structures de données montre que le kd-Tree a un algorithme de traversée plus efficace. Cependant, son arbre plus profond implique des difficultés

FIG. V.2 – Les modèles utilisés pour les expérimentations sont de gauche à droite : *ERW6* (804 triangles), *Bunny* (69K triangles), *Fairy Forest* (174K triangles), *Conference* (282K triangles), et *Soda Hall* (2 169K triangles).

à traiter de manière efficace de larges groupes de rayons (cf. V.1).

L'arbre BVH requiert plus de calculs par étape de traversée, mais ceux-ci sont suffisamment amortis quand ils sont partagés par les rayons d'un groupe, et moins nombreux car l'arbre est moins profond.

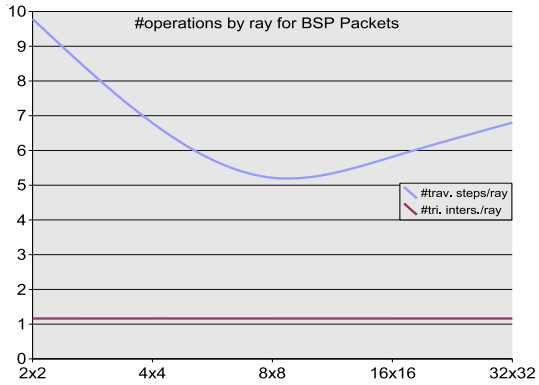
Cette analyse est confirmée en pratique en mesurant les performances brutes.

1.3 Performances brutes

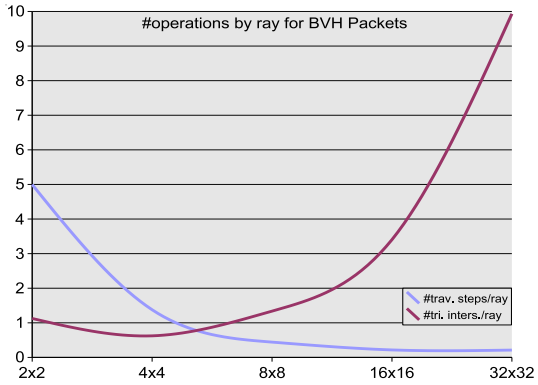
Afin d'évaluer les performances des groupes de rayons, des mesures sont réalisées à partir des rayons primaires. Ceux-ci ont l'avantage d'être facile à générer par un simple découpage de l'image en rectangles ; de plus le temps de construction des faisceaux n'a aucune conséquence sur les résultats.

Les expérimentations ont été réalisées sur les cinq modèles présentés dans la figure V.2 pour une résolution d'image de 1024^2 pixels. Les modèles ont été rendus pour les deux structures de recherche en faisant varier la taille des groupes de rayons. La figure V.3 montre les résultats moyennés pour les facteurs d'accélération, ainsi que les nombres d'étapes de traversées et de calculs d'intersections. Les accélérations minimales et maximales sont également indiquées dans le but d'évaluer les cas extrêmes. Toutes les accélérations sont relatives à un temps de référence obtenu avec des rayons simples pour le kd-Tree. Trois principales observations peuvent être mises en avant :

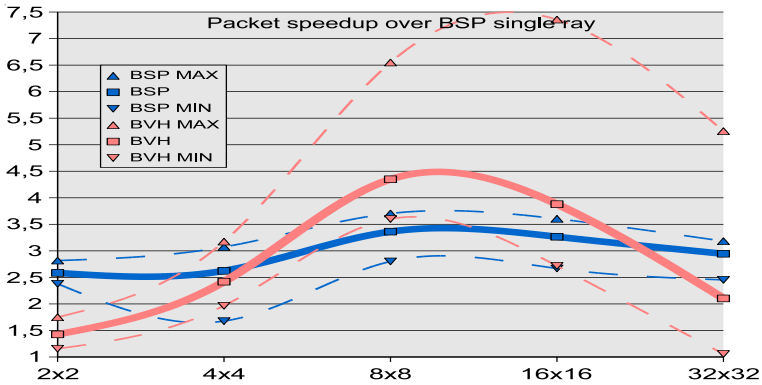
- Si l'on ne considère que la performance absolue, l'arbre BVH offre les meilleures performances avec un facteur d'accélération de x4,5. Cette accélération atteinte pour des groupes de 8x8 rayons est expliquée par une réduction conséquente du nombre d'étapes de traversée V.3 (divisé par 10) et une augmentation tolérable du nombre de tests d'intersections (x1,3).
- Toujours en étudiant seulement la performance, l'arbre BVH détient les plus mauvais temps avec un facteur d'accélération de seulement x1,5 pour les rayons SIMD. Même si le nombre d'étapes de traversée est moitié moins important que pour le kd-Tree pour le même nombre de rayons, les tests d'intersection rayon/boîte plus coûteux que les tests d'intersection rayon/plan, ne permettent pas à l'arbre BVH d'atteindre le même facteur d'accélération. L'efficacité du kd-Tree est alors plus intéressante pour les petits groupes de rayons.



(a) Moyenne des nombres d'étapes de traversée et de calculs d'intersections par rayon pour le kd-Tree.



(b) Moyenne des nombres d'étapes de traversée et de calculs d'intersections par rayon pour l'arbre BVH.



(c) Accélération relative des groupes de rayons par rapport à des temps de référence obtenus avec des rayons simples pour le kd-Tree. La meilleure performance est obtenue pour un large groupe de 8x8 rayons avec l'arbre BVH (x 4,5). La meilleure performance pour un petit groupe de 2x2 rayons est obtenue avec les rayons SIMD du kd-Tree (x 2,5).

FIG. V.3 – Performance relative des rayons primaires.

– Chaque arbre atteint un facteur d'accélération maximal pour des groupes de 8x8 rayons. Pour des groupes plus larges, les performances décroissent tant pour l'arbre BVH que pour le kd-Tree. Pour l'arbre BVH, le trop grand nombre d'intersections devient alors pénalisant, tandis que l'accélération du kd-Tree disparaît à mesure que le nombre d'étapes de traversée augmente.

1.4 Stratégie de couplage

1.4.1 Choix de la meilleure structure de recherche

Les résultats de ces expérimentations amènent à s'interroger sur la meilleure structure de recherche à utiliser dans le but d'exploiter au mieux les groupes de rayons.

1.4.1.1 Antagonisme Si l'on considère tous les rayons potentiellement générés lors d'un rendu, il n'est alors pas toujours possible de construire des groupes cohérents. Un moteur de requêtes d'intersection complet doit néanmoins pouvoir supporter au mieux tous les types de rayons, du rayon simple (ou SIMD) qui s'adapte bien à un échantillonnage omnidirectionnel, aux larges groupes de rayons capables d'exploiter au mieux la cohérence spatiale.

Une approche commune choisirait l'arbre pour lequel les performances moyennes seraient les meilleures avec un large ensemble de modèles, même si celles-ci ne le sont pas pour des modèles spécifiques, ou plus précisément pour des types de rayons particuliers. En suivant cette logique, le kd-Tree est la meilleure solution car ses performances sont toujours bonnes sans avoir recours à des solutions pouvant être complexes [9], et parfois défavorables, pour regrouper des rayons. Cependant, l'arbre BVH offre un facteur d'accélération meilleur pour les larges groupes de rayons. Ces performances antagonistes mènent vers un choix de la meilleure structure de recherche qui n'est pas optimal.

1.4.1.2 Généralisation du problème Bien que les performances relatives du kd-Tree et de l'arbre BVH peuvent être dépendantes de choix d'implémentation et d'efforts d'optimisation, les résultats peuvent être généralisés pour tout type de rayons et pour tout type d'implémentation si l'on considère les remarques suivantes :

- Les résultats des rayons primaires peuvent être transposés à tout type de rayon. Les résultats montrent alors un manque de performances pour le kd-Tree quand, à cohérence constante, le nombre de rayons dans un groupe devient important.
- L'analyse générale des structures de données présentée précédemment, qui ne dépend pas de l'implémentation, confirme des expérimentations faites selon des implémentations spécifiques.
- L'équation proposée, qui prétend que l'arbre BVH est mieux adapté aux larges groupes de rayons et que le kd-Tree est plus performant avec des rayons simples ou SIMD, est partagée avec d'autres travaux. En premier lieu Dmitriev *et al.* [21] et Benthin [6] ont également prouvé que la performance des groupes de rayons avec le kd-Tree est limitée. Ensuite, en comparant les expérimentations exploitant un kd-Tree, présentées dans la thèse de Wald [76], aux expérimentations exploitant un arbre BVH présentées dans d'autres travaux de Wald *et al.* [77], et implémentées très probablement avec le même niveau d'optimisation, l'équation proposée est encore confirmée. De plus, Boulos *et al.* [10] suggèrent d'utiliser un arbre BVH de préférence au kd-Tree, pour mieux supporter les groupes de rayons générés dans un rendu réaliste. Enfin, Reshetov [64] montre une sévère limitation des groupes de rayons secondaires avec le kd-Tree.
- Les structures de recherches exploitent des algorithmes différents mais sont implémentés, dans notre cas, avec le même niveau d'optimisation. Plusieurs fonctions et types de données sont partagés par les deux implémentations comme : le générateur de rayons, la procédure de construction des groupes de rayons, les données d'accélération des triangles, la fonction d'intersection rayon/triangle, les modèles d'illumination... De ce fait, les différents temps d'exécution sont

directement en rapport avec les complexités algorithmiques dégagées lors de l'analyse des structures.

L'antagonisme exposé n'est donc pas lié à un choix d'implémentation spécifique, mais peut être généralisé pour toutes les implémentations. Afin d'éviter un choix difficile, une utilisation couplée des deux structures de recherche est proposée.

1.4.2 Stratégie globale

Ray types	BSP 2x2 rays	BVH 8x8 rays
Primary rays		
Shadow rays		
Secondary rays		
Omnidirectionnal rays		

FIG. V.4 – La stratégie globale exploite conjointement le kd-Tree et l'arbre BVH selon le type de rayons.

En fonction des résultats des rayons primaires, une limite empirique est fixée pour dissocier les rayons incohérents des rayons cohérents, qui sont alors respectivement traités avec de petits groupes rayons ou de larges groupes de rayons. Les petits groupes sont modélisés par des rayons SIMD, tandis que les larges groupes sont constitués de 8x8 rayons (soit 4x4 rayons SIMD). Pour bénéficier du meilleur facteur d'accélération, le kd-Tree est uniquement utilisé pour les rayons SIMD, tandis que l'arbre BVH est utilisé pour les groupes de 8x8 rayons. La figure V.4 présente la stratégie de couplage selon le type de rayon.

En raison d'une grande cohérence de construction, les rayons primaires et les rayons d'ombre peuvent exploiter les larges groupes de rayons, ils sont alors naturellement utilisés avec l'arbre BVH. Les rayons secondaires sont plus délicats à traiter, il est nécessaire de les séparer en deux catégories :

- Le premier cas rassemble tous les groupes de rayons intersectant des surfaces homogènes (voir la figure V.6). Par surface homogène, on peut comprendre une surface coplanaire dont le matériau est homogène. La construction des rayons secondaires dans ces cas là est triviale, et présente une très forte probabilité de cohérence.
- Le second cas traite de tous les rayons qui ne se situent pas dans le premier cas. Les groupes de rayons sont alors éclatés en rayons SIMD, et exécutés avec le kd-Tree.

Tous les autres rayons, que l'on qualifie de rayons omnidirectionnels mais qui peuvent aussi bien représenter des rayons d'illumination globale, que des rayons d'occlusion ambiante, sont traités comme des rayons SIMD avec le kd-Tree.

Il aurait certainement été plus attrayant de créer une structure de recherche hybride qui puisse satisfaire aussi bien les petits groupes de rayons que les plus larges. Cependant, les solutions de ce type existantes [75, 32] montrent, bien qu'offrant des propriétés très intéressantes quant au temps de construction, qu'elles ne rivalisent pas avec les points forts détenus par le kd-Tree et l'arbre BVH. De plus, sans surcharger lourdement un moteur de requêtes d'intersection, les deux structures de recherche peuvent cohabiter simplement.

1.4.3 Cohabitation des arbres

Modèles	# ref tri (x 1000)	# noeuds (x 1000)	Taille	Profondeur moyenne	Surface des feuilles non vides
Kd-Tree					
Conference	1 232	607	9,8 Mo	25,5	6,3
Fairy	635	434	6,0 Mo	25,3	7,8
Soda Hall	11 553	7 995	110,2 Mo	29,6	16,1
BVH					
Conference	283	114	4,8 Mo	18,6	4,2
Fairy	174	68	2,9 Mo	17,3	2,9
Soda Hall	2 146	821	34,8 Mo	21,9	5,8

TAB. V.1 – Données statistiques collectées lors de la construction totale des arbres kd-Tree et BVH. De part un nombre moins important de références aux triangles, et de part une arborescence moins complexe, l'occupation mémoire de l'arbre BVH est bien inférieure à celle du kd-Tree. La surface totale des feuilles node vides, de 2 à 3 fois plus importantes avec le kd-Tree, tend à prouver que les espaces vides sont plus importants avec l'arbre BVH. Ces grands espaces vides rendent l'arbre BVH plus efficace quand le nombre de rayons dans un groupe augmente.

La figure V.5 montre comment une simple encapsulation permet d'abstraire l'accès à plus d'une structure de recherche. Les points suivants montrent que la surcharge occasionnée est acceptable :

- *Occupation mémoire.* Une stratégie de construction paresseuse (voir la section 2.4 à la page 33) limite la consommation globale de mémoire. De plus, l'ajout d'un arbre BVH en supplément d'un kd-Tree occasionne une surcoût acceptable (voir le tableau V.1).
- *Dynamique de mise à jour.* Si l'on considère le besoin de mettre à jour dynamiquement la géométrie, la nécessité de reconstruire deux arbres devrait nuire à l'interactivité. Cependant, la construction paresseuse atténue ce problème, et l'utilisation d'instances, pour dissocier la géométrie statique de la géométrie dynamique, limite généralement le besoin de reconstruction à un petit nombre de surfaces.
- *Incohérence d'accès à la mémoire.* Un problème d'incohérence d'accès la mémoire est provoqué par l'alternance des requêtes entre les deux arbres. Afin

d'optimiser la localité d'accès aux données, les calculs doivent être regroupés par type d'accès aux structures de recherche.

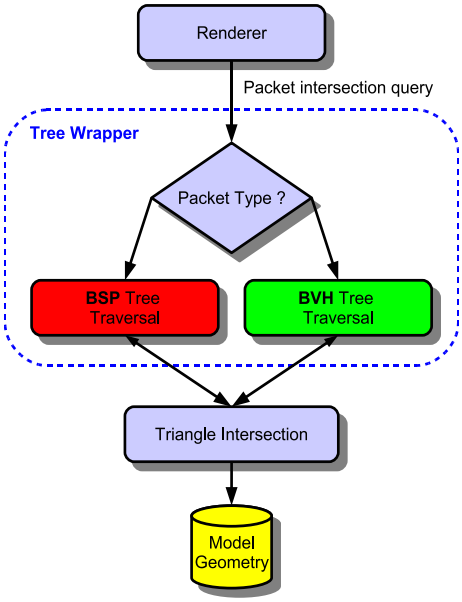


FIG. V.5 – Encapsulation du kd-Tree et de l'arbre BVH dans un moteur de requêtes d'intersection.

1.5 Résultats

1.5.1 Description des expérimentations

Pour l'ensemble des expérimentations, la répartition entre petits groupes de rayons et larges groupes de rayons est identique ; seules diffèrent les structures de recherche utilisées. Afin d'évaluer la stratégie globale, trois scénarios sont envisagés :

- Le premier scénario utilise le *kd-Tree uniquement* ;
- Le second scénario utilise l'arbre *BVH uniquement* ;
- Le troisième scénario utilise le *couplage des arbres* tel que décrit par la stratégie globale ;

Tous les scénarios sont testés en animant un parcours de caméra dans les modèles *Fairy Forest*, *Conference* et *Soda Hall*. Les matériaux ont été adaptés par rapport aux originaux dans le but de générer plus de rayons secondaires par la réflexion et la transparence. Deux sources de lumière ponctuelles ont été ajoutées dans chaque modèle.

Pour le modèle *Fairy Forest*, seule la première séquence de l'animation a été utilisée par rapport à l'animation complète téléchargeable. Des matériaux réfléchissants ont été ajoutés sur le sol, l'enclos, les rochers, les ailes de la fée et la baguette magique. Des matériaux transparents ont été ajoutés sur les plantes et l'habit de la fée. La caméra a été animée de manière à générer différents rapports de nombre de surfaces par pixel. En partant d'une vue globale, la caméra effectue une avancée sur la fée puis tourne autour d'elle et termine sa course par un gros plan sur son visage.

Pour le modèle *Conference*, tous les matériaux ont été mis réfléchissants à l'exception d'une transparence pour la vitre. La caméra a effectué un tour complet autour de la table centrale.

Pour le modèle *Soda Hall*, les sols de toutes les pièces et de petits détails, comme les lumières décoratives, ont été mis réfléchissants. La caméra a débuté son animation par une vue globale du bâtiment, puis est descendue dans une grande pièce au premier étage générant ainsi beaucoup de réflexions, et ensuite, après être passée dans un bureau et longer un couloir, la caméra a repris une vue globale du bâtiment.

Ces trois modèles sont libres d'utilisation et peuvent être retrouvés dans de nombreux travaux. Aucune mise à jour de géométrie n'est réalisée. Les performances d'exécution, en nombre de millions de rayons par seconde, sont présentées pour les modèles *Fairy Forest* et *Soda Hall* dans les figures V.7 et V.8. La figure V.9 compare les performances relatives des différents scénarios pour les trois modèles.

1.5.2 Rayons primaires et rayons d'ombre

Les rayons primaires profitent logiquement plus à l'arbre BVH comme le montre les figures V.8(a) et V.9(a) pour les deux modèles. Les accélérations moyennes sont comprises entre +10% et +24%. De la même manière, les rayons d'ombre bénéficient de meilleures performances pour les deux modèles avec l'arbre BVH (voir les figures V.8(b) et V.9(b)). L'accélération moyenne est de +17% pour le modèle *Fairy Forest* et +160% pour le modèle *Soda Hall*. Dans ce dernier cas, les groupes de rayons d'ombre sont largement plus efficaces avec l'arbre BVH car le modèle comporte beaucoup d'occlusions. Bien que l'arbre BVH ne dispose pas d'un ordre de traversée strict, les rayons d'ombre présentent la particularité générale d'arrêter tout calcul de traversée dès qu'une intersection est trouvée. Le kd-Tree exploite également cette particularité, mais doit cependant lutter avec une arborescence particulièrement profonde au regard de ce modèle complexe.

1.5.3 Rayons secondaires

Les rayons secondaires sont propagés avec un seul niveau de récursion pour simuler les réflexions et les transparences. En considérant le scénario *BVH uniquement*, les performances sont dégradées de -8% pour le modèle *Soda Hall* et de -13% pour le modèle *Fairy Forest* (voir les figures V.8(c) et V.9(c)). Le *couplage des arbres* permet alors d'avoir les meilleures performances en offrant une légère accélération de +5% pour le modèle *Soda Hall* et une accélération plus significative de +12% pour le modèle *Fairy Forest*. Ce dernier scénario (voir la figure V.9) évite alors les faibles performances des rayons SIMD obtenues avec l'arbre BVH.

1.5.4 Rayons omnidirectionnels

L'occlusion ambiante, telle que présentée dans la section 2 à la page 102, est utilisée pour simuler des rayons omnidirectionnels. Environ 200 000 rayons de ce

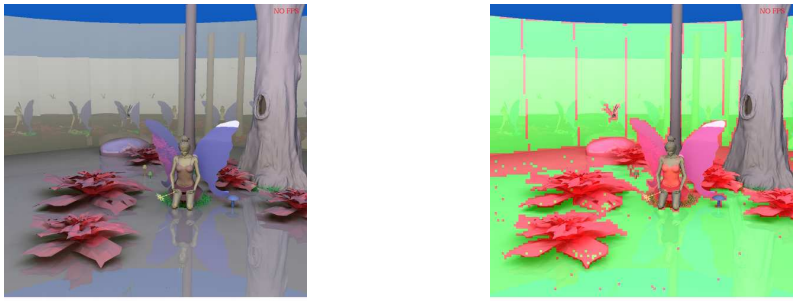
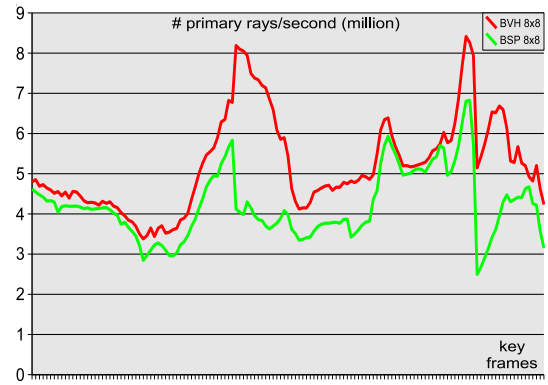


FIG. V.6 – Répartition des rayons secondaires entre rayons SIMD et groupes de rayons représentés respectivement en rouge et en vert. A gauche : image originale. A droite : 230K rayons rouges et 620K rayons verts.

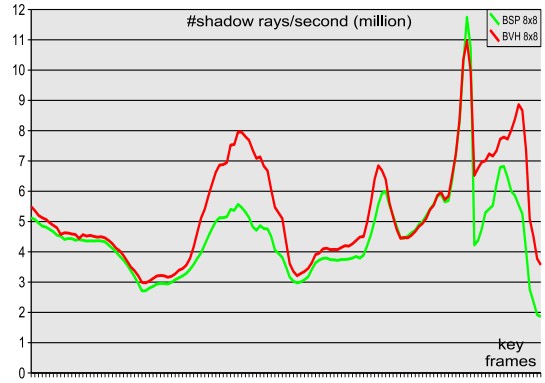
type regroupés sous forme de rayons SIMD sont lancés pour chaque image. Pour les deux modèles, l'arbre BVH génère les moins bons temps d'exécution avec une dégradation moyenne de -21% et -50% (cf V.8(d) et V.9(d)).

1.5.5 Performances relatives

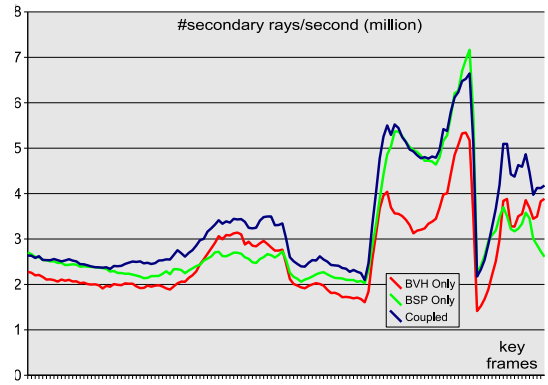
Tous les facteurs d'accélération présentés dans la figure V.9 sont relatifs au scénario *kd-Tree uniquement*. Si l'on considère le scénario *BVH uniquement*, les facteurs d'accélération sont mitigés car compris entre -10% et +22% pour l'ensemble des modèles. Ces résultats s'expliquent par un manque de performances dans le traitement des rayons secondaires incohérents et des rayons omnidirectionnels. Le scénario de *couplage des arbres* comble ces lacunes car il ne dégrade pas le facteur d'accélération dans le pire des cas, et offre une accélération intéressante de x1,5 dans le meilleur des cas avec le modèle *Soda Hall*. De ce fait, le *couplage des arbres* est le meilleur scénario pour l'exploitation efficace des groupes de rayons.



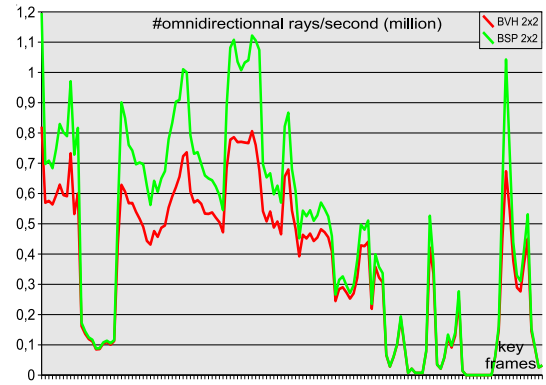
(a) Fairy Forest : rayons primaires (1 024K/image)



(b) Fairy Forest : rayons d'ombre (≈ 2 000K/image)

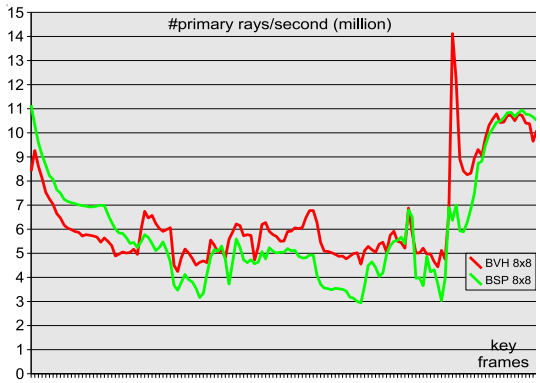


(c) Fairy Forest : rayons secondaires (≈ 900K/image)

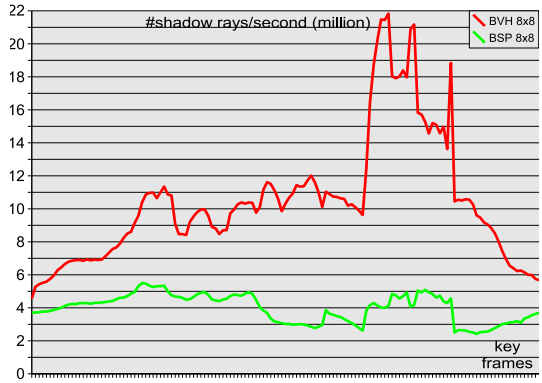


(d) Fairy Forest : rayons omnidirectionnels (≈ 200K/image)

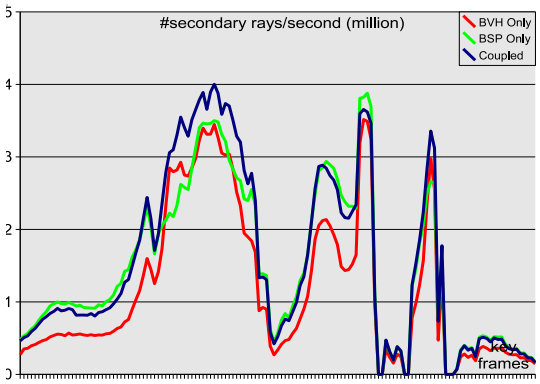
FIG. V.7 – Millions de rayons par seconde pour le modèle *Fairy Forest* selon le type de rayons.



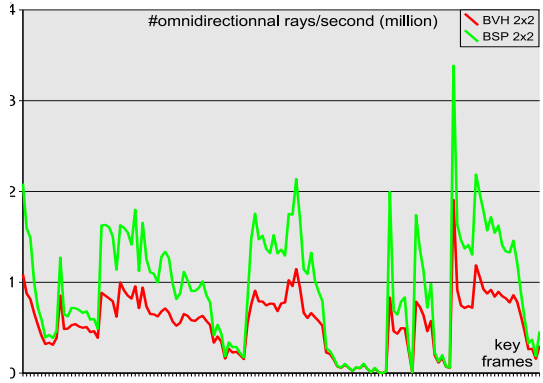
(a) Soda Hall : rayons primaires (1 024K/image)



(b) Soda Hall : rayons d'ombre (≈ 1 700K/image)

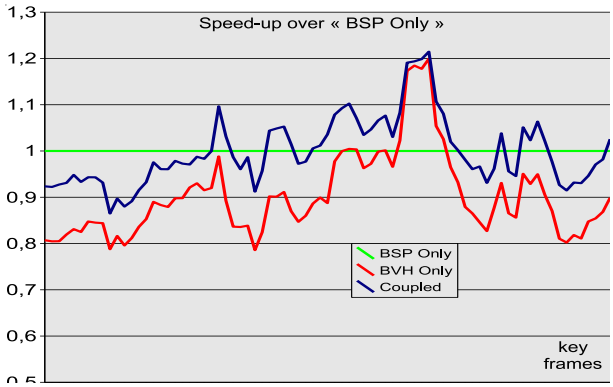


(c) Soda Hall : rayons secondaires (≈ 150K/image)

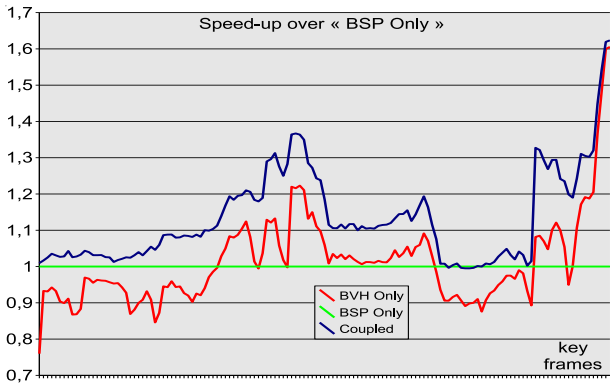


(d) Soda Hall : rayons omnidirectionnels (≈ 280K/image)

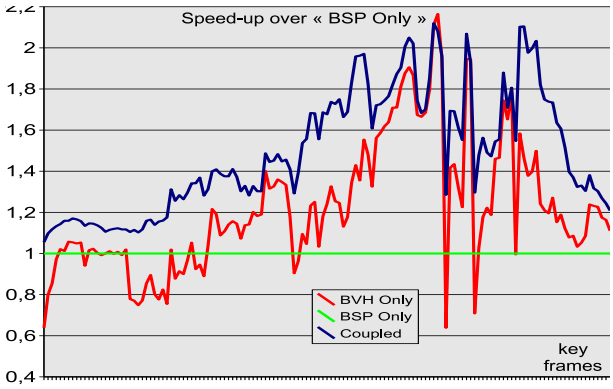
FIG. V.8 – Millions de rayons par seconde pour le modèle *Soda Hall* selon le type de rayons.



(a) Conference : accélération relative.



(b) Fairy Forest : accélération relative.



(c) Soda Hall : accélération relative.

FIG. V.9 – Accélération relative du scénario *BVH uniquement* et du scénario de *Couplage des arbres* par rapport au scénario *kd-Tree uniquement*. Les accélérations sont de +3 % pour le modèle *Conference*, + 22% pour le modèle *Fairy Forest* et +50 % pour le modèle *Soda Hall*.

1.6 Discussion

Les expérimentations sur les regroupements de rayons ont montré que l'arbre BVH est la meilleure structure de recherche pour traiter les rayons primaires et les rayons

d'ombre. Pour les rayons secondaires et les rayons omnidirectionnels, une approche de couplage de l'arbre BVH pour les larges groupes de rayons, et du kd-Tree pour les rayons SIMD, est proposée dans le but de maximiser les performances.

Les principales motivations de cette approche sont de montrer d'une part qu'il est intéressant d'exploiter les groupes de rayons, et d'autre part que l'utilisation de deux structures de recherche sur le même espace est une solution viable à condition d'exploiter de manière pertinente les points forts de chacune.

Naturellement, tous ces mécanismes ont pour but d'accélérer tout type de requête d'intersection. La possibilité de classer dynamiquement les types de rayons entre rayons incohérents et rayons cohérents offre l'avantage de choisir la structure de recherche la plus adaptée. Cependant, obtenir la performance des groupes de rayons pour des rayons omnidirectionnels est un nouvel enjeu. Il est maintenant nécessaire de vérifier si le regroupement d'un plus grand ensemble de rayons incohérents peut amener à des sous-ensembles nouveaux plus cohérents.

2 Vers un regroupement automatique des rayons

2.1 Problématique

La méthode proposée de couplage des structures de recherche s'appuie sur une stratégie simple et sûre pour dissocier les groupes de rayons cohérents de ceux incohérents. Cette méthode minimise le risque d'incohérence mais ne maximise pas l'utilisation de la cohérence. Plusieurs travaux [50, 6, 78] montrent que la cohérence se dégrade très vite lorsque la profondeur de récursion augmente pour les rayons secondaires, ou lors de l'utilisation de rayons omnidirectionnels. Cependant ces travaux mesurent la cohérence pour un groupe de rayons restreint. Afin de maximiser la cohérence, il est nécessaire de travailler avec un plus grand ensemble de rayons. Plus cet ensemble est grand et plus il est probable de trouver des sous-ensembles cohérents. Les travaux de Pharr *et al.* [55] et de Navratil *et al.* [53] vont dans ce sens en cherchant à limiter le nombre d'accès à la géométrie par les rayons. Tous les rayons sont alors regroupés dans les noeuds d'une structure de recherche additionnelle. La taille de ces noeuds est choisie de telle manière à optimiser la bande passante mémoire : du disque à la mémoire centrale pour Pharr et de la mémoire centrale au cache L2 pour Navratil. Ces travaux montrent alors une effective réduction du nombre d'accès à la géométrie pour des rendus complexes, ce qui induit une bonne exploitation sous-jacente de la cohérence des rayons. Cependant ces travaux n'exploitent pas, par la suite, les groupes de rayons dans la structure d'accélération. Navratil *et al.* soulignent alors que l'incohérence des rayons serait nuisible aux performances.

Pour exploiter avantageusement un grand ensemble de rayons, il est nécessaire d'une part de construire rapidement des groupes de rayons et d'autre part de savoir déterminer le facteur de cohérence afin de guider au mieux la construction. Dans les

paragraphes suivants, une analyse de la cohérence est proposée en premier lieu. Celle-ci montre un critère fiable permettant de mesurer la cohérence en évaluant la surface de coupe du faisceau, et propose une stratégie pour limiter les conséquences de l'utilisation d'un groupe de rayons incohérent. Dans un second temps, en s'appuyant sur le critère établi de cohérence, un algorithme de réorganisation automatique des rayons est proposé. Ces études se limitent à l'analyse des groupes de rayons avec l'arbre BVH, mais il est certain que celles-ci devront être étendues avec le kd-Tree.

2.2 Mesure de la cohérence

2.2.1 Travaux précédents

Quand la divergence des rayons devient trop grande, le nombre excessif de calculs d'intersections ne peut plus être amorti par le nombre de rayons. Au delà d'une perte d'efficacité, la performance peut être pire que si le groupe de rayons n'avait pas été utilisé (voir la figure V.12). Pour remédier à ce problème, il est nécessaire de déterminer a priori le seuil de cohérence nécessaire à une exploitation efficace. Benthin [6] propose un critère efficace de mesure de la cohérence C_t en comparant le nombre de noeuds traversés $N_{t/r}$ par rayon, au nombre de noeuds traversés au moins une fois N_{tu} pour l'ensemble des rayons. Ce même critère est transposé aux calculs d'intersection tel que :

$$C_t = \frac{N_{t/r}}{N_{tu}} \quad , \quad C_i = \frac{N_{i/r}}{N_{iu}}$$

C_t et C_i désignent alors le taux de cohérence pour les étapes de traversée et pour les calculs d'intersection. Les travaux montrent effectivement une diminution de la cohérence pour les rayons secondaires et les rayons omnidirectionnels. Cependant, bien que cette cohérence diminue et qu'elle soit corrélée avec un facteur de perte de performance, aucune méthode n'est proposée pour déterminer un seuil de cohérence permettant ainsi, soit de gérer autrement les groupes de rayons, soit de basculer de l'exécution d'un groupe de rayons à l'exécution indépendante de rayons. Pour parvenir à cela, il est nécessaire de savoir prédire la cohérence.

Les variables utiles aux calculs de C_t et de C_i ne sont pas disponibles avant l'exécution d'un groupe de rayons. Cette mesure de la cohérence ne peut se faire qu'a posteriori rendant alors impossible son exploitation pour une prévision dynamique de la cohérence. Reinhard *et al.* [63] proposent un modèle de prédiction de coût pour un rayon. Bien qu'offrant une approximation réaliste, la formulation proposée nécessite de mesurer la profondeur moyenne de l'arbre et de calculer un facteur d'occlusion des surfaces dans les feuilles alourdissant alors la construction de l'arbre. Cette formulation est d'une part trop générale pour la mesure de cohérence et inadaptée aux groupes de rayons.

Mansson *et al.* [50] proposent un critère plus adapté à la mesure de cohérence d'un groupes de rayons C_g en comparant le nombre de noeuds traversés $N_{t/g}$, et le nombre

d'intersections calculées $N_{i/g}$ pour le groupe de rayons, au nombre de noeuds traversés $N_{t/r}$, respectivement au nombre d'intersections calculées $N_{i/g}$, individuellement pour chaque rayon tel que :

$$C_{gt} = \frac{N_{t/r}}{N_{t/g}} \quad , \quad C_{gi} = \frac{N_{i/r}}{N_{i/g}}$$

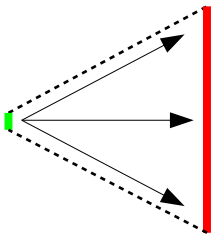
Comme le précisent Mansson *et al.* , C_{gt} et C_{gi} peuvent alors être traduits comme le nombre de rayons actifs dans le groupe. Une valeur inférieure ou égale à 1 indique une très mauvaise cohérence car chaque rayon du groupe a alors réalisé autant, voire plus d'opérations que s'il avait été exécuté individuellement. Une valeur proche du nombre de rayons dans le groupe indique une très forte cohérence car toutes les opérations ont pu être partagées. Ces valeurs semblent intéressantes pour déduire une valeur limite entre cohérence et incohérence mais les variables utiles à leur calcul ne sont toujours pas déterminables avant l'exécution d'un groupe de rayons.

A l'inverse des travaux présentés, nous proposons une nouvelle approche qui détermine la cohérence à partir de valeurs disponibles avant l'exécution d'un rayon, et non à partir de données statistiques collectées. Ainsi, le paragraphe suivant montre que le niveau de cohérence d'un groupe de rayons, caractérisé par C_{gt} et C_{gi}, est lié à la surface de coupe d'un faisceau.

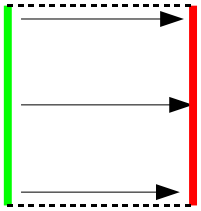
2.2.2 Cohérence liée à la surface de coupe d'un faisceau

2.2.2.1 Modélisation La cohérence d'un groupe de rayons est fonction de la proximité des rayons entre eux. Intituitivement, cette proximité semble pouvoir s'évaluer par la distance des origines et par l'écart des directions. Cependant, comme l'illustre la figure V.10, il n'est pas possible d'évaluer la proximité des rayons en s'appuyant sur l'un ou l'autre de ces paramètres, ou même les deux à la fois, pour les raisons suivantes :

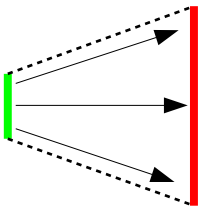
- des rayons ayant des origines proches peuvent diverger rapidement si leurs directions diffèrent ;
- des rayons colinéaires peuvent être très éloignés les uns des autres ;
- deux groupes de rayons présentant les mêmes écarts, entre origines et directions, n'ont pas forcément le même facteur de proximité.



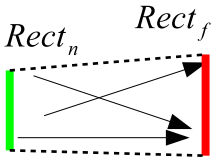
(a) Rayons à origine commune : incohérence.



(b) Rayons parallèles : incohérence.



(c) Rayons à origines et directions proches : incohérence.



(d) Rayons à origines et directions proches : cohérence.

FIG. V.10 – Cohérence liée à la surface de coupe d'un faisceau. (a) et (b) montrent que malgré des propriétés fortes d'origines communes ou de colinéarité, les groupes de rayons sont incohérents. (c) et (d) montrent deux groupes de rayons ayant les mêmes propriétés d'agencement. Cependant chacun d'eux présente un facteur de cohérence différent, qui peut être distingué par la surface de coupe.

En admettant que tous les rayons partagent une distance maximale de recherche t_{max}, la proximité des rayons peut être concrètement calculée par l'espacement moyen des points le long des rayons, depuis les origines jusqu'à t_{max} :

$$Proximite = \frac{1}{t_{max}} \int_0^{t_{max}} \sum_i^N \sum_j^N |p_i(t) - p_j(t)|.dt \qquad \text{(V.1)}$$

Dans cette équation, $p_k(t)$ désigne le point le long du rayon k tel que $p_k(t) = O_k + t.D_k$, et N désigne le nombre de rayons dans le groupe. Bien que cette équation ne puisse pas se résoudre dans des temps acceptables, elle pose des bases qui permettent d'effectuer des approximations en tenant notamment compte de la nature de construction des faisceaux.

Les faisceaux sont modélisés par quatre rayons encadrants issus de la construction de deux rectangles $Rect_n$ et $Rect_f$ englobants les points extrêmes des rayons (voir la section 1.3 à la page 56). On peut remarquer que les extrémités des rayons encadrants sont comparables aux points à $t = 0$ et $t = t_{max}$ de l'équation V.1, et que l'aire des rectangles $Rect_n$ et $Rect_f$ est fonction de l'espacement entres les points extrêmes. Puisque l'aire du rectangle $Rect_n$ évolue linéairement selon t vers l'aire du rectangle

$Rect_f$, le calcul d'intégrale de l'équation V.1 peut être avantageusement remplacé par un calcul de moyenne d'aires :

$$\begin{aligned} S_f &= \frac{1}{2}\{surface(Rect_n) + surface(Rect_f)\} \\ Proximite &\approx S_f \end{aligned}$$

Cette valeur approchée de la proximité des rayons correspond alors à la surface de coupe du faisceau S_f. Le paragraphe suivant valide cette modélisation en montrant par l'expérimentation, la corrélation entre la valeur de cette surface et les indicateurs C_{gt} et C_{gi} de cohérence proposés par Mansson *et al.* [50].

2.2.2.2 Validation Afin de valider la relation entre la surface de coupe d'un faisceau S_f et le niveau de cohérence, un banc de test à été développé afin de pouvoir créer un ensemble varié de groupes de rayons, tout en s'affranchissant d'une technique de génération de rayons contrainte par la physique.

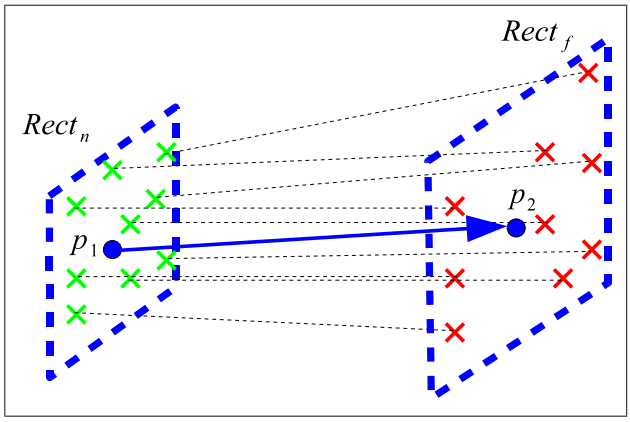


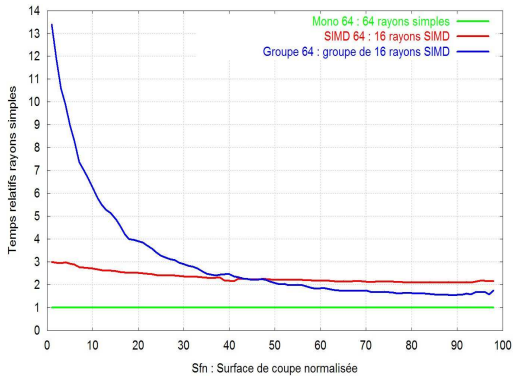
FIG. V.11 – Groupe de rayons généré aléatoirement dans la boîte englobante du modèle.

Un ensemble de 100 000 groupes de N_r rayons est généré aléatoirement. Pour chaque groupe, plusieurs scénarios d'exécution sont évalués :

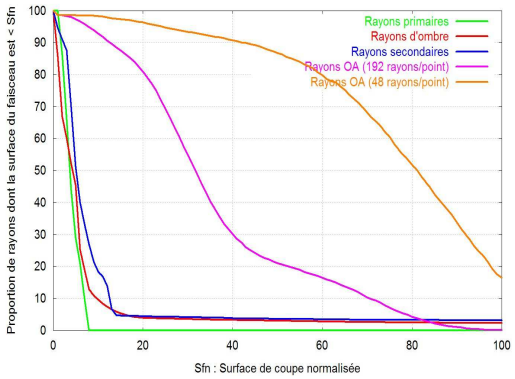
- *Mono* : exécution de chaque rayon simple individuellement ;
- *SIMD* : exécution de chaque rayon SIMD individuellement ;
- *Groupe* : exécution du groupe de rayons de manière classique (voir la section 4 à la page 76) ;

Pour générer un groupe particulier de N_r rayons, deux points p_1 et p_2 sont tirés aléatoirement dans la boîte englobante du modèle (voir la figure V.11). Selon la plus grande dimension séparant les deux points, deux rectangles orthogonaux de tailles aléatoires sont créés autour de chaque point. Dans chaque rectangle sont tirés aléatoirement un ensemble de N_r sous-points. Les sous-points du premier rectangle servent alors à définir les origines des rayons tandis que les seconds servent à déterminer les directions des rayons. Il est ainsi possible de créer toute forme de groupe de rayons.

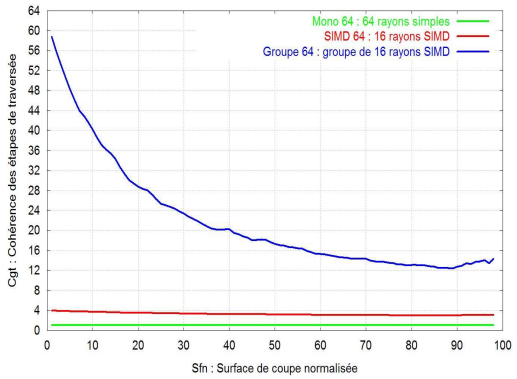
Pour chaque groupe de rayons, sont relevés le temps d'exécution (nombre de cycles CPU), le nombre d'étapes de traversée et le nombre de calculs d'intersection. Les groupes de rayons ne donnant lieu a aucune intersection ne sont pas retenus car ils ne sont pas représentatifs d'un cas concret d'utilisation. Chaque scénario est exécuté deux fois, et seuls les résultats de la deuxième exécution sont utilisés. Ceci permet d'éviter de perturber les résultats avec la construction paresseuse de l'arbre BVH.



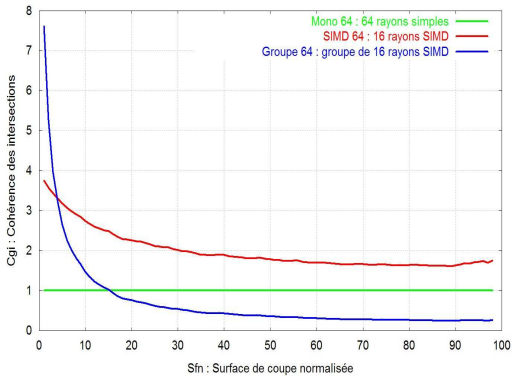
(a) Performances relatives aux rayons simples en fonction de S_{fn}.



(b) Proportion de rayons, selon le type, dont la surface du faisceau est inférieure à S_{fn}.



(c) Cohérence des étapes de traversées C_{gt} en fonction de S_{fn}.



(d) Cohérence des intersections C_{gi} en fonction de S_{fn}.

FIG. V.12 – Evolution des performances relatives (a), de C_{gt} (c) et de C_{gi} (d), pour les rayons SIMD et les groupes de rayons, par rapport aux rayons simples, selon la surface de coupe normalisée des faisceaux. Les valeurs C_{gt} et C_{gi} ont une forte corrélation avec S_{fn}, ce qui implique que S_{fn} peut être utilisé en lieu et place de ces indicateurs qui ne sont exploitables qu'a posteriori. La figure (b) montre la répartition de différents types de rayons issus d'un rendu en fonction de S_{fn}.

La figure V.12 présente les résultats d'expérimentations pour le modèle *Conference*. Les données statistiques sont mises en relation avec la surface de coupe S_f qui est normalisée par la somme des surfaces S_m de toutes les boîtes du modèle tel que :

$$S_{fn} = \frac{100 \times S_f}{10^{-4} \times S_m}$$

La surface de coupe normalisée S_{fn} varie alors essentiellement dans une plage de valeurs comprises entre 0 et 100, et offre la propriété intéressante d'être comparable entre différents modèles.

En terme de performance relative aux rayons simples, on constate que les temps sont directement fonction de la surface de coupe des faisceaux. Les rayons SIMD ont un facteur d'accélération qui varie significativement de x3 à x2 quand S_fn devient grand. Les groupes de rayons ont un facteur d'accélération, supérieur à x6 pour les surfaces de faisceau inférieures à 10, supérieur à x4 pour les surfaces comprises entre 10 et 20, et au delà le facteur d'accélération baisse rapidement jusqu'à être inférieur à celui des rayons SIMD pour des surfaces supérieures à 45. A titre de référence, les rayons de type primaires, d'ombre et les premiers niveaux de rayons secondaires ont des surfaces inférieures à 20 (voir la figure V.13(b)). Ces mesures concordent donc avec les accélérations mesurées pour ces types de rayons dans les sections 4 à la page 76 et 1.5 à la page 126. Les rayons d'occlusion ambiante à faible densité d'échantillonnage ont pour leur majorité (environ 90%) une surface supérieure à 45 ; cette répartition explique pourquoi ces types de rayons ne bénéficient pas de l'accélération des groupes de rayons comme montré dans la section 2 à la page 102.

Les figures V.13(c) et V.13(d) montrent, d'une part, que les indicateurs de cohérence C_{gt} et C_{gi} sont fortement liés à la surface de coupe normalisée S_{fn}. Cette propriété confirme que S_{fn} est un critère sûr de mesure de la cohérence, et donc que cette valeur peut être utilisée pour optimiser l'organisation d'un groupe de rayons. L'évolution même des indicateurs montre que la cohérence se dégrade fortement quand S_{fn} augmente. La cohérence de traversée est très forte initialement, avec 60 rayons sur 64 qui partagent les mêmes étapes de traversée, puis atteint approximativement un quart de rayons cohérents quand S_{fn} est très grand (proche de 100). Bien que diminuée, cette cohérence de traversée existe toujours et n'explique pas à elle seule la dégradation des performances. L'explication provient de la cohérence des calculs d'intersection qui passe rapidement en dessous de la barre de 1 pour S_{fn} supérieur à 15, indiquant alors un nombre de calculs plus important que si les rayons avaient été lancés individuellement.

Afin de remédier au fait que le traitement d'un groupe de rayons peut être plus lent qu'un ensemble de rayons SIMD, on pourrait envisager une heuristique qui arbitre entre ces deux types de rayons en fonction de S_{fn}. Cependant, cette valeur est globale pour le groupe de rayons et ne traduit pas fidèlement qu'un groupe de rayons peut être tour à tour cohérent ou incohérent en fonction de la propagation de son faisceau. Pour parvenir à cela, il est intéressant d'évaluer la proportion de rayons actifs *avant* la traversée d'une boîte et de choisir, en fonction de cette proportion, entre continuer la propagation du faisceau ou changer pour l'exécution plus sûre de rayons SIMD. Le paragraphe suivant présente une façon d'évaluer la proportion de rayons actifs à partir des rayons encadrants, et propose une heuristique pour arbitrer durant la propagation, entre groupe de rayons et rayons SIMD.

2.2.3 Evaluation de la proportion de rayons actifs pour chaque boîte

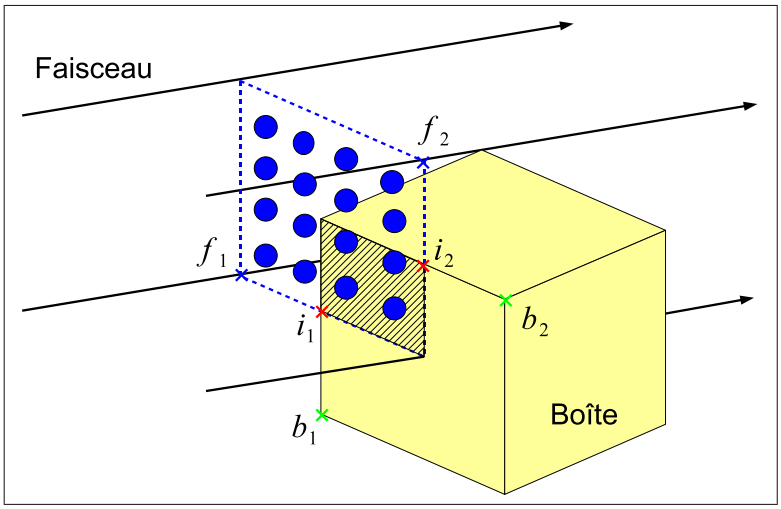


FIG. V.13 – Approximation de la proportion de rayons actifs, par l'utilisation des rayons encadrants, lors de l'intersection d'un faisceau et d'une boîte. Avec un surface relative d'environ 25%, la proportion de 4/16 rayons actifs est correctement estimée.

2.2.3.1 Modélisation La proportion de rayons actifs $P_{ra/b}$ supposés traverser une boîte b, peut être déduite, sans calcul explicite d'intersections rayon/boîte et en supposant une répartition homogène des rayons dans le groupe, en comparant la surface d'intersection $S_{f\cap b}$ du faisceau f avec la boîte b, à la surface propre $S_{f/b}$ du faisceau au niveau de la boîte b tel que :

$$P_{ra/b} = \frac{S_{f\cap\ b}}{S_{f/b}}$$

$S_{f\cap\ b}$ et $S_{f/b}$ sont calculés indépendamment pour les trois faces b_i de la boîte les plus proches de l'origine du faisceau. La figure V.14 illustre le calcul pour une face de la boîte. Pour chacune des faces, sont calculés les points d'intersection 2D $f1$ et $f2$ des quatre rayons constituants le faisceau. L'aire formée par le rectangle $\{f1, f2\}$ représente la surface propre S_{f/b_i} du faisceau f pour la face b_i de la boîte. En utilisant les sommets extrêmes $b1$ et $b2$ de la face b_i de la boîte, le rectangle $\{i1, i2\}$ est déterminé par intersection tel que :

$$\begin{aligned} i1_u &= max(f1_u, b1_u)\ , \quad i1_v = min(f1_v, b1_v) \\ i2_u &= max(f2_u, b2_u)\ , \quad i2_v = min(f2_v, b2_v) \end{aligned}$$

L'aire formée par le rectangle $\{i1, i2\}$ représente alors la surface d'intersection $S_{f\cap b_i}$ du faisceau. La valeur de $P_{ra/b}$ est obtenue par addition des valeurs intermédiaires calculées pour chaque axe tel que :

$$P_{ra/b} = \sum_{i\in\{x,y,z\}} \frac{S_{f\cap b_i}}{S_{f/b_i}}$$

On peut noter qu'un cas particulier est traité si $P_{ra/b} = 0$, car cela traduit la non intersection du groupe de rayons avec la boîte b. Une interprétation directe impliquerait qu'aucun rayon ne soit actif pour ce noeud, et donc que la cohérence est inexistante. Or, c'est exactement l'inverse qui se produit car le faisceau est cohérent en évitant le noeud dans son ensemble. Pour ce cas précis, la résultat $P_{ra/b} = 0$ doit être traité comme un facteur de cohérence maximale comme cela est proposé en suivant.

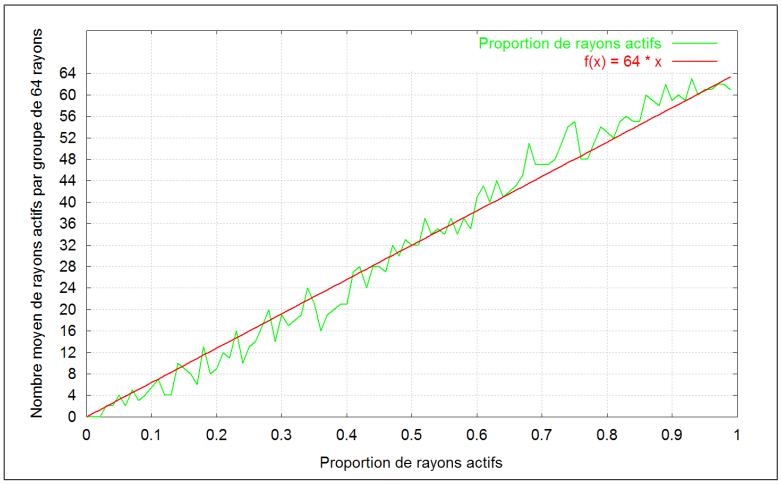


FIG. V.14 – Nombre moyen de rayons actifs, pour des groupes de 64 rayons, en rapport avec la proportion de rayons actifs $P_{ra/b}$ calculée. En comparant les mesures à la courbe théorique, la formulation permet de déterminer précisément le nombre de rayons actifs.

La figure V.14 compare l'évaluation de la proportion de rayons actifs au nombre moyen de rayons actifs lors de chaque traversée de boîte, et montre que la fonction $P_{ra/b}$ permet d'approcher précisément la réalité. Il est alors intéressant de voir dans quelle mesure cette fonction permet de déterminer la cohérence d'un groupe de rayons.

2.2.3.2 Heuristique de surface pour limiter la dégradation des performances Disposant d'un opérateur a priori pour déduire un facteur de cohérence au niveau d'une boîte, une série d'expérimentations a été réalisée pour valider l'approche. En premier lieu, l'algorithme de traversée de l'arbre BVH est modifié pour intégrer une heuristique de surface $D(f, b)$ qui évalue dynamiquement la cohérence d'un faisceau f avant chaque test d'intersection faisceau/boîte. Cette heuristique indique alors si le faisceau est divergent selon le principe suivant :

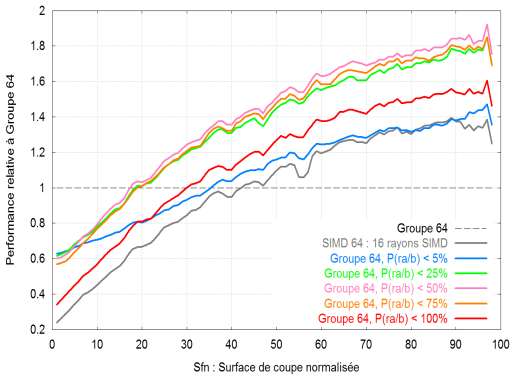
$$D(f,b) = \begin{array}{ll} Vrai & \text{si } 0 < P_{ra/b} < K \\ Faux & \text{sinon} \end{array}$$

Dans le cas où le faisceau est divergent, la traversée par groupe de rayons est interrompue pour la boîte b, et chaque rayon SIMD est lancé individuellement dans

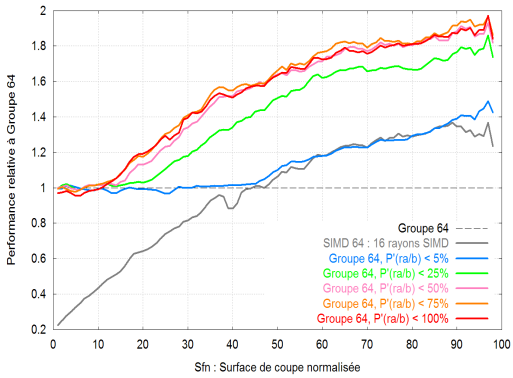
le sous-arbre ayant pour racine b. Aucun changement n'est appliqué en cas de cohérence. On peut noter que K est une constante à déterminer qui correspond au seuil entre cohérence et incohérence. Pour trouver une valeur optimale et valider l'approche, les expérimentations ont été étendues en exécutant les groupes de 64 rayons pour différentes valeurs de K.

Les figures V.16(a), V.16(c) et V.16(e) présentent les résultats des expérimentations en terme de performance et de nombre d'opérations relativement aux groupes de rayons sans l'usage de l'heuristique de surface. Les meilleurs résultats sont obtenus pour une valeur de K comprise entre 25% et 75%. Dans cette plage de valeurs et pour une surface de coupe normalisée du faisceau supérieure à 45, l'heuristique permet d'avoir un facteur d'accélération supérieur aux rayons SIMD d'environ 40%. Cette amélioration est due essentiellement à une meilleure maîtrise du nombre de calculs d'intersection tout en augmentant faiblement les nombre d'étapes de traversée.

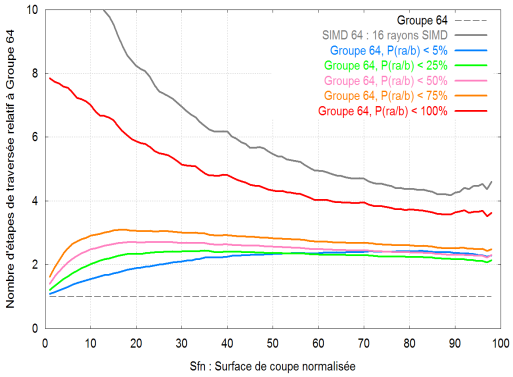
Cependant, si l'on considère les performances pour S_{fn} inférieur à 45, les temps sont augmentés d'environ 40% par rapport aux groupes de rayons sans heuristique. Deux hypothèses peuvent être émises pour expliquer cette pénalité. La première hypothèse suppose que l'heuristique utilisée n'est efficace que pour des valeurs de S_{fn} supérieures à 45 et pénalisante dans le cas inverse. La seconde hypothèse suppose que l'évaluation de la proportion de rayons actifs ajoute un coût constant au calcul d'intersection faisceau/boîte, et que ce coût n'est amorti par une réduction du nombre de calculs d'intersections qu'à partir de valeurs de S_{fn} supérieures à 45. En observant la courbe du groupe de rayons avec un usage très modéré de l'heuristique pour $P_{ra/n} < 5\%$, on peut remarquer, quand S_{fn} est proche de 0, que les nombre d'étapes traversées et nombre de calculs d'intersections sont les même que ceux des groupes de rayons sans heuristique. Pour autant, la courbe de performance est dès le départ affectée par une pénalité de 40%, ce qui valide la seconde hypothèse du surcoût.



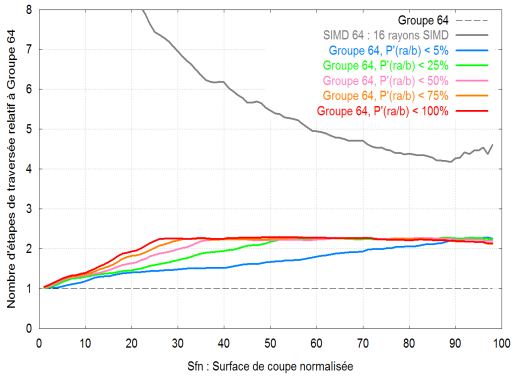
(a) Performance relative à Groupe 64. Utilisation de $P_{ra/b}$.



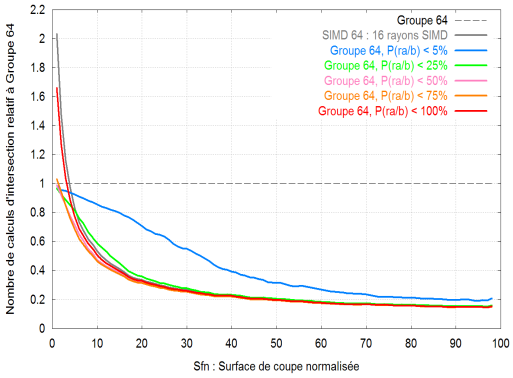
(b) Performance relative à Groupe 64. Utilisation de $P'_{ra/b}$.



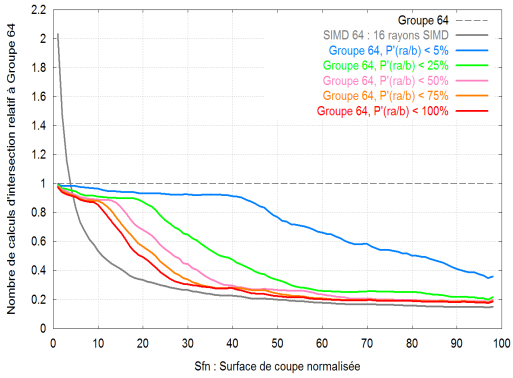
(c) Nombre d'étapes de traversées. Utilisation de $P_{ra/b}$.



(d) Nombre d'étapes de traversées relatif à Groupe 64. Utilisation de $P'_{ra/b}$.



(e) Nombre de calculs d'intersections relatif à Groupe 64. Utilisation de $P_{ra/b}$.



(f) Nombre de calculs d'intersections relatif à Groupe 64. Utilisation de $P'_{ra/b}$.

FIG. V.15 – Performances et données statistiques des groupes de rayons avec utilisation de l'heuristique par rapport aux groupes de rayons sans heuristique. Les figures (a), (c) et (e) ont été réalisées avec une utilisation de l'évaluation de la proportion $P_{ra/b}$, tandis que les figures (b), (d) et (f) ont été réalisées avec la valeur approchée $P'_{ra/b}$. Bien que l'évaluation de la proportion de rayons actifs $P_{ra/b}$ permettent de limiter efficacement le nombre de calculs d'intersections, le surcoût de calculs dû à sa mise en place pénalise les groupes de rayons cohérents. L'approximation $P'_{ra/b}$ évite ce surcoût tout en maîtrisant le nombre de calculs d'intersection.

Etant donné que l'évaluation de la proportion de rayons actifs $P_{ra/b}$ est coûteuse, une approximation de cette valeur ne nécessitant aucun calcul est réalisée. L'évaluation de la surface du faisceau $S_{f/b}$ au niveau d'une boîte est remplacée par la surface moyenne de coupe S_f du faisceau précalculée, et l'évaluation de la surface d'intersection $S_{f\cap b}$ est remplacée par la surface S_b de la boîte également précalculée. La valeur approchée de la proportion de rayons actifs devient alors :

$$P'_{ra/b} = \frac{S_b}{S_f}$$

Les figures V.16(b), V.16(d) et V.16(f) montrent les résultats obtenus en utilisant la même heuristique mais en remplaçant $P_{ra/b}$ par $P'_{ra/b}$. Les performances des groupes de rayons cohérents ($S_{fn} < 45$) ont alors les mêmes performances avec et sans utilisation de l'heuristique, et les groupes de rayons incohérents ($S_{fn} \geq 45$) ont le même facteur d'accélération qu'avec l'évaluation précise de la proportion de rayons actifs. Bien que les données statistiques suivent une évolution similaire entre l'utilisation de la valeur approchée et de la valeur précise, on peut remarquer l'apparition de paliers qui traduisent une adaptation moins progressive de l'heuristique. Ce phénomène s'explique par l'utilisation de valeurs globales et non locales qui font qu'entre deux tailles proches de faisceaux, un ensemble de boîtes peut basculer de l'exécution de groupes de rayons à l'exécution de rayons SIMD.

L'utilisation de l'heuristique de surface avec la valeur approchée de la proportion de rayons actifs $P'_{ra/n}$ est donc un moyen efficace d'exploiter la cohérence partielle dans un groupe de rayons incohérent. Sa mise en place permet d'une part d'accélérer les performances à partir d'une seuil d'incohérence, et d'autre part d'exploiter tout groupe de rayons sans risque de perte de performances. Cette dernière propriété est intéressante quand il s'agit de regrouper automatiquement des rayons dont l'organisation n'est a priori pas cohérente.

2.3 Regroupement automatique des rayons

2.3.1 Contexte

L'heuristique de surface est un moyen efficace de limiter la dégradation de performances liée à l'incohérence des rayons. Les deux principaux avantages de cette approche sont, d'une part de pouvoir créer de très larges groupes de rayons, et d'autre part d'avoir un facteur de performance optimisée selon la réelle cohérence sous-jacente des rayons. Cette dernière propriété implique qu'il reste nécessaire de consacrer des efforts à un agencement cohérent des rayons, mais que le facteur même de cohérence n'est plus un point limitatif de l'utilisation d'un groupe de rayons.

En se plaçant dans le contexte de la production d'images de synthèse, la majorité des moteur de rendus utilisent essentiellement des requêtes d'intersection simples. Plusieurs raisons peuvent expliquer ce choix d'implémentation. D'une part toute la chaîne de traitement graphique est construite de telle manière que le point orienté (avec toutes ses propriétés associées) est la primitive de base manipulée. L'ensemble

des fonctions de calcul d'intervisibilité, d'illumination locale et globale, et d'application des matériaux, prennent en entrée un point orienté. Changer toute cette chaîne graphique pour prendre en compte la possibilité de regroupement de rayons est lourd de conséquences. De plus, la complexité des effets de rendus (ou d'autres domaines physiques) est susceptible de générer des rayons dont la cohérence est difficile à exploiter. Ce manque de cohérence diminue l'intérêt des groupes de rayons, car le gain en performances serait alors faible, voire négligeable. Il est donc intéressant pour répondre à un besoin particulier de performances et de simplicité d'accélérer de manière transparente les requêtes d'intersection simples. La seule contrainte serait alors de différer le traitement du résultat d'une requête d'intersection.

Plusieurs travaux [3], [55], [53] ont déjà démontré qu'il était possible de dissocier l'exécution d'un rayon du traitement de son résultat. Un grand nombre de rayons est alors généré et mémorisé dans une structure spéciale. Cette structure permet alors de réorganiser les rayons pour en accroître la cohérence puis de les traiter individuellement. Cependant, les objectifs ne sont pas de réduire le nombre d'étapes de traversée ou le nombre de calculs d'intersection, mais essentiellement d'améliorer la localité d'accès aux données géométriques et de rendu comme les textures. Les travaux d'Arvo *et al.* font exception car la réorganisation des rayons est utilisée pour remplacer la construction d'une structure de recherche. Dans le même ordre d'idée de ces travaux, la réorganisation des rayons est explorée afin de déterminer des sous-ensembles de rayons cohérents.

2.3.2 Objectif

Le principe de la réorganisation des rayons doit être simple dans son utilisation et générique. Pour cela, la fonction de requête d'intersection est scindée en deux fonctions. La première fonction permet de définir les paramètres d'un rayon et son exécution renvoie un identificateur unique de rayon. La deuxième fonction permet de récupérer le résultat de l'exécution d'un rayon grâce à son identificateur.

De manière interne, à chaque fois qu'un rayon est défini, celui-ci est mémorisé dans une liste de rayons. Tant qu'il existe des rayons à définir, cette fonction doit être appelée dans la mesure où il est plus probable de trouver des groupes cohérents quand l'ensemble initial de rayons est grand [78]. Lors du premier appel de la fonction de récupération des résultats d'un rayon quelconque, tous les rayons listés sont triés pour former un ou plusieurs groupes de rayons le plus cohérent possible. Ces groupes sont alors exécutés tour à tour et les résultats mémorisés. La liste de rayons à traiter est alors vidée et prête à recevoir de nouvelles définitions de rayons.

Ce principe de requête différée offre l'avantage de pouvoir être intégré facilement dans une application complexe. Il n'est pas utile de se soucier de l'organisation, de la cohérence ou du nombre de rayons à traiter simultanément. Toute la réorganisation est réalisée de manière automatique. Un gain de performance est obtenu si de la cohérence a été exploitée et dans le pire des cas, la performance n'est pas dégradée. Le seul bémol de cette approche comme le souligne Navratil [53] est que le temps de réorganisation des rayons $T_{Reorganisation}$ est un temps additionnel qu'il est absolument

nécessaire de compenser par une performance meilleure en respectant la condition suivante :

$$T_{Reorganisation} + T_{ExecutionGroupe} \leq T_{ExecutionSimple}$$

Dans le but de concentrer les efforts sur une réorganisation efficace, l'approche en cours s'est focalisée sur un objectif plus simple :

$$T_{ExecutionGroupe} < T_{ExecutionSimple}$$

Le temps de réorganisation est donc négligé et son étude reportée. Il sera bien sûr vital pour valider l'approche dans son ensemble d'optimiser voire de relâcher cette réorganisation ; des premiers éléments de solution sont apportés dans le paragraphe 2.4 à la page 150.

2.3.3 Réorganisation des rayons

2.3.3.1 Travaux précédents La réorganisation des rayons doit permettre de créer des groupes de rayons dont la cohérence est maximale. Arvo *et al.* [3] proposent de trier les rayons dans la structure d'un arbre 5D. Chaque rayon est représenté par sa position et sa direction. Pour simplifier le codage de la direction, Arvo normalise celle-ci par la valeur de sa plus grande composante. Seules les deux autres composantes u et v sont alors utilisées. L'axe et le signe de la direction principale génèrent six cas différents traités comme 6 arbres 5D distincts. Pour raffiner un arbre 5D, chaque axe x, y, z, u et v est tour à tour subdivisé en deux parties égales. Le critère d'arrêt est contrôlé par un nombre maximal de rayons dans les feuilles, ou une taille maximale de l'arbre. L'approche d'Arvo *et al.* est intéressante, cependant elle ne construit pas des groupes de rayons de manière optimale en subdivisant cycliquement chaque axe. En effet, des rayons ayant des origines très proches et/ou des directions parallèles peuvent être dissociés à tort. De plus, la cohérence n'est pas forcément liée à la proximité des origines ou au parallélisme des directions. Comme cela a été montré, la cohérence est fonction de la surface de coupe d'un faisceau (voir la figure V.10).

Pharr *et al.* [55] avaient pressenti qu'il ne suffisait pas de regrouper les rayons par origine, ou par direction, pour obtenir des groupes de rayons cohérents. Pour cela, une structure d'organisation est utilisée, au dessus de la structure d'accélération, pour propager les rayons. Chaque feuille de cette structure mémorise tous les rayons y accédant. Quand tous les rayons sont affectés à une feuille, chaque feuille est traitée en lançant individuellement les rayons dans la structure d'accélération. L'ordre d'exécution des feuilles est choisi de manière à maximiser les plus grands nombres de rayons ainsi qu'une certaine capacité à en générer d'autres. Les rayons n'ayant pas trouvé de point d'intersection sont ensuite propagés dans la feuille suivante de la structure d'organisation. Navratil *et al.* [53] reprennent la même approche en remplaçant les grilles utilisées par Pharr *et al.* par un kd-Tree à deux niveaux. Ces approches sont difficilement exploitables avec des groupes de rayons pour les raisons suivantes :

- la cohérence n'est pas maximisée pour les groupes de rayons dans leur ensemble, mais de proche en proche selon un premier niveau d'organisation assez grossier ;
- un groupe de rayons pour une feuille donnée de la structure d'organisation n'est pas forcément maintenu dans la feuille suivante. Ce changement de groupes impliquerait de reconstruire, pour chaque feuille de la structure d'organisation traversée, le faisceau encadrant du nouveau groupe, ce qui pénaliserait les performances ;
- le parcours dans la structure d'organisation doit être réalisé individuellement pour chaque rayon. Il n'est alors pas possible de regrouper des étapes de traversées durant cette phase de calculs. D'autre part, si l'on suppose que la structure d'organisation est très grossière et que ce nombre d'opérations peut être considéré comme négligeable, il en résulte que les rayons situés dans les feuilles sont trop nombreux et ne peuvent alors pas présenter une cohérence suffisante ;

2.3.3.2 Principe L'approche retenue utilise, comme les travaux précédents [3, 53], une structure arborescente pour regrouper les rayons, mais exploite additionnellement la surface de coupe de chaque groupe de rayons pour guider et optimiser la construction.

Dans une première étape, chaque rayon est ajusté à la boîte englobante du modèle afin de déterminer les points extrêmes. Les rayons n'ayant pas d'intersection avec le modèle ne sont pas retenus. Dans son état initial, l'ensemble des rayons est représenté par deux boîtes englobantes :

- B_o : celle contenant les points origine des rayons ;
- B_d : celle contenant les points destination des rayons ;

Les rayons sont ensuite subdivisés en deux sous groupes. Afin d'évaluer plusieurs répartitions, une subdivision sur chaque axe est réalisée pour la boîte B_o ou B_d présentant la plus grande dimension, en plaçant un plan séparateur au centre des points. Chaque répartition génère quatre sous-boîtes dont $SB1_o$ et $SB1_d$ pour le premier sous-groupe, et $SB2_o$ et $SB2_d$ pour le second sous-groupe (voir la figure V.16). Une fonction de coût $C_{subdivision}$ est utilisée pour déterminer la meilleure répartition en évaluant le rapport entre la surface de coupe et le nombre de rayons N_{sg1} et N_{sg2} de chaque sous-groupe tel que :

$$\begin{aligned} C_{sg1} &= \frac{surface(SB1_o) + surface(SB2_d)}{N_{sg1}} \\ C_{sg2} &= \frac{surface(SB2_o) + surface(SB2_d)}{N_{sg2}} \\ C_{subdivision} &= C_{sg1} + C_{sg2} \end{aligned}$$

La répartition présentant le meilleur coût de subdivision est retenue, et l'algorithme de subdivision s'arrête quand un nombre maximal de rayons prédéfini est atteint.

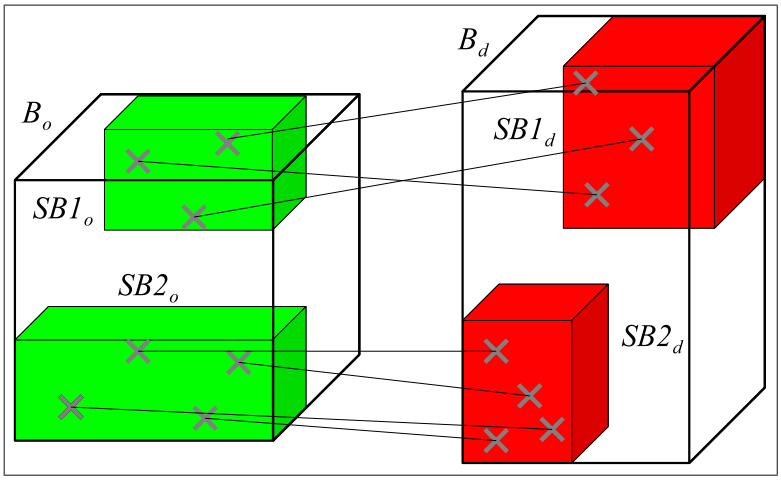


FIG. V.16 – Subdivision d'un groupe de rayons par séparation des extrémités.

2.3.4 Résultats préliminaires

Afin d'évaluer les performances du regroupement automatique de rayons, une série d'expérimentations a été réalisée en distinguant trois types de rayons : primaires, d'ombrage et d'occlusion ambiante. Les deux premiers types de rayons sont connus pour être cohérents par construction ; leur évaluation permet alors de valider le principe de la réorganisation pour des cas simples. Les rayons d'occlusion ambiante permettent de générer plusieurs niveaux d'incohérence selon la densité d'échantillonnage. La section 2 à la page 102 a montré qu'au delà de 200 rayons par point, l'occlusion ambiante bénéficiait des capacités de regroupement classique des rayons. En deçà, seuls des rayons SIMD offrent un apport intéressant par rapport aux rayons simples. Les expérimentations suivantes évaluent alors des densités plus faibles entre 4 et 64 rayons par point.

Le tableau V.2 présente des données statistiques de construction des arbres de regroupement de rayons. Afin de limiter la consommation mémoire et le temps de réorganisation, les regroupements sont réalisés pour chaque sous-ensemble de rayons issus d'une portion de 32^2 pixels de l'image. Chaque type de rayons, et chaque signe de direction, sont traités distinctement dans un arbre, ce qui génère pour une image de 512^2 pixels un peu plus de deux milles arbres. En définissant un nombre maximal de rayons par feuille à 128, on obtient en moyenne des groupes de 87 à 90 rayons ce qui est proche de la taille de groupe optimal pour l'arbre BVH. On constate logiquement que le nombre de feuilles, et la profondeur moyenne des arbres, augmentent avec le nombre de rayons à traiter mais n'influent pas sur le nombre moyen de rayons par feuille.

Rayons	$\#A$	$\#F/\#A$	$\#R/\#F$	$\overline{P}$	$\overline{S_{fn}}$
P + O + OA 4r/p	2109	10,0	87	3,6	87
P + O + OA 16r/p	2119	26,5	89	5,2	70
P + O + OA 36r/p	2123	54,4	89	6,5	55
P + O + OA 64r/p	2125	92,0	90	7,3	47

Tab. V.2 – Données statistiques collectées lors de la réorganisation des rayons pour différents types de rendus du modèle *Conference* en 512^2 pixels. Dans chaque cas, des rayons primaires, d'ombres et d'occlusion ambiante sont générés. $\#A$ désigne le nombre d'arbre construits. $\#F/\#A$ désigne le nombre moyen de feuilles par arbre. $\#R/\#F$ désigne le nombre moyen de rayons par feuille. $\overline{P}$ désigne la profondeur moyenne des arbre, et $\overline{S_{fn}}$ désigne la moyenne des surfaces de coupe normalisée pour chaque groupe de rayons.

Chaque feuille d'un arbre correspond à un ensemble de rayons qu'il est possible de traiter par groupe. Le tableau V.3 montre des données statistiques issues de l'exécution de ces ensembles selon plusieurs tailles de groupes. Pour constituer des groupes de taille 4, exploitables avec des rayons SIMD, les rayons sont arbitrairement regroupés selon l'ordre établi lors de l'ajout. Bien qu'il aurait été possible de continuer la subdivision des arbres jusqu'à une taille plus adaptée, et augmenter en conséquence le temps de réorganisation, cette simple stratégie de regroupement permet de réduire, par rapport aux rayons simples, à moins de 40% le nombre d'étapes de traversées et à moins de 60% le nombre de calculs d'intersections. En traitant l'ensemble des rayons comme un groupe, le nombre d'étapes de traversée est efficacement réduit ($<$ 10%), mais le nombre de calculs d'intersections augmente considérablement jusqu'à doubler. L'utilisation de l'heuristique de surface pour limiter la divergence d'un faisceau, avec l'approximation de la proportion P' de rayons actifs, permet de maîtriser cet l'accroissement tout en maintenant une bonne réduction du nombre d'étapes de traversée inférieur à 20%.

Rayons		1	4		Max		$Max^{P'}$	
P + O + OA 4r/p	# T	34,7	12,9	(37%)	2,3	(7%)	6,2	(18%)
	# I	4,1	2,0	(49%)	6,4	(154%)	2,4	(57%)
P + O + OA 16r/p	# T	88,5	34,1	(39%)	5,6	(6%)	16,7	(19%)
	# I	8,9	5,1	(58%)	17,8	(201%)	5,9	(67%)
P + O + OA 36r/p	# T	178,2	67,2	(38%)	10,3	(6%)	30,0	(17%)
	# I	16,7	9,9	(59%)	33,7	(202%)	12,2	(73%)
P + O + OA 64r/p	# T	303,8	111,6	(37%)	16,4	(5%)	46,0	(15%)
	# I	27,8	16,3	(59%)	53,5	(193%)	20,6	(74%)

Tab. V.3 – Nombre d'étapes de traversée et de calculs d'intersection, en millions, pour différents types de rendus du modèle *Conference* en 512^2 pixels. L'exécution des rayons simples (1) est comparée à l'exécution de rayons SIMD (4), et aux groupes de rayons avec ($Max^{P'}$) et sans (Max) utilisation de l'heuristique de surface.

Le tableau V.4 présente les temps d'exécution suite au regroupement automatique des rayons. Trois résultats peuvent être dégagés de ces mesures : le temps de réorganisation, le temps d'exécution et le temps total. Le temps de réorganisation est acceptable pour des petits ensembles de rayons comme les rayons primaires, les rayons d'ombre et les rayons d'occlusion ambiante avec 4 évaluations par point, où il représente entre 10% et 15% du temps de traitement des rayons simples. Pour de plus grands groupes de rayons, ce temps représente entre 25% et 50% du temps de traitement des rayons primaires réduisant considérablement le gain global apporté par l'accélération des groupes de rayons.

En considérant le temps seul d'exécution, les rayons SIMD permettent d'accélérer les performances par un facteur x2 pour les rayons d'occlusion ambiante et x3 pour les rayons primaires et d'ombre. Les groupes de rayons apportent un facteur d'accélération additionnel (x6) pour les rayons primaires et d'ombre, par contre les rayons d'occlusion d'ambiante sont peu accélérés, voire dégradés, quand la densité d'échantillonnage est très faible (4 rayons par point d'évaluation). Ce dernier point est cependant corrigé par l'utilisation de l'heuristique de surface, qui accélère également les rayons d'occlusion ambiante, et présente notamment pour la densité de 4 rayons par point, un facteur d'accélération de x2,6.

Rayons	R : Réorg.	G : Groupes				$G_1/(R+G_N)$		
		1	4	Max	$Max^{P'}$	$N=4$	$N=Max$	$N=Max^{P'}$
P : Primaires	55	518	175	89	88	x2,0	x2,9	x2,9
O : Ombres	63	431	140	70	68	x2,1	x3,1	x3,2
OA 4r/p	404	2 707	1 287	1 419	1 055	x1,2	x1,1	x1,3
OA 16r/p	2 521	10 344	4 750	4 558	3 659	x1,2	x1,3	x1,4
OA 36r/p	7 387	22 745	10 378	9 158	7 771	x1,3	x1,3	x1,5
OA 64r/p	20 741	40 110	18 174	15 614	13 701	x1,3	x1,4	x1,5

Tab. V.4 – Temps d'exécution en millisecondes, pour différents types de rendus du modèle *Conference* en 512^2 pixels. Le temps R correspond au temps pris par la réorganisation des rayons. L'exécution des rayons simples (1) est comparée à l'exécution de rayons SIMD (4), et aux groupes de rayons avec ($Max^{P'}$) et sans (Max) utilisation de l'heuristique de surface.

Le temps total, bien que pénalisé par un temps de réorganisation assez conséquent, est toujours meilleur que l'exécution individuelle des rayons. Les meilleurs temps sont obtenus avec les groupes de rayons en utilisant l'heuristique de surface. Bien que la cohérence des rayons s'améliore avec la densité d'échantillonnage de l'occlusion ambiante (voir le tableau V.2), les performances sont fortement limitées par un temps de réorganisation qui augmente plus rapidement, limitant ainsi à un faible gain (x1,5) quand l'échantillonnage est de 64 rayons par point.

Le résultat le plus intéressant de ces expérimentations concerne l'accélération de l'occlusion ambiante avec un échantillonnage de 4 rayons par point. Aucune approche

classique ne permet de regrouper ce type de rayon que ce soit en groupe de 4 ou de plus grande taille. La réorganisation automatique des rayons permet alors, par l'exploitation d'un grand ensemble de rayons, de dégager une cohérence initialement inexistante.

2.4 Discussion

Nous nous sommes intéressés dans cette section, à exploiter les rayons sous forme de groupes créés automatiquement. Le but de cette automatisation est double : il s'agit d'une part d'étendre le potentiel des groupes de rayons quand la cohérence est faible, et d'autre part d'apporter une solution pour accélérer de manière transparente des outils conçus pour ne lancer que des rayons simples. Dans les deux cas, il a été nécessaire de dégager un indicateur de cohérence qui soit disponible avant l'exécution d'un groupe de rayons et rapide à calculer. Nous avons alors montré que la cohérence est liée à la surface de coupe moyenne S_f du faisceau encadrant d'un groupe de rayons, en rapprochant cette valeur aux temps d'exécution, et à des indicateurs fiables issus de données statistiques [50].

L'analyse des performances en fonction de S_{fn} a également permis de mettre en évidence qu'en dessous d'un certain seuil de cohérence, les groupes de rayons sont moins performants que des rayons SIMD. Pour remédier à ce problème, nous avons proposé une heuristique de surface qui, dans un cas d'incohérence révélée par l'évaluation approximative de la proportion de rayons actifs $P'_{ra/b}$, peut opter pour traiter une partie de la propagation avec des rayons SIMD. Cette heuristique permet alors de traiter la partie cohérente de la propagation par des groupes de rayons, et l'autre partie par rayons SIMD. La mise en place de cette heuristique permet d'une part d'éviter la pénalité apportée par les groupes de rayons incohérents au regard des rayons SIMD, et d'autre part d'améliorer la performance dès que de l'incohérence est partiellement présente.

Afin de générer automatiquement des groupes de rayons à partir d'un ensemble de rayons simples, nous nous sommes inspirés des travaux de Arvo *et al.* [3] pour trier les rayons par la construction d'une structure arborescente. Nous avons proposé une nouvelle approche en dissociant les extrémités des rayons dans deux boîtes englobantes, et subdivisant récursivement ces boîtes de manière à minimiser la surface de coupe moyenne des sous-groupes de rayons formés. La qualité des groupes de rayons, issus des feuilles de cet arbre, permet alors, si l'on ne considère que le temps d'exécution, d'accélérer par un facteur proche de x6 les rayons cohérents et par un facteur proche de x3 des rayons incohérents. De plus, la réorganisation a permis de traiter et d'accélérer des rayons omnidirectionnels très incohérents (4 rayons dans un champ hémisphérique) en exploitant la cohérence globale d'un large ensemble de rayons.

Cependant, bien que toujours favorable, l'amélioration des performances baisse de moitié si l'on considère dans l'équation globale le temps pris par la réorganisation des rayons. Maintes améliorations sont envisageables pour accélérer ce temps de réorganisation. L'implémentation actuelle du regroupement automatique a été réalisée

sous la forme d'un prototype en privilégiant la qualité de la construction à sa rapidité. En ayant pour objectif d'accélérer des rayons simples de manière transparente, les interfaces d'entrée et de sortie qui permettent d'ajouter un rayon à traiter puis de récupérer le résultat d'intersection de ce rayon sont entièrement scalaires. De manière interne, le passage en rayons SIMD plus optimisés se fait tardivement dans l'enchaînement des opérations de réorganisation, et gagnerait à être remonté au plus tôt dès l'ajout des rayons simples. D'autre part, la même structure de données est utilisée pour définir les paramètres d'un rayon et mémoriser le résultat de son exécution ; en dissociant plusieurs types de données, on pourrait gagner en localité des calculs. Enfin, certains calculs effectués dans la phase de tri des rayons sont redondants avec des calculs réalisés durant l'exécution des rayons. La nécessité d'inverser la direction des rayons ou bien d'intersecter les rayons avec la boîte englobante du modèle sont, par exemple, des opérations relativement coûteuses qui pourraient être factorisées en ayant une meilleure intégration. De nature algorithmique, le temps de réorganisation des rayons croît en $O(N\ log\ N)$ en fonction du nombre de rayons ; il peut alors être intéressant de limiter automatiquement la quantité de rayons dans un arbre, et de reporter les rayons excédentaires dans un ou plusieurs autres arbres.

En terme de facilité d'utilisation, le regroupement automatique des rayons nécessite de dissocier la définition d'un rayon, du traitement de son résultat. Nous pensons que cette contrainte est beaucoup moins forte, et moins délicate à supporter dans une chaîne de production, que la nécessité de regrouper "manuellement" des rayons souvent situés dans des étapes distinctes de traitement. Une approche en deux passes, en séparant en deux fonctions les opérations réalisées avant le traitement d'un ou plusieurs rayons, de celles réalisées après, pourrait suffire dans de nombreux cas, à supporter le différé d'exécution. En extrapolant l'automatisation, le regroupement automatique de rayons peut être utilisé avantageusement, et en toute transparence, par des compilateurs qui supportent des langages de programmation dédiés au lancer de rayons comme RTSL (Ray Tracing Shading Language) [54].

Conclusion

Dans ce chapitre, nous avons proposé deux approches différentes pour traiter de la performance des groupes de rayons quand il y a perte de cohérence. Dans la première approche, nous avons proposé une stratégie originale de couplage du kd-Tree et de l'arbre BVH qui choisit, selon le type de rayon, la structure de recherche la plus adaptée au traitement efficace d'un groupe de rayons. Dans la deuxième approche, nous avons limité notre étude à l'arbre BVH, et avons montré, par l'intermédiaire d'un prototype restant à améliorer, qu'il est possible de réorganiser automatiquement un large ensemble de rayons incohérents dans le but de trouver des sous-ensembles plus cohérents.

Le couplage des structures de recherche s'appuie sur la constatation simple que les larges groupes de rayons présentant de la cohérence sont plus efficaces avec l'arbre BVH, et que les rayons SIMD, qui ont un facteur d'accélération relativement stable

même en cas d'incohérence, sont traités plus rapidement avec le kd-Tree. Les rayons lancés dans un rendu de synthèse d'images sont de natures variées, et un grand nombre d'entre eux, notamment ceux participant aux calculs de l'illumination globale, sont incohérents. Les groupes de rayons de l'arbre BVH, initialement très performants, ne sont alors plus exploitables, et nous proposons comme solution de remplacement de traiter ces rayons dans une approche couplée avec le kd-Tree. Une stratégie globale est alors utilisée pour choisir, en fonction de la cohérence induite par le type de rayon, la structure la plus adaptée à leur traitement. Nous montrons que le couplage de l'arbre BVH et du kd-Tree, pour un rendu de type lancer de rayons agrémenté d'occlusion ambiante, permet d'accélérer les performances globales jusqu'à un facteur x1,5.

Sans remettre en cause la stratégie de couplage, mais dans le but de contrôler plus finement et déplacer la limite entre cohérence et incohérence, nous avons montré que, la surface moyenne de coupe du faisceau encadrant un groupe de rayons, est un indicateur fiable de mesure de la cohérence. Cet indicateur offre l'avantage d'être facile à calculer et disponible a priori. Il permet d'une part, grâce à l'utilisation d'une heuristique, d'exploiter la cohérence partielle dans un groupe de rayons globalement incohérent, et d'autre part, d'être le critère de coût à minimiser quand l'on souhaite regrouper des rayons automatiquement.

L'intérêt du regroupement automatique des rayons est de pouvoir accélérer des rayons dont la cohérence n'est pas évidente, mais également de pouvoir être intégré dans une chaîne de production exploitant essentiellement des rayons simples. Nous avons montré, qu'en triant les rayons dans une structure arborescente, et en guidant cette construction par la minimisation de la surface moyenne de coupe, que la méthode permet de former des groupes de rayons dont la cohérence est suffisante pour fournir une accélération significative par rapport aux rayons simples, qui peut par ailleurs être bien meilleure moyennant quelques efforts d'optimisation.

Pour mettre en place le regroupement automatique des rayons, il est nécessaire, par rapport à une approche itérative traditionnelle, de dissocier la définition des paramètres d'un rayon de l'exploitation de son résultat. Cette dissociation, bien que contraignante, est indispensable pour construire un large ensemble de rayons que l'on puisse réorganiser efficacement. D'autres travaux montrent que cette dissociation est intéressante et praticable, notamment pour améliorer, par la réorganisation, la localité d'accès à la géométrie [50]. Nous pensons également que cette réorganisation est intéressante pour traiter des modèles complexes, qui ne peuvent être que partiellement chargés en mémoire, dans le but de masquer les latences d'accès aux données géométriques localisées sur un support de masse. Le chapitre suivant traite en détail de cette problématique, et propose une solution de réorganisation des rayons pour traiter des modèles complexes dans un environnement de calculs distribués.

Chapitre VI

Réorganisation et distribution des rayons pour les modèles complexes

Introduction

Un modèle complexe peut être défini comme un modèle qui ne peut être traité de manière standard sur une seule machine. Deux principales raisons, non exclusives, peuvent rendre un modèle complexe :

- une grande complexité physique impliquant de lancer un très grand nombre de rayons dont le temps d'exécution ne serait pas acceptable sur une seule machine ; un calcul d'illumination globale peut représenter ce type de complexité.
- une grande complexité géométrique impliquant un temps d'exécution par rayon significativement plus important et une occupation mémoire supérieure à celle d'une seule machine ; avec 1 Go de mémoire centrale, un modèle de 100 millions de triangles est, par exemple, 8 fois trop volumineux.

Bien qu'une grande complexité physique ne soit pas un facteur bloquant mais juste un frein à l'interactivité, la capacité à traiter des modèles volumineux est quant à elle une réelle limitation. Une solution économiquement viable pour traiter ces modèles géométriquement complexes consiste à distribuer les calculs sur une grappe de calculateurs afin de bénéficier d'une puissance de calculs et d'une capacité mémoire accrues et suffisantes.

On peut remarquer que la notion de complexité est un seuil qui évolue avec le temps. Cette étude située chronologiquement au début de la thèse prenait tout son sens avec l'incapacité des architectures 32 bits à adresser plus de 4 Go de mémoire. Depuis, les architectures 64 bits ont considérablement repoussées cette limite, bien qu'elle soit en pratique limitée par la capacité des cartes mères dont le haut de gamme actuel supporte entre 32 Go et 128 Go de mémoire vive. Néamoins, les techniques mises en place restent valides aujourd'hui pour tout modèle dépassant la capacité mémoire.

L'utilisation de l'algorithme du lancer de rayons pour rendre de très grands modèles dépassant la capacité mémoire d'une machine, et sa distribution efficace sur

une grappe de calculateurs standard, est un travail difficile qui pose principalement deux types de problèmes. Le premier est lié d'une manière générale à l'architecture distribuée utilisée, tandis que le second est dû aux accès irréguliers aux données par le lancer de rayons.

Les grappes de calculateurs sont des architectures qui offrent une puissance de calcul cumulée très importante à faible coût. Cependant, elles souffrent d'une faible bande passante et d'une forte latence lors de la communication des données. L'utilisation des mémoires distribuées implique une restructuration des algorithmes en vue de maximiser un traitement local des données. Contrairement aux machines massivement parallèles (SMP), l'utilisation de l'ensemble des mémoires, soit leur agrégation, passe obligatoirement par des échanges sur le réseau. Pour traiter de très grands modèles, il est donc nécessaire soit de procéder à des échanges entre les différentes mémoires via le réseau, soit d'exploiter localement le disque dur comme une extension de la mémoire. Ces deux points sont des contraintes fortes car les différences de débits et de latences existantes entre la mémoire centrale, et le réseau ou le disque, rendent les applications essentiellement limitées par l'accès aux données.

Du point de vue algorithmique, le lancer de rayons accède aux différentes parties du modèle de façon très irrégulière. Quand la profondeur de récursion et le nombre de rayons secondaires sont importants, les mêmes données sont sollicitées plusieurs fois au cours du rendu et ce de manière imprévisible. Par exemple, les sources de lumière ou les surfaces réfléchissantes engendrent un grand nombre de traversées globales des rayons dans le modèle.

La première section de ce chapitre présente la mise en place d'une pile de rayons, par opposition au lancer de rayons classique récursif, pour rendre des modèles complexes sur une seule machine. Cette pile de rayons permet de masquer les latences d'accès au support de masse, en différant le traitement des requêtes d'intersection dont les données ne sont pas chargées en mémoire. Après avoir montré la validité de la solution, une étude de la distribution de cette pile de rayons est présentée dans la seconde section. Deux topologies de distribution sont envisagées : soit en dupliquant le modèle sur chaque machine de la grappe, ou soit en répartissant le modèle dans l'ensemble des mémoires de la grappe.

1 Mise en place d'une pile de rayons avec un calculateur

1.1 Problématique

Par rapport à une technique de lancer de rayons classique, le traitement des très grands modèles se distingue par l'impossibilité de charger entièrement un modèle en mémoire centrale. Cette limitation implique de prétraiter les données géométriques afin d'en faire des blocs : des éléments indépendants les uns des autres qui suffisent à l'exécution partielle des calculs. La problématique essentielle, dans le but de maxi-

miser la performance, est de déterminer les blocs les plus propices à l'avancée des calculs, et de minimiser le temps de chargement de ces blocs.

Comme celà a déjà été évoqué à plusieurs reprises, Pharr *et al.* [55] proposent un système de gestion des grands modèles en réorganisant le parcours des rayons. Le modèle est prétraité en amont en voxels par l'intermédiaire d'une grille régulière. Ces voxels sont enregistrés dans un fichier et constituent les lignes de cache directement chargeables en mémoire ; ils contiennent chacun environ un millier de triangles. Chaque voxel contient également une grille d'accélération pour le lancer de rayons dont les sous-voxels référencent quelques centaines de triangles. Les voxels sont chargés en mémoire selon les besoins des calculateurs ce qui permet d'éviter le chargement de voxels inutiles.

Le principe de ce système est fondé sur un réordonnancement dynamique des rayons afin de maximiser l'utilisation des voxels déjà chargés en cache. Pour celà, Pharr *et al.* créent une grille de gestion au-dessus des voxels de données ; à chaque voxel de cette grille de gestion sont associés les rayons qui n'ont pu se propager car bloqués par un défaut de cache. Les voxels de gestion et les rayons associés sont ordonnés selon une fonction de coût/bénéfice qui favorise l'utilisation des données déjà chargées en cache et la propagation des rayons. Le système gère en parallèle un cache de rayons limité en taille ; les rayons excédentaires sont alors écrits sur disque. Les résultats montrent que le système est capable, grâce une très bonne utilisation de la localité des données, de traiter des modèles dix fois plus grands que la mémoire centrale disponible. La solution proposée semble intéressante bien que le réordonnancement des rayons peut être une procédure coûteuse et profite essentiellement à des rayons qui sont très cohérents.

Plusieurs idées comme l'organisation et le chargement paresseux des données sont retenues de cette approche. Cependant, dans les travaux de Pharr *et al.* , lorsqu'un voxel est élu candidat au parcours d'un ensemble de rayons, le chargement de celui-ci est réalisé de manière synchrone empêchant potentiellement d'autres calculs, en parallèle, avec des voxels déjà chargés en mémoire. L'approche proposée se différencie essentiellement sur ce point en masquant, par du calcul utile, les temps d'accès au support de masse qui peuvent représenter la moitié du temps total de rendu (voir le tableau VI.4).

L'organisation des données géométriques en blocs, issus de la construction d'un arbre octal, est présentée dans un premier temps. Ensuite, la mise en place d'une pile de rayons, pour différer les requêtes d'intersection en attente d'un chargement de bloc, est présentée. Les détails d'implémentation dans un contexte multiprocesseur sont également décrits, avant de discuter finalement des résultats de la solution générale avec un calculateur.

1.2 Découpage de la géométrie en blocs

L'utilisation de très grands modèles implique d'adapter les structures de données pour qu'elles puissent être manipulées partiellement et indépendamment. Chaque

partie du modèle peut alors être chargée en mémoire centrale et exploitée localement tant qu'elle contribue aux calculs. La mémoire centrale joue alors un rôle de *cache* pour le disque dur.

En prenant expérience de travaux antérieurs [15], il est nécessaire que les parties chargeables en mémoire soient cohérentes spatialement. Pour parvenir à ce découpage, un arbre octal [25] est utilisé comme structure de gestion et comme structure d'accélération, selon deux niveaux de hiérarchie (voir la figure VI.1). Le premier niveau constitue des blocs de données géométriques chargeables en mémoire, tandis que le second constitue une finesse d'arborescence suffisante pour accélérer la recherche des intersections. On peut remarquer, qu'une structure arborescente de type BVH ou kd-Tree aurait pu également être utilisée, mais, bien que non optimal en terme de performance de la requête d'intersection, l'arbre octal offre l'avantage d'être plus rapide à construire. De plus, étant donnée la problématique principale du temps de chargement des blocs, le temps seul de la requête d'intersection est une préoccupation moins importante.

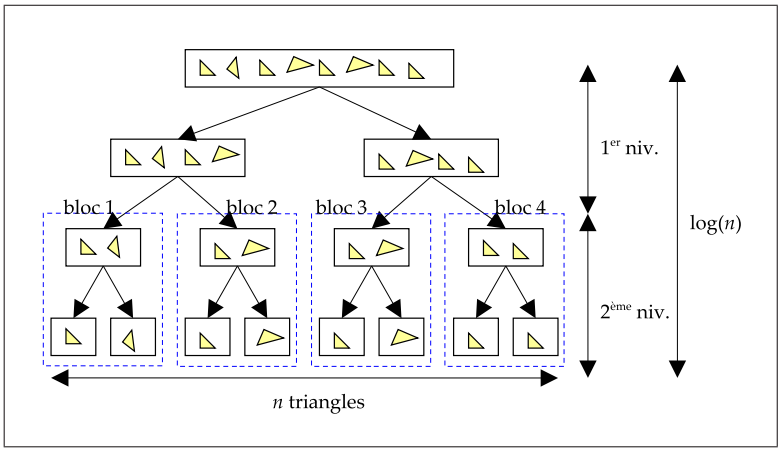


FIG. VI.1 – Construction d'un arbre octal à 2 niveaux.

Initialement, seul le premier niveau de la hiérarchie est entièrement chargée en mémoire. Les blocs, représentés par les feuilles, sont ensuite chargés dynamiquement dans une mémoire cache, lors du parcours de l'arbre octal. Chaque bloc contient alors un sous-arbre permettant de propager les rayons jusqu'aux surfaces.

Cette mémoire "cache" est de type purement associative avec une politique de déchargement des données LRU (Least Recently Used). Afin d'obtenir un bon compromis entre le débit de lecture, le taux de redondance des références de triangles dans les blocs et la finesse de granularité du cache, la taille des blocs est limitée par construction. Dans l'ensemble des expérimentations la taille maximale de 128 Ko a été choisie pour obtenir en moyenne une taille de bloc de 32 Ko (voir le tableau VI.1).

Modèle	Taille maximale bloc	# blocs	Taille moyenne bloc	Taille blocs
Chevrolet	32 Ko	264	8,2 Ko	2,2 Mo
	64 Ko	120	16,2 Ko	1,9 Mo
	128 Ko	83	22,4 Ko	1,9 Mo
	256 Ko	53	33,5 Ko	1,8 Mo
Engine	32 Ko	1 158	10,4 Ko	12,0 Mo
	64 Ko	555	19,4 Ko	10,8 Mo
	128 Ko	275	36,1 Ko	9,9 Mo
	256 Ko	117	79,1 Ko	9,3 Mo

TAB. VI.1 – Nombre et taille moyenne des blocs en fonction du paramètre de taille maximale.

Afin que la structuration des données soit compacte et rapide à parcourir, durant la phase de prétraitement, des fichiers d'index, de voxels et de blocs sont générés. Le fichier d'index est chargé intégralement en mémoire au début du rendu et permet de se positionner instantanément dans le fichier des blocs. Le fichier de voxels, correspond au premier niveau de l'arbre octal, et est également totalement chargé en mémoire. Le fichier des blocs contient pour chaque feuille du premier niveau les surfaces et le sous-arbre.

La section suivante décrit comment sont chargés les blocs en mémoire durant un rendu par lancer de rayons.

1.3 Empilement des rayons

L'aspect récursif traditionnel du lancer de rayons, en profondeur puis en largeur, ne permet pas de contrôler l'ordre d'accès aux données. Une implémentation synchrone de cet algorithme, pour supporter l'organisation en blocs de géométrie localisés sur un support de masse, serait de charger ceux-ci en fonction des sollicitations des requêtes d'intersection, et d'attendre la lecture effective de chaque bloc avant de poursuivre la propagation des rayons dans les sous-arbres. Cependant, cette synchronisation sur la lecture des blocs empêche de masquer les grandes latences d'accès aux disques.

1.3.1 Principe

L'ordre d'exécution des rayons peut être représenté comme un arbre de dépendances entre les rayons (voir la figure VI.2). Il n'est pas possible de traiter les rayons fils sans avoir traité les rayons pères préalablement. Cependant, il est possible de traiter les rayons fils dans un ordre quelconque sans influer sur le résultat des calculs. La figure VI.2 montre, par un exemple simple, la réduction du nombre d'accès aux données quand les rayons sont traités dans un ordre plus approprié.

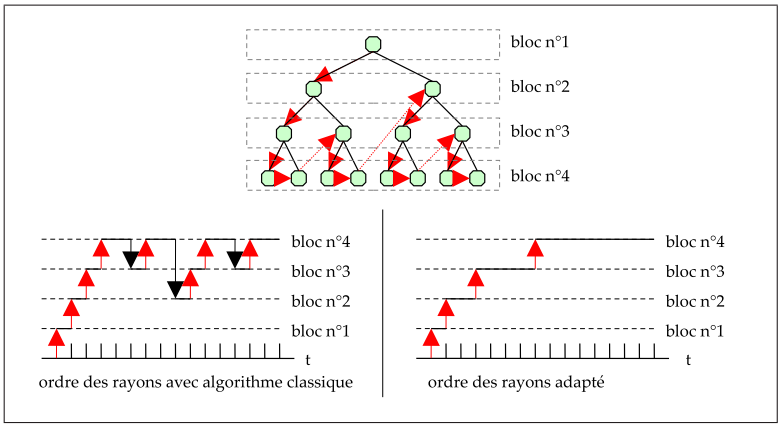


FIG. VI.2 – Apport d'une exécution ordonnée des rayons pour un cache de taille 1.

```
//Tant que tous les rayons ne sont pas traités
while (!(rayStack.empty() && toloadBlockQ.empty() && loadedBlockQ.empty()) {
  //recuperation en priorité des blocs récemment chargés pour
  //alimenter la pile principale de rayons
  Block *b;
  while (b = loadedBlockQ.pop()) rayStack.addQ(b->waitingRayQ);
  //s'il y a un rayon alors traitement
  Ray *r;
  if (r = rayStack.pop()) {
    //recherche du prochain bloc impacte
    if (!(b = tree.getNextBloc(r))) continue;
    //traitement des rayons + ajout des rayons secondaires
    if (b->state == LOADED) r->process();
    else { //mise en attente du rayon pour ce bloc
      b->waitingRayQ.push(r);
      //demande de chargement du bloc
      if (b->state != LOADING) {
        b->state = LOADING;
        toloadBlockQ.push(b);
      }
    }
  }
}
```

FIG. VI.3 – Traitement de la pile de rayons.

Dans l'optique d'effectuer un traitement asynchrone des rayons, il est nécessaire de connaître ceux susceptibles d'être exécutés en remplacement d'un rayon bloqué par un accès aux données. L'utilisation d'une pile de rayons, en distinguant une pile principale et une pile d'attente par bloc, permet une mise en place simple de ce principe (voir la figure VI.3). Les rayons primaires alimentent directement la pile principale, dans laquelle sont retirés unitairement chaque rayon pour traitement. Si tous les blocs de données, nécessaires au traitement d'un rayon, sont préalablement chargés dans la mémoire cache, alors le rayon est traité et les rayons fils, s'ils existent, sont rajoutés dans la pile principale. Si un rayon ne peut pas être exécuté, en raison d'un défaut de cache, alors ce rayon est ajouté à la pile des rayons en attente du bloc manquant. Ce rayon sera remis dans la pile principale, ainsi que tous les autres rayons en attente de ce même bloc, lorsque ce dernier sera effectivement chargé.

L'algorithme présenté dans la figure VI.3 fonctionne de pair avec un thread de chargement des blocs. Ce thread s'occupe de lire en continu les blocs référencés dans la file *toloadBlockQ*, puis les ajoute, lorsque la lecture est complète, dans la file *loadedBlockQ*. La mise en place de cette pile de rayons contribue de deux façons complémentaires aux performances du moteur de lancer de rayons. Elle permet d'une part de faire du calcul efficace car les latences d'accès aux données sont masquées par le traitement d'autres rayons ; d'autre part, elle permet de mettre en place une politique de sélection des rayons dans le but de maximiser un objectif. Il est possible par exemple de chercher à minimiser le nombre de chargement des blocs, ou encore de visualiser prioritairement certaines parties de l'image (voir la section 2.4.2 à la page 173).

1.3.2 Occupation mémoire

Etant donné qu'un rayon peut être différé à tout moment, il doit comporter des informations pour sauvegarder son contexte de calcul. Le tableau VI.2 recense les informations nécessaires au rendu. En dehors des caractéristiques classiques d'un rayon, il est nécessaire de mémoriser des informations pour le calcul de l'illumination et la traversée des voxels. Le calcul des couleurs de l'image finale ne nécessite aucune phase supplémentaire au cours du rendu car chaque rayon ajoute sa contribution à l'illumination d'un pixel. On peut remarquer que le rayon mémorise les 8 prochains blocs à parcourir. Ce nombre de blocs a été empiriquement déterminé comme offrant le meilleur compromis entre la recherche exhaustive de tous les blocs à parcourir, et la recherche de proche en proche du prochain bloc à parcourir.

Variables	Type	Taille (en octets)	Description
Données principales			
Type	Entier	1	Type de rayon (Primaire, Ombre, Secondaire, Anti-Aliasing)
Depth	Entier	1	Profondeur de récursion
Org	Vecteur	12	Origine
Dir	Vecteur	12	Direction
Données utiles aux calculs d'illumination			
Pixel	Pointeur	4	Pointeur sur le pixel originel
Att	Vecteur	12	Coefficient d'atténuation RGB
Object	Pointeur	4	Pointeur sur l'objet émetteur du rayon (Caméra, Surface, Lumière)
Données utiles à la traversée des blocs			
Blocks	8 pointeurs	32	Liste des *NbBlock* prochains blocs à traverser
CurBlock	Entier	1	Index du bloc en cours de traitement (ou en attente de chargement)
NbBlock	Entier	1	Nombre de blocs (≤ 8)

TAB. VI.2 – Informations nécessaires pour mémoriser l'état d'un rayon.

Le principal désavantage de la pile de rayons est lié à son utilisation intensive de la mémoire, tant en nombre qu'en taille des allocations. Pour remédier à une occupation excessive de la mémoire, la pile principale de rayons (voir la figure VI.3) est initialisée avec un sous-ensemble des rayons primaires. Dans ce cas, le rendu global de l'image est obtenu après plusieurs appels de la fonction, et limite l'occupation de la mémoire. Il existe cependant un compromis entre le nombre de rayons dans la pile, l'efficacité de rendu et l'occupation mémoire (voir le tableau VI.3). En effet, plus la pile principale est grande, et plus il est possible de recouvrir les temps de chargements par des temps de calcul.

Pour éviter la fragmentation de la mémoire, les rayons sont alloués par paquet de 32, occupant ainsi une taille de 2 Ko proche de la taille d'une page mémoire. Un paquet ne peut alors être libéré de la mémoire si et seulement si tous ses rayons sont traités. Une fonction de nettoyage (garbage) scrute régulièrement les paquets vides et libère la mémoire inutilisée. Il s'avère cependant, sans exécuter la fonction de nettoyage, que l'on puisse réutiliser la mémoire des paquets vides pour la création des nouveaux paquets de rayons. L'occupation mémoire de la pile des rayons est donc acceptable, et ne dépasse rarement quelques dizaines de Mo lors d'un rendu (voir le tableau VI.3).

Le fonctionnement asynchrone de la pile de rayons est rendu possible par l'exécution distincte des requêtes d'intersection et des lectures de blocs dans des threads différents. L'implémentation de cet asynchronisme dans un contexte multiprocesseur est décrit en suivant.

1.4 Contexte multiprocesseur

1.4.1 Organisation des différents flots d'exécution

1.4.1.1 Synchronisation Afin de bénéficier des performances des machines multiprocesseurs, le moteur de lancer de rayons est adapté en conséquence. Les différents flots d'exécution sont constitués d'un ou plusieurs threads de traitement des rayons, et d'un thread de chargement des blocs (voir la figure VI.4). Le chargement des données est limité en terme d'efficacité par le nombre de disques utilisés ; dans le contexte d'un seul disque dur par calculateur (voir la section B à la page 192), un seul thread de ce type est instancié. De manière similaire, pour exploiter le parallélisme d'exécution des rayons, un thread de traitement des rayons est instancié par processeur. Le multithreading permet alors essentiellement d'améliorer les performances pour des modèles occasionnant peu d'accès aux données.

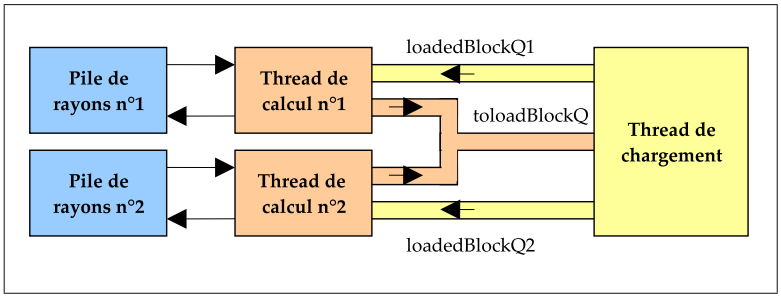


FIG. VI.4 – Organisation des files de données entre les threads de calcul et le thread de chargement.

En raison de la structure d'allocation mémoire des rayons, et pour maintenir une certaine équité entre les threads de calcul, chacun d'eux gère sa propre pile de rayons. Un verrou (le seul) est nécessaire pour la communication entre les threads de calcul et le thread de chargement. Lors de l'ajout d'un nouveau bloc à charger dans la file *toloadBlockQ*, chaque thread de calcul doit d'une part vérifier que le bloc ne soit pas déjà dans la file, et d'autre part verrouiller l'accès à la file. Ce verrou n'est pas pénalisant car il est utilisé rarement, comparé au traitement engendré par le grand nombre de rayons ; sa mise en place est réalisée par l'incrémentation protégée d'un pointeur de positionnement.

1.4.1.2 Equilibre de charge L'exécution parallèle des threads de calculs doit être contrôlée pour que chaque thread ait une charge de travail équitable. Il n'est pas optimal dès le départ de remplir les piles de rayons de chaque thread, car même si elles sont de même taille, les temps de calcul pour chaque thread seraient complètement différents. Ce déséquilibre provient du découpage initial qui ne tient compte que des rayons primaires ; ainsi à partir de ces seules informations, aucune estimation réaliste ne peut prévoir le nombre de rayons secondaires générés et/ou le nombre de voxels traversés. Un équilibrage de charge dynamique s'impose.

Une simple affectation dynamique des parties de l'image à rendre permet de réguler efficacement la charge de travail. Contrairement à une approche distribuée (voir la section 2 à la page 167), les parties de l'image doivent être proches les unes des autres afin de mieux exploiter le cache de blocs qui est partagé.

```
//Tant que tous les rayons ne sont pas traités
bool no_job = false;
while (!(rayStack.empty() && toloadBlockQ.empty() && loadedBlockQ.empty() &&
     !no_job) {
   //recuperation en priorité des blocs chargés
   while (b = loadedBlockQ.pop()) {} //...
   //s'il y a un rayon alors traitement
   if (r = rayStack.pop()) {} //...
   //s'il reste des régions à traiter et que l'équilibre de charge est bon
   if (!no_job && isGoodLoadBalancy()) {
     //affectation d'une nouvelle région (répartiteur centralisé)
     Tile *t = getNextTile();
     if (t == NO_JOB) no_job = true;
     else {
       //creation des rayons associes
       t->createPrimaryRays(rayStack);
       //initialisation de l'équilibre de charge pour cette nouvelle région
       t->initLoadBalancy();
     }
   }
}
```

FIG. VI.5 – Equilibre de charge de la pile de rayons, dans un contexte multiprocesseur, contrôlé par la fonction *isGoodLoadBalancy*.

Une propriété délicate de la pile de rayons est qu'elle doit être la plus grande possible afin de générer un maximum de blocs à charger pour optimiser la lecture séquentielle, et d'offrir un large choix lors de la sélection du prochain rayon à traiter. Si l'on réalise une affectation séquentielle des parties de l'image (entre 64 et 1024 rayons), cela oblige un thread de calcul à terminer la partie en cours avant de pouvoir commencer à traiter la prochaine. La fin du traitement d'une partie de l'image se fait alors avec un nombre réduit de rayons, et il n'est plus alors possible de masquer les latences de lecture des blocs. Pour remédier à ce problème, chaque thread de calcul doit traiter seulement une proportion de la partie d'image affectée (ex : 10%) avant de demander la prochaine partie ; cette valeur est contrôlée par la fonction *isGoodLoadBalancy* dans la figure VI.5. De cette manière l'affectation dynamique rend satisfaisant l'équilibre de charge global, tout en maintenant une taille minimale suffisante de la pile de rayons (voir le tableau VI.3).

1.4.2 Politiques de chargement/déchargement des blocs

La gestion des allocations et désallocations mémoire des blocs devient une tâche délicate lors de l'exécution parallèle de plusieurs threads de calcul. Dans le contexte monoprocesseur, la responsabilité de la gestion mémoire pouvait être donnée sans conflit d'accès soit au thread de chargement, ou soit à l'unique thread de calcul. Dans le contexte multiprocesseur, il est nécessaire de déporter cette responsabilité sur le

thread de chargement. Bien que l'allocation de la mémoire reste simple à réaliser, la désallocation, caractérisée par la recherche d'un bloc à décharger et sa libération effective, nécessitent de s'assurer que les threads de calcul n'utilisent pas le bloc. Les politiques de chargement/déchargement des blocs sont décrites en suivant.

```
//boucle sans fin
while (!mustStop()) {
  //changement de contexte si rien à faire
  if (toloadBlockQ.empty()) Sleep(0);
  BlockQ bQ;
  //recopie locale de MAX_COLLECT blocs à charger
  for (int i = 0; i < MAX_COLLECT && !toloadBlockQ.empty(); i++)
    bQ.add(toloadBlockQ.pop());
  //Si il y a des blocs a charger
  if (!bQ.empty()) {
    //Tri des blocs à charger selon l'ordre d'accès au fichier
    bQ.sort();
    //Signale un besoin de mémoire de "blocsList.totalSize()" octets
    garbage(bQ.totalSize());
    Block *b;
    //Chargement des blocs
    while (b = bQ.pop()) {
      b->load(); //Lecture physique
      //signale aux threads de calcul qu'un nouveau bloc est charge
      for (int i = 0; i < nRayThreads; i++) loadedBlockQ[i].push(bloc);
    }
  }
}
```

FIG. VI.6 – Thread de chargement des blocs.

1.4.2.1 Chargement Dans l'algorithme VI.6, après la lecture effective d'un bloc, le thread de chargement signale le chargement à tous les threads de calcul en ajoutant la référence du bloc dans chaque file *loadedBlockQ*. Cette diffusion générale évite une synchronisation complexe sur le nombre de rayons en attente d'un bloc et n'entraîne aucune surcharge de calculs. L'agrégation des files *loadedBlockQ* peut être vue comme un bus de données sur lequel chaque thread de calcul écouterait et prendrait les informations intéressantes : en l'occurrence les rayons en attente attachés à un bloc récemment chargé. On peut remarquer que, même si chaque thread de calculs récupère le même bloc nouvellement chargé, seuls les rayons en attente, dont ils ont la propriété, sont récupérés. Pour implémenter cette notion de propriété, chaque bloc contient autant de files *waitingRayQ* que de threads de calculs. Ceci implique qu'un rayon primaire initialement affecté dans un thread de calcul, ne pourra jamais changé de thread, et aura également toute sa descendance traitée dans ce même thread.

Le débit de lecture (ou d'écriture) d'un disque dur est directement fonction de la localité de ses accès. L'utilisation d'une pile de rayons permet d'améliorer cette localité, car elle génère une file de blocs à charger qui peut être réorganisée pour optimiser les accès au disque. Pour cela, un tri des blocs est réalisé à partir de la position de lecture actuelle dans le fichier. L'implémentation de ce tri permet globalement de multiplier par deux les débits de lecture.

1.4.2.2 Déchargement L'ajout d'une section critique à chaque utilisation (traversée) d'un bloc (voxel) par les threads de calculs, dans le but de vérifier la validité d'un bloc chargé en mémoire, est une solution certes opérationnelle mais trop pénalisante. La solution adoptée évite toute synchronisation explicite, et part du principe que la désallocation n'est pas une tâche critique. En effet, l'épuration de la mémoire doit se faire rapidement mais pas instantanément ; il suffit pour cela de prévoir une marge de mémoire supplémentaire (moins de 50 Mo).

```
int minBlock = MAX_INT;
//Recherche de la plus petite position de tounloadBlockQ signalee
//par les threads de calcul
for (int i = 0; i < nRayThreads; i++)
  minBlock = __min(minBlock, rayThread[i].unloadLimit);
//Liberation des blocs jusqu'à le plus petite limite commune
while (tounloadBlockQ.pos < minBlock) tounloadBlockQ.pop()->free();
```

FIG. VI.7 – Fonction de déchargement des blocs.

La fonction *garbage* dans la figure VI.6 s'occupe de décharger le nombre de blocs nécessaire à la libération d'une taille mémoire spécifiée en paramètre. L'implémentation de cette fonction est réalisée en deux parties : la première partie s'occupe de collecter les blocs à libérer dans une file *tounloadBlockQ*, et positionne l'état de ces blocs comme libre ; de cette façon, les threads de calcul peuvent continuer à utiliser les blocs en cours mais n'utiliseront plus les blocs à décharger lorsqu'ils auront pris connaissance de leurs états. La seconde partie (voir la figure VI.7) s'occupe de retirer de cette file les blocs à libérer, au fur et à mesure que les threads de calcul aient pris réellement connaissance de l'état libre des blocs. Pour s'assurer que les threads de calcul aient bien détecté cet état, ceux-ci mettent à jour régulièrement une variable privative *unloadLimit* qui est une copie de la position du pointeur de la file de déchargement. De cette façon, le thread de chargement récupère la valeur de ces variables et peut libérer, sans compromettre le système, tous les blocs jusqu'à la plus petite position commune signalée.

1.5 Résultats

Des expérimentations ont été réalisées pour valider la mise en place de la file de rayons sur un seul calculateur. Les rendus ont été exécutés sur une machine du cluster (voir la section B à la page 192), et paramétrés avec une profondeur de récursion de 3 ainsi qu'une résolution d'image de 800x600 pixels. Le tableau VI.3 montre les temps de rendu, avec utilisation des rayons primaires, d'ombre et secondaires, pour les très grands modèles. Les mémoires caches sont vides au début de chaque expérimentation.

Modèle	P_{pr}	Temps	$\#C/\#U$	Taille max. pile rayon	Taille blocs chargés	Cache plein
chevrolet512	0%	**32 s**	1	73,6 Mo	330 Mo	Non
	10%	33 s	1	17,9 Mo	330 Mo	
	50%	41 s	1	8,2 Mo	330 Mo	
chevrolet4096	0%	182 s	1,03	98,3 Mo	1,7 Go	Oui
	10%	**174 s**	1	20,9 Mo	1,6 Go	
	50%	194 s	1	4,2 Mo	1,6 Go	
engine256	0%	1 153 s	6,07	239,5 Mo	15,3 Go	Oui
	10%	**408 s**	1,56	34,9 Mo	3,9 Go	
	50%	589 s	1,77	11,2 Mo	4,4 Go	
engine512	0%	3 195 s	7,25	281,9 Mo	34,7 Go	Oui
	10%	**1 843 s**	3,31	34,6 Mo	15,8 Go	
	50%	2 263 s	3,56	12,1 Mo	17 Go	

TAB. VI.3 – Temps de rendu des très grands modèles. Le rapport $\#C/\#U$ représente le nombre de blocs chargés sur le le nombre de blocs utiles.

Le champ P_{pr} signifie qu'une certaine proportion des pixels doit être rendu avant de pouvoir ajouter d'autres rayons primaires dans la pile de rayons. Cette limite permet d'équilibrer la charge lors d'une exécution multithreadée, et également de diminuer la taille maximale occupée en mémoire par la pile de rayons (voir la section 1.4.1.2 à la page 161). Le champ *CachePlein* indique que le cache a été rempli intégralement au cours du rendu.

Les résultats montrent que la meilleure valeur de P_{pr} est de 10% pour la majorité des modèles. Pour $P_{pr} = 0\%$, la taille de la pile des rayons peut occuper plus de 25% de la mémoire centrale, ce qui diminue d'autant la taille disponible pour le cache de blocs et provoque ainsi plus de défauts de cache. Avec $P_{pr} = 50\%$, la taille de la pile de rayons est réduite, mais la capacité de masquer les latences par le traitement d'autres rayons, est fortement diminuée. On peut constater que $P_{pr} = 0\%$ est optimal pour le modèle *chevrolet512* car la taille de son cache n'a pas été influencée par celle de la pile de rayons.

Le tableau VI.4 présente une comparaison des résultats obtenus entre le lancer de rayons synchrone et l'utilisation de la pile de rayons pour le rendu des très grands modèles sur un seul calculateur.

Modèle	Lancer de rayons classique	Pile de rayons $P_{pr} = 10\%$	Accélération
chevrolet512	68 s	33 s	x2,1
chevrolet4096	296 s	174 s	x1,7
engine256	730 s	408 s	x1,8
engine512	3 168 s	1 843 s	x1,7

TAB. VI.4 – Performance de la pile de rayons par rapport à un lancer de rayons synchrone, pour les très grands modèles

Ces temps de rendu ont été obtenus avec une exécution multithreadée de la pile de rayons. Ce parallélisme ne profite qu'au modèle *chevrolet512* car le temps de rendu des autres modèles est prédominé par le chargement des blocs depuis le disque. Globalement, la pile de rayons permet d'accélérer les performances de rendu des très grands modèles par un facteur x1,8 par rapport à un lancer de rayons synchrone. Cette accélération est essentiellement due au flux constant des blocs à charger en mémoire qui permet une réorganisation efficace de l'ordre des lectures.

1.6 Discussion

Dans cette section, nous avons présenté une solution pour rendre, sur un seul calculateur, des modèles dont la taille est jusqu'à 8 fois plus grande que celle de la mémoire centrale. L'approche construit un arbre octal dont le premier niveau sert comme structure de gestion, et le second niveau comme structure d'accélération. Seul le premier niveau de cet arbre est entièrement chargé en mémoire. Le second niveau est découpé en blocs correspondant directement aux feuilles de la structure de gestion. Ces blocs constituent des éléments chargeables à la volée dans une mémoire de type cache gérée en LRU, selon les besoins des requêtes d'intersection ; leur taille d'environ 32 Ko a été choisie afin d'offrir un bon débit de lecture et une bonne granularité de cache.

Pour masquer les temps de lecture des blocs, nous avons proposé l'utilisation d'une pile de rayons en remplacement de l'algorithme classique récursif. Cette pile est initialement alimentée avec les rayons primaires, et reçoit par la suite tous les rayons secondaires (au sens large). Le principe général est de traiter immédiatement tous les rayons qui utilisent des blocs présents dans le cache, et de mettre en attente les autres. Ces derniers sont alors retirés de la pile, puis réintroduits lorsque le bloc manquant est effectivement chargé. En parallèle, un processus s'occupe en continu de charger dans le cache tous les nouveaux blocs sollicités. De ce fait, le temps de lecture des prochains blocs utiles peut être recouvert par le traitement des rayons qui exploitent actuellement le cache.

Le choix d'une pile permet par son ordre d'exécution de type LIFO (Last In First Out), de traiter en priorité tous les nouveaux rayons, ainsi que ceux qui étaient dans l'attente d'un bloc. Ceci a pour effets de favoriser l'avancée dans les calculs, de maximiser l'utilisation du cache et de maintenir une grande file de blocs à charger. Le réordonnancement de cette file, pour refléter l'organisation physique séquentielle des blocs sur le disque, permet d'accélérer par x2 le débit de lecture.

La performance globale obtenue par l'exploitation de la pile de rayons, permet d'accélérer le traitement des très grands modèles en moyenne par un facteur x1,8 par rapport à une solution par lancer de rayons synchrone. Nous pensons que cette performance peut être encore améliorée par la construction paresseuse des structures d'accélération incluses dans les blocs. Cela permettrait d'une part d'accélérer d'un ordre de grandeur le temps de création des blocs, et d'autre part de réduire leur taille, ce qui aurait directement pour conséquence de diminuer les temps de lecture. Bien que la construction à la volée des structures d'accélération puisse représenter

un temps additionnel, nous pensons que ce temps, grâce à la pile de rayons, serait généralement recouvert par du temps de lecture. Le temps de rendu peut également être fortement diminué par la distribution de la pile de rayons sur une grappe de calculateurs ; ce point est présenté dans la section suivante.

2 Distribution de la pile de rayons sur une grappe de calculateurs

2.1 Problématique

La grappe de calculateurs (ou cluster) offre, par le parallélisme, une plus grande puissance de calculs, et par l'agrégation, une plus grande capacité mémoire. Ces avantages, disponibles à un faible coût si l'on considère des machines standard B à la page 192, suscitent un grand intérêt pour traiter les modèles complexes. Cependant, la pleine exploitation d'une architecture distribuée requiert un fin contrôle de l'équilibre de charge, ainsi qu'une forte optimisation des interactions intra-machines, notamment en privilégiant les communications asynchrones pour recouvrir les latences réseau, et en exploitant au mieux la localité des calculs pour minimiser les échanges de données.

Deux principales approches, issues de l'état de l'art, sont analysées dans un premier temps : la distribution pilotée par les données (data driven), et la distribution pilotée par les calculs (demand driven). De l'analyse des travaux précédents, une distribution de la pile des rayons, pilotée par les calculs, est proposée. La fonction de chargement des blocs est alors étendue pour récupérer par le réseau les données situées dans les mémoires distantes, lorsque le modèle est réparti dans l'ensemble des mémoires. Ensuite, des détails d'optimisation des interfaces pour accélérer les communications, et afficher au plut tôt les rendus par un affichage progressif, sont explicités. Enfin, les résultats d'une série de rendus distribués, directement comparables aux résultats mono-machine, sont synthétisés.

2.2 Travaux précédents

La première approche, historiquement la plus ancienne, répartit initialement le modèle dans l'ensemble des mémoires des calculateurs [59]. La distribution du calcul est ensuite pilotée par les données possédées par chaque calculateur, en propageant les rayons de machine en machine [59, 62]. Cette forme de distribution offre l'avantage d'utiliser l'ensemble des mémoires, sans redondance d'informations, pour y charger intégralement le modèle. Cependant, les solutions obtenues sont viables pour un faible nombre de processeurs, et souffrent d'une mauvaise "scalabilité", notamment en raison du nombre important de transferts de rayons entre chaque machine. Pour remédier au problème d'équilibre de charge, il est possible d'estimer a priori la charge de travail, soit par un sous échantillonnage initial du modèle [59], ou soit par l'utilisation d'une pyramide de vision [74]. Au cours du rendu, lorsque le déséquilibre de

charge est trop important, il est possible de redistribuer les données dynamiquement entre machines [20].

La distribution pilotée par la répartition des données semble dans un premier temps attrayante mais souffre essentiellement d'un équilibre de charge difficile à contrôler, et d'un nombre de communications entre calculateurs trop important. De plus, les solutions sont forcément limitées à des modèles pouvant contenir dans l'ensemble des mémoires.

La seconde approche, plus courante, est pilotée par le découpage de l'image (ensemble de rayons primaires) en régions de pixels connexes, qui sont ensuite distribuées à chaque calculateur. Les informations de géométrie du modèle nécessaires à chaque machine sont alors acheminées via le réseau ou récupérées localement depuis le disque en fonction des besoins des calculs.

Wald *et al.* [81] montrent une solution interactive de visualisation de très grands modèles en exploitant un serveur de données et des calculateurs. La géométrie est centralisée sur le serveur de données et distribuée de manière asynchrone, voxel par voxel, à la demande des calculateurs qui gèrent un cache LRU des voxels récupérés. La solution permet de rendre un modèle, représentant une usine électrique, composé de 12,5 millions de triangles. Le fichier du modèle occupe alors une taille de 2,5 Go sur disque, après un prétraitement d'une durée de 2 heures et 30 minutes. Les résultats obtenus avoisinent les 5 images/s sur un cluster de 7 biprocesseurs.

Bien que la solution permette effectivement de charger paresseusement les données par le réseau, et de rendre intéractivement un grand modèle, l'analyse des résultats montre que l'animation sollicite un faible débit de données de 2 Mo/s avec quelques pics à 13 Mo/s qui saturent le réseau. Ceci indique que le *working set* est relativement petit par rapport à la mémoire des machines, et que la problématique traitée relève plus de la parallélisation que de la distribution. De plus, Wald *et al.* font état d'un mécanisme de réordonnancement des rayons pour masquer les latences d'accès, mais ne mentionnent à aucun moment la technique utilisée pour cela.

Demarle *et al.* [18] proposent d'accéder aux blocs de données sur une architecture distribuée par un système de mémoire partagée. Le modèle est alors réparti équitablement entre les calculateurs, et chargé intégralement dans l'ensemble des mémoires. La mise en place d'un gestionnaire de mémoire sur chaque calculateur permet de rendre transparent l'accès aux blocs de données, qui sont récupérés soit depuis la mémoire locale, ou soit depuis la mémoire d'une machine distante.

Les performances atteintes sont de l'ordre de 2 images/s, pour un modèle d'une taille de 7,5 Go, sur un cluster doté de 31 processeurs. La solution est alors viable, s'il l'on ne considère que des modèles pouvant résider dans l'ensemble des mémoires centrales. Cependant, ces travaux réalisent des chargements de blocs synchrones, sans se soucier de recouvrir les latences, certainement parce qu'un réseau haute performance est utilisé (environ 10 fois plus rapide que celui décrit dans la section B à la page 192).

Par rapport à ces travaux, la distribution de la pile de rayons apporte un mécanisme efficace pour recouvrir toutes les latences : disque et réseau. Pour ne pas

limiter la taille des modèles exploitables, deux solutions sont proposées pour accéder aux blocs lors d'un défaut de cache :

- lecture depuis les mémoires distantes si le modèle tient dans l'ensemble des mémoires ;
- lecture depuis le disque local sinon.

2.3 Distribution

2.3.1 Répartition des calculs

La répartition des calculs entre les différentes machines reprend la même stratégie de distribution maître/esclaves que celle présentée dans [15]. Le maître affecte dynamiquement des zones rectangulaires de l'image aux esclaves, tout en veillant à maintenir une pleine charge de travail. Chaque esclave traite les rectangles reçus, effectue le rendu puis renvoie la partie d'image calculée au maître. Sur ce dernier point, une technique originale d'affichage progressif, décrite dans la section 2.4.2 à la page 173, implique que l'image est renvoyée de manière asynchrone par groupes de pixels non obligatoirement connexes.

Dans le contexte de traitement de très grands modèles, la méthode d'affectation veille à distribuer à un esclave des rectangles proches de ceux précédemment affectés afin de maximiser l'utilisation des caches locaux. La topologie générale de cette distribution est modélisée dans la figure VI.8.

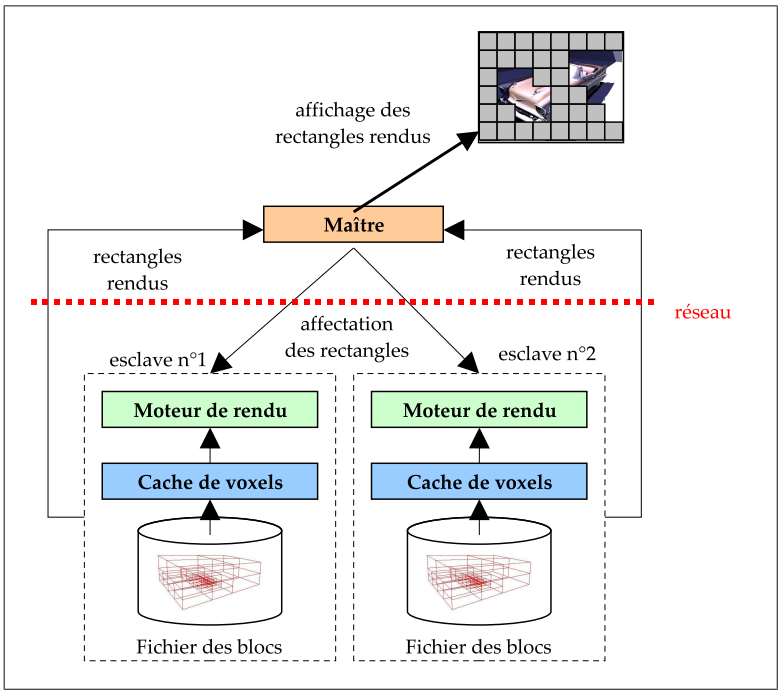


FIG. VI.8 – Topologie de distribution maître/esclaves

Pour améliorer l'équilibre de charge et diminuer le nombre de communications, un

mécanisme de raffinement dynamique des rectangles, déclenché par une estimation du temps de calcul restant [15], est une solution optimale. Néanmoins, l'utilisation de la pile de rayons n'est réellement efficace que lorsque le nombre de rayons empilé est important. Ainsi, pour maintenir un nombre suffisant de rayons, chaque esclave demande l'affectation du rectangle suivant dès qu'il a traité une proportion prédéfinie P_{ra} de la dernière affectation (voir la section 1.4.1.2 à la page 161). Malheureusement dans ce cas, l'estimation du temps est biaisée par le calcul partiel, et la subdivision des rectangles intervient trop tard. Ces erreurs sont accrues quand le traitement d'un rectangle engendre de nombreux chargement de blocs, car une partie des rayons est traitée très rapidement, tandis que l'autre partie est dans l'attente des données. Le temps mesuré par le maître pour ce dernier type de rectangle est le temps de traitement des rayons les plus rapides, ce qui implique alors une mauvaise estimation. Pour remédier à ce problème, il est nécessaire d'augmenter la proportion des rayons à traiter jusqu'à obtenir un équilibre de charge satisfaisant.

Bien que la répartition des calculs soit importante, celle-ci n'est pas suffisante pour traiter efficacement les très grands modèles. Il reste nécessaire d'améliorer la fonction de chargement des blocs, exécutée de nombreuses fois au cours du rendu, afin de profiter pleinement de la mémoire distribuée.

2.3.2 Chargement des blocs

Lorsque le modèle peut être contenu dans l'ensemble des mémoires distribuées, il est intéressant en cas de défaut de cache, de récupérer via le réseau, plus rapide d'accès que le disque, les blocs situés dans les mémoires distantes. Pour des modèles plus conséquents, le chargement des blocs depuis le disque reste une solution viable. Afin d'offrir une solution générale, la fonction de chargement de blocs est adaptée pour accéder aux données, en toute transparence, aussi bien depuis le disque que depuis le réseau.

2.3.2.1 Depuis le disque Pour charger les blocs depuis le disque dans une topologie distribuée, la solution mono-machine peut être directement reprise pour chaque esclave en dupliquant le modèle. Il est cependant nécessaire, dans une phase d'initialisation, de recopier le fichier modèle sur chaque machine. Bien que cette opération représente un temps additionnel, sa complexité d'exécution peut être rendue logarithmique en réalisant une recopie arborescente entre le serveur et les esclaves.

2.3.2.2 Depuis les mémoires distantes En raison de sa nature mécanique, les latences d'un disque sont beaucoup plus grandes que celles d'un réseau (rapport 30 mesuré avec la configuration matérielle présentée dans la section B à la page 192). Pour bénéficier d'un accès aux données plus performant, il est intéressant de remplacer tout accès au disque par un accès réseau.

Si l'on considère une solution centralisée [81] dans laquelle les esclaves récupèrent les blocs par le réseau depuis un serveur centralisé, la solution limite d'une part la

taille des modèles exploitables à la taille mémoire du serveur, et génère d'autre part un goulet d'étranglement vers ce serveur. Par contre, en répartissant équitablement le modèle dans l'ensemble des mémoires, chaque esclave devient responsable d'une partie des blocs et peut récupérer, en fonction des calculs, les blocs manquants depuis les mémoires distantes. Cette topologie permet de mieux répartir la charge réseau, et de n'être limitée que par la capacité mémoire de l'ensemble des machines.

Pour réaliser la répartition, il est nécessaire de dédier, une partie de la mémoire, au sous-ensemble du modèle à charge de l'esclave. Ceci implique alors une diminution de la taille du cache de chaque esclave, et entraîne, en raison d'un plus grand taux de défaut de cache, un accroissement du nombre de chargement des blocs. Cependant, bien que la répartition des blocs soit statique, leur chargement en mémoire depuis le disque est reporté à une utilisation effective, soit en raison d'une utilisation locale, ou soit en raison d'une sollicitation extérieure. La taille du cache reste alors acceptable tant qu'une partie restreinte du modèle est utile. Dans le cas contraire, caractérisé par une taille de mémoire cache insuffisante, la solution peut basculer dynamiquement vers la méthode de chargement des blocs depuis le disque.

2.3.2.3 Solution mixte Etant donné la nécessité de supporter les deux types d'accès aux données, l'organisation des threads de calculs et de chargement est adaptée, par rapport à la version mono-machine, comme illustrée dans la figure VI.9.

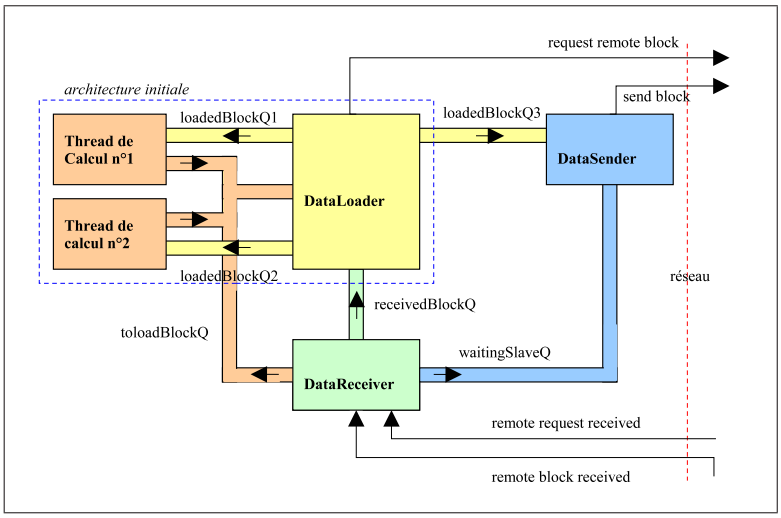


FIG. VI.9 – Organisation des threads et des files de données, dans un processus esclave, pour supporter le chargement des blocs depuis le disque et depuis le réseau. Un carré représente un thread, et une flèche représente une file de données.

Cette organisation étend l'architecture utilisée pour lire les blocs depuis le disque. Chaque type d'action occasionnant une latence d'accès est exécuté dans un thread dédié pour ne pas perturber le traitement de la pile de rayons. Les threads de calculs échangent avec le *DataLoader* de la même manière que dans la version mono-machine. Deux nouveaux threads, qui sont le *DataSender* et le *DataReceiver*, s'oc-

cupent respectivement de l'émission et de la réception des requêtes et des blocs de données.

En suivant un cycle complet de chargement d'un bloc, le *DataLoader* commence par récupérer de la file *toloadBlockQ* les blocs à charger. Cette file peut aussi bien être alimentée par les threads de calculs locaux, que par les machines distantes (via le *DataReceiver*). En fonction de la stratégie en cours et du type de bloc, le *DataLoader* peut soit lire ce bloc depuis le disque, ou soit émettre une requête vers la machine le possédant. Dès qu'un bloc est chargé, en raison d'une fin de lecture fichier, ou d'un retour de bloc en réponse à une requête (via le *DataReceiver*), le *DataLoader* signale le chargement en envoyant une information dans toutes les files *loadedBlockQ*. Les threads de calculs récupèrent alors ces informations, et réveillent les rayons qui étaient en attente (voir la section 1.4 à la page 161). Le *DataSender*, également connecté à une file *loadedBlockQ*, vérifie si un esclave, référencé dans la liste *waitingSlaveQ*, est en attente d'un bloc nouvellement chargé, et s'occupe dans l'affirmative de transmettre le bloc par réseau. On peut remarquer que la liste *waitingSlaveQ* est alimentée par le *DataReceiver* qui recueille toutes les requêtes extérieures de chargement. Le bloc transmis est enfin reçu par le *DataReceiver* de l'esclave en attente, et ajouté à la liste des blocs reçus par réseau *receivedBlockQ* ; le *DataLoader* traite alors cette information comme précédemment décrit au début du cycle.

La mise en place de ce système a été délicate, notamment pour maintenir une bonne équité d'exécution entre les différents threads, et également pour supporter l'interruptibilité des calculs, caractérisée par l'annulation globale des blocs à traiter, sans compromettre l'intégrité du système. Avant de présenter les résultats de cette solution, les optimisations apportées aux interfaces de communication et d'affichage, sont décrites en suivant.

2.4 Interfaces

La bonne "scalabilité" d'une application parallèle requiert de minimiser les calculs séquentiels [1], ainsi que les interactions entre calculateurs. Pour ces raisons, l'interface de communication pour échanger des messages d'ordre et de données, et ainsi que la récupération centralisée des parties d'image vers le serveur, ont été particulièrement optimisées.

2.4.1 Couche de communication

Pour obtenir un meilleur asynchronisme, il a été nécessaire d'écrire une bibliothèque de communication sur mesure. L'utilisation de la bibliothèque MPI [49] qui offre des fonctions évoluées, mais non thread-safe (pour la version antérieure à MPI-2), ne permet pas de communiquer dans des threads différents. Une couche de plus bas niveau a été entièrement écrite en s'appuyant sur les sockets TCP/IP. Outre la possibilité de supporter le multithreading, cette nouvelle interface, en remplacement de MPI, a permis de réduire les latences réseau d'environ 10%.

Cette couche de communication offre des fonctions qui permettent au maître d'envoyer des ordres à un esclave particulier, ou à tous les esclaves en même temps (broadcast). Dans une phase d'initialisation, le programme maître se met en attente de connexion des programmes esclaves, en écoutant sur un port prédéfini. Dès qu'un esclave se connecte, celui-ci crée alors un thread d'écoute dédié sur un nouveau port de communication qu'il fournit dynamiquement au maître. Ce sont ces ports de communication qui permettent aux esclaves de traiter toutes les commandes du maître, et des autres esclaves. Dans le même principe que l'invocation d'objets distants [71], les communications sont encapsulées de telle manière à créer et synchroniser sur chaque esclave les mêmes objets que ceux manipulés par le maître. Cette abstraction permet alors, en toute transparence, d'exécuter le programme maître aussi bien dans une version mono-machine, que dans une version distribuée.

Au-delà des communications maître/esclaves, il est nécessaire, pour supporter la lecture des blocs dans les mémoires distantes, d'interconnecter les esclaves entre eux. Le maître, après avoir collecté l'ensemble des ports de communications dans la phase d'initialisation, redistribue la table complète à chaque esclave, qui s'interconnecte alors en respectant l'ordre d'établissement présenté dans la figure VI.10.

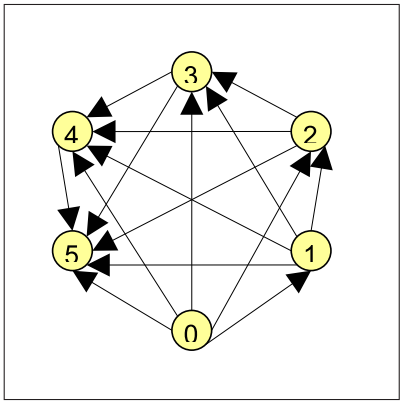


FIG. VI.10 – Interconnexion de six esclaves selon leur ordre de création.

2.4.2 Affichage progressif

L'affichage de l'image est réalisé progressivement par le serveur, au fur et à mesure de l'exécution des rectangles affectés aux esclaves. Ceci implique une utilisation de la bande passante réseau assez conséquente, qui pourrait être réduite en adoptant des stratégies de compression/décompression à la volée. Cependant, afin de fournir au plus tôt une vision globale du rendu, permettant par exemple de valider/invalider instantanément le point de vue en cours, certains pixels de l'image sont affectés d'une priorité de calcul plus forte. Cette désorganisation des pixels rend alors difficile l'exploitation d'algorithmes de compression basés image, et a impliqué un mécanisme particulier de rapatriement des pixels par lot.

2.4.2.1 Notion de pixels prioritaires Une propriété intéressante de la pile de rayons est que l'ordre d'exécution des rayons n'est pas prédéfini ; chacun pou-

vant être potentiellement élu pour traitement. Dans une version sans politique de réordonnancement, le prochain rayon de la pile utilisée est celui situé à son sommet. Choisir volontairement un rayon plutôt qu'un autre permet d'optimiser une opération de rendu. Cette opération, dans le contexte du traitement de très grands modèles, peut consister à sélectionner les rayons afin de minimiser le nombre total de blocs chargés en mémoire. Cependant, ce réordonnancement est implicitement réalisé par l'ordre de chargement des blocs même, car à ceux-ci sont rattachés des sous-ensembles de rayons en attente qui seront réveillés dès que le bloc sera chargé. Un autre contexte d'utilisation du réordonnancement, plus intéressant, est d'afficher en priorité certains pixels de l'image pour obtenir un rendu plus interactif. Les pixels affichés prioritairement peuvent soit recouvrir un objet du modèle qu'un opérateur aurait décidé de voir rapidement, ou soit recouvrir globalement l'image pour un affichage, certes grossier, mais global. Une étude portée sur ce dernier point a permis de mettre en place une pile de rayons avec priorité.

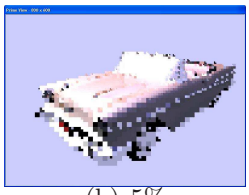
Chaque rayon est affecté d'une priorité qui influe sur son ordre d'exécution. A priorité égale, les rayons sont retirés en respectant l'ordre classique LIFO de dépilement. L'implémentation est réalisée en utilisant une table de priorités dont chaque entrée pointe sur une pile de rayons de même priorité. L'ajout d'un rayon se fait alors directement en trouvant l'entrée de la table correspondant à la priorité du rayon. Le retrait d'un rayon est également direct, car un pointeur, indiquant l'entrée de la table de plus haute priorité, est utilisé. Ce pointeur est mis à jour lors de l'ajout de chaque rayon, par une simple opération de comparaison avec la priorité du rayon, et lors du retrait du dernier rayon d'une pile, par recherche de proche en proche de la prochaine entrée contenant des rayons. Cette opération de recherche peut être considérée comme négligeable car elle est réalisée peu de fois par rapport au nombre de rayons traités.

La priorité d'un rayon primaire est affectée en fonction des coordonnées du pixel générateur sur le plan de l'image ; elle est ensuite transmise par recopie à toute la descendance de rayons. En terme d'affichage, les pixels non calculés sont colorés avec la couleur des pixels récemment calculés. Des arbres de coloration (quad-tree), dans lesquels les pixels représentent les feuilles, sont alors utilisés. La racine d'un arbre est représentée par un macro-pixel (voir la figure VI.12(a)), et chacun de ses noeuds représente un quart du macro-pixel parent. Lorsqu'un pixel est rendu, l'information de couleur remonte récursivement dans l'arborescence jusqu'au premier macro-pixel non coloré. Ce dernier macro-pixel, se marque alors comme coloré, et remplit de sa couleur la partie de l'image qu'il recouvre.

La matrice de priorité présentée dans la figure VI.12(a) est utilisée pour obtenir une répartition homogène. Le principe d'affichage progressif permet alors d'obtenir rapidement une image globale du rendu comme le montrent les images de la figure VI.11.

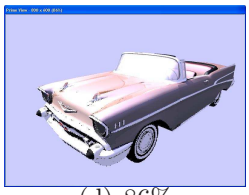
1	0	1	0	1	0	1
0	2	0	0	0	2	0
1	0	1	0	1	0	1
0	0	0	3	0	0	0
1	0	1	0	1	0	1
0	2	0	0	0	2	0
1	0	1	0	1	0	1

(a) Matrice de priorités par région de 7^2 pixels.

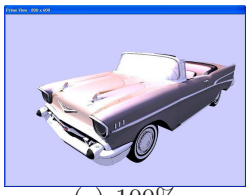


(b) 5%

(c) 32%



(d) 86%



(e) 100%

FIG. VI.11 – Progression de l'affichage avec utilisation d'un arbre de coloration.

2.4.2.2 Rapatriement des pixels par lot La conséquence d'un traitement non déterminé des rayons est que la coloration des pixels peut avoir lieu à plusieurs endroits de l'image, sans pour autant remplir intégralement un des rectangles affectés. Pour permettre la visualisation progressive de l'image, et éviter un envoi massif de données à la fin des calculs, chaque esclave mémorise dans un tampon un nombre maximal de pixels (coordonnées et couleur), et transmet ce tampon, de manière asynchrone, dès qu'il atteint sa taille maximale (voir la figure VI.12). La taille de ce tampon permet soit d'optimiser le débit réseau (environ 8000 pixels), ou soit d'augmenter le confort de vision (1 pixel) ; un compromis de 1000 pixels permet de satisfaire ces deux besoins.

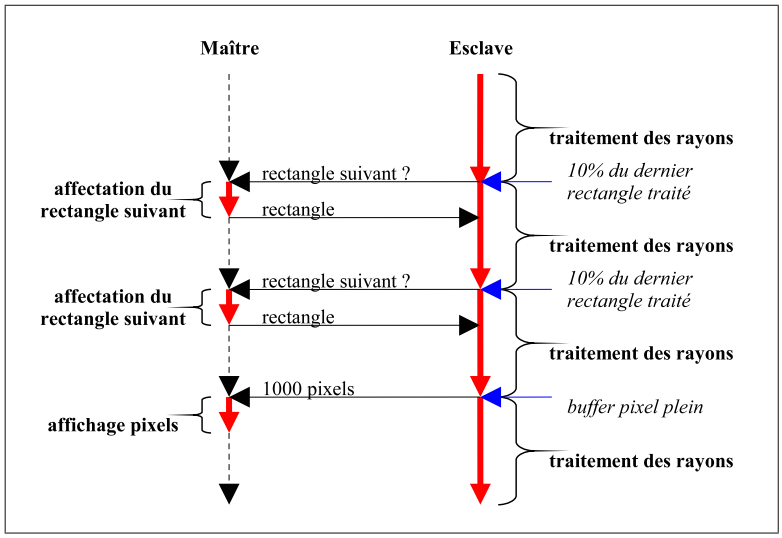


FIG. VI.12 – Affectation des rectangles et rapatriement asynchrone des pixels par lot.

2.5 Résultats

2.5.1 Présentation de l'application

Un outil purement démonstratif a été développé afin de pouvoir naviguer dans les très grands modèles, afficher un rendu de la position de la caméra en cours, activer la distribution des calculs, et fournir des statistiques sur les calculs en cours.

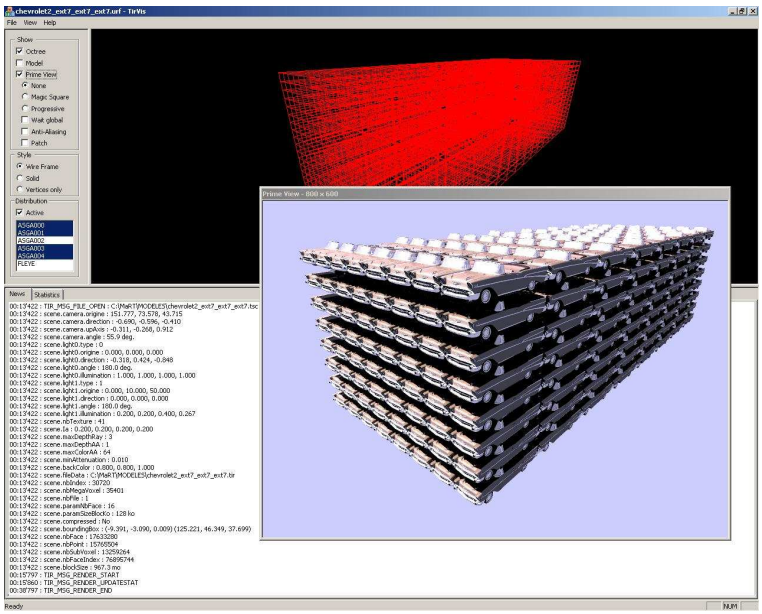


FIG. VI.13 – Ecran principal du prototype de rendu distribué.

L'interface de cet outil (voir la figure VI.13) est divisée en trois zones. La première est une zone d'actions qui permet de spécifier des options de rendu tel que l'affichage de l'arbre octal et/ou du modèle. Elle permet également de lancer le rendu et d'activer la distribution des calculs en choisissant les esclaves à exploiter.

Les deux autres zones permettent la visualisation du modèle au moyen de deux vues fonctionnellement différentes. La première vue permet d'afficher instantanément un modèle quelle que soit sa taille. Celle-ci permet alors de naviguer dans ce modèle et de positionner la caméra. L'interactivité de l'affichage est garantie par l'utilisation d'OpenGL pour afficher le premier niveau de l'arbre octal, qui a une faible complexité géométrique.

La deuxième vue affiche le rendu calculé par lancer de rayons. Sa visualisation est déportée dans une fenêtre spéciale qui, permet entre autre de redimensionner l'image. Cette fenêtre de visualisation affiche progressivement l'image rendue, et peut, à la fin des calculs afficher les pixels anti-aliasés. Le rendu peut être réinitialisé à tout moment lors du redimensionnement de l'image, ou quand l'opérateur navigue dans le modèle ; cette interactivité, au niveau de l'IHM, n'aurait pu avoir lieu si le système de rendu distribué ne supportait pas, au plus bas niveau, l'interruptibilité.

2.5.2 Mesures de performances

Afin de vérifier et valider les méthodes utilisées, une batterie de tests a été réalisée sur les mêmes modèles que ceux présentés dans la section 1 à la page 154. Ces tests ont été mesurés sur le cluster en utilisant 5 machines (voir la section B à la page 192). Les expérimentations comparent les performances de la distribution avec un chargement des blocs depuis le disque, à une distribution avec un chargement des blocs depuis les mémoires distantes. Les résultats obtenus dans la version distribuée sont directement comparables à ceux obtenus dans la version mono-machine de la pile de rayons (voir le tableau VI.4). Tous les calculs d'efficacité de parallélisation et de surcharges de chargement s'y réfèrent.

Les paramètres suivants sont utilisés par défaut quand aucune autre indication supplémentaire n'est spécifiée dans la légende des tableaux :
- la profondeur du lancer de rayons est limitée à 3 ;
- l'anti-aliasing n'est pas activé ;
- la résolution des images rendues est fixée à 800x600 pixels ;
- l'affichage progressif n'est pas activé, les temps mesurés comprennent alors le rendu de tous les pixels de l'image.

Les colonnes utilisées dans les tableaux ci-dessous ont la signification suivante :
- *Action.* Désigne une étape de test du rendu. Une $n^{ième}$ exécution signifie que le rendu est exécuté pour la $n^{ième}$ fois sans changer aucun paramètre et en conservant l'état des mémoires. Une action de *zoom* signifie que la position de la caméra est avancée d'un offset prédéfini. Une action *Déc. droite* signifie que la position de la caméra est décalée sur la droite d'un offset prédéfini.
- *N calculateurs.* Donne le temps d'exécution global du rendu pour N calculateurs. On peut remarquer que $N = 1$ signifie que la version mono-machine est utilisée, ce qui n'est pas identique à une version distribuée avec 1 esclave.
- *Imb.* : Donne la mesure du déséquilibre de charge selon la formule suivante :

$$Desequilibre = 1 - \frac{TempsMoyenParEsclace}{TempsMaxEsclave}$$

  $TempsMoyenParEsclave$ désigne la moyenne des temps de rendu des esclaves, et $TempsMaxEsclave$ désigne le temps du rendu le plus long parmi les esclaves ;
- *Mémoire.* Donne la taille totale des blocs chargés dans l'ensemble des mémoires distribuées ;
- *Diff.* : Donne la différence entre la taille mémoire en cours et la taille mémoire de l'action précédente ;
- *Débit* : Donne le débit global de chargement des blocs dans les mémoires pour l'action en cours.

Pour plus de concision, deux sigles sont utilisés pour différencier les types de distribution :
- *LOC* : Chargement des blocs depuis la mémoire de masse locale ;

– *REM* : Chargement des blocs depuis les mémoires distantes ;

2.5.2.1 Modèle *Chevrolet 512* (1 Go) Le test n°1 (voir le tableau VI.5) montre que la distribution *LOC* a une efficacité de parallélisation de 45%, qui est essentiellement due à une surcharge de lecture de 41%. Pour rappel, l'efficacité de parallélisation se calcule de la manière suivante :

$$Efficacite = \frac{TempsMonoMachine}{NbEsclave \times TempsMaxEsclave}$$

Le modèle *Chevrolet 512* est un "petit" modèle car, dans la version mono-machine, il peut être intégralement chargé en mémoire. La distribution, n'apportant pas de mémoire supplémentaire utile, est confrontée aux chargements redondants, mais indispensables, de blocs en mémoire. L'efficacité de parallélisation est donc sous-optimale lors de la $1^{ère}$*exécution* ; les exécutions suivantes montrent que la distribution profite des mémoires caches, et permet ainsi de réaliser ce rendu entre 3 s et 5 s.

Action	1 calculateur	5 calculateurs	Imb.	Mémoire	Diff.	Débit
1ère exéc.	33 s	14,0 s	23%	468 mo	468 mo	33 mo/s
2nde exéc.	-	3,7 s	37%	520 mo	52 mo	14 mo/s
3ème exéc.	-	3,0 s	45%	546 mo	26 mo	9 mo/s
Zoom	-	5,0 s	42%	612 mo	66 mo	13 mo/s
Déc. droite	-	4,0 s	40%	658 mo	47 mo	12 mo/s

TAB. VI.5 – Test n°1. *LOC*.

La distribution *REM* dans le test n°2 (voir le tableau VI.6) provoque une surcharge de lecture de 85% par rapport au rendu mono-machine, et de 30% par rapport à la distribution *LOC*. Cette surcharge est due à la répartition du modèle dans l'ensemble des mémoires qui implique en moyenne de charger deux fois un bloc, une unique fois depuis le disque, et une ou plusieurs fois ensuite par le réseau, pour les esclaves les sollicitant. Cependant, cette surcharge n'est constatée que lors de la $1^{ère}$*exécution*, et disparaît ensuite pour les exécutions suivantes, permettant globalement à la distribution *REM* d'être plus réactive (moins de 3 s). Cette performance peut être expliquée par des latences plus faibles et plus régulières du réseau qui permettent de mieux contrôler l'équilibre de charge ($Imb. < 10\%$).

Action	5 calculateurs	Imb.	Mémoire	Diff.	Débit
1ère exéc.	14,2 s	3%	611 mo	611 mo	43 mo/s
2nde exéc.	1,7 s	9%	655 mo	44 mo	26 mo/s
3ème exéc.	1,3 s	3%	670 mo	15 mo	12 mo/s
Zoom	3,0 s	10%	763 mo	93 mo	31 mo/s
Déc. droite	2,3 s	3%	829 mo	66 mo	29 mo/s

TAB. VI.6 – Test n°2. *REM*.

Le test n°3 (voir le tableau VI.7) est un test de performances qui montre qu'avec une résolution moindre (400x300 pixels), et pour une quantité négligeable de géométrie à charger, que ce modèle peut être rendu en 2 images/s (voire la description du cluster à la section B à la page 192).

Action	5 calculateurs	Imb.	Mémoire	Diff.	Débit
1ère exéc.	11,0 s	2%	383 mo	383 mo	35 mo/s
2nde exéc.	994 ms	26%	539 mo	156 mo	157 mo/s
3ème exéc.	499 ms	10%	545 mo	6 mo	12 mo/s
4ème exéc.	755 ms	24%	562 mo	17 mo	22,5 mo/s
5ème exéc.	495 ms	19%	571 mo	9 mo	18,2 mo/s
6ème exéc.	499 ms	3%	571 mo	0 mo	0 mo/s

TAB. VI.7 – Test n°3. *REM* avec une résolution de 400x300 pixels.

2.5.2.2 Modèle *Chevrolet 4096* (8 Go) Le test n°4 (voir le tableau VI.8) permet de calculer une efficacité de parallélisation de 69%, et une surcharge de lectures de seulement 20%. Les différents temps d'exécution, montrent que ce modèle peut être rendu entre 7 s et 12 s sous condition de légers mouvements de caméra.

Action	1 calculateur	5 calculateurs	Imb.	Mémoire	Diff.	Débit
1ère exéc.	174 s	50,5 s	18%	1 934 mo	1 934 mo	38 mo/s
2nde exéc.	-	12,6 s	39%	2 187 mo	254 mo	20 mo/s
3ème exéc.	-	7,8 s	64%	2 255 mo	68 mo	9 mo/s
Zoom	-	12,3 s	25%	2 497 mo	242 mo	20 mo/s
Déc. droite	-	9,6 s	30%	2 654 mo	157 mo	16 mo/s

TAB. VI.8 – Test n°4. *LOC*.

Bien que, en raison de sa taille, ce modèle ne puisse pas en théorie être réparti dans l'ensemble des mémoires distribuées, son *working set* restreint, ainsi que le chargement à la volée des blocs utiles, permettent de le rendre dans une distribution *REM*. Le test n°5 (voir le tableau VI.9) occasionne 48% de plus de lectures qu'une distribution *LOC*. Cette surcharge se fait ressentir lors de la $1^{ère}$*exécution* qui dure 20 s de plus. La $3^{ème}$*exécution* donne le meilleur temps de rendu pour ce modèle avec 4,3 s. Par contre, le temps de rendu peut atteindre 16 s, contre un maximum de 12 s, dans la distribution *LOC*. Cette différence est induite par la nécessité de charger de nouveaux blocs lors d'une action de *zoom* ou de *décalage droite*, ce qui implique le double chargement, disque puis réseau, expliqué précédemment.

De plus en comparant la différence mémoire entre la première action et la dernière action, la distribution *LOC* charge 720 Mo de données, tandis la distribution *REM* charge 2 Go de données. En plus du phénomène de double chargement de blocs, cette différence est accrue par la réduction de la taille de la mémoire cache due à une configuration répartie.

Action	5 calculateurs	Imb.	Mémoire	Diff.	Débit
1ère exéc.	70,8 s	0%	2 858 mo	2 858 mo	40 mo/s
2nde exéc.	7,1 s	7%	3 252 mo	394 mo	56 mo/s
3ème exéc.	4,3 s	23%	3 475 mo	223 mo	52 mo/s
Zoom	14,1 s	2%	4 237 mo	762 mo	54 mo/s
Déc. droite	16,2 s	5%	4 866 mo	629 mo	39 mo/s

TAB. VI.9 – Test n°5. *REM*.

Le test n°6 (voir le tableau VI.10) montre que ce modèle peut également être rendu en 2 images/s, dès que le cache est stabilisé, pour une résolution de 400x300 pixels. On peut remarquer, malgré des exécutions successives du même point de vue, une légère remontée dans les temps de calculs entre la $3^{ème} exécution$ et $4^{ème} exécution$, de 1,6 s à 1,8 s. Cette différence s'explique par de légères variations des affectations de rectangles entre deux rendus.

Action	5 calculateurs	Imb.	Mémoire	Diff.	Débit
1ère exéc.	61,6 s	0%	2 278 mo	2 278 mo	37 mo/s
2nde exéc.	3,6 s	47%	2 453 mo	175 mo	49 mo/s
3ème exéc.	1,6 s	34%	2 541 mo	88 mo	55 mo/s
4ème exéc.	1,8 s	62%	2 590 mo	49 mo	27 mo/s
5ème exéc.	798 ms	39%	2 602 mo	12 mo	0 mo/s
6ème exéc.	552 ms	5%	2 602 mo	0 mo	0 mo/s

TAB. VI.10 – Test n°6. *REM* avec une résolution de 400x300 pixels.

2.5.2.3 Modèle *Engine 256* (2,4 Go) L'efficacité de parallélisation de la distribution *LOC* dans le test n°7 (voir le tableau VI.11) est de 77% pour une surcharge de lecture par rapport au rendu mono-machine de 20%. Les temps de rendu entre les différents changements de position de la caméra varient entre 75 s et 88 s. En comparaison de ces temps, la $2^{nde} exécution$ qui a le temps de rendu le plus grand avec 128 s, peut être mise en marge et qualifiée de cas particulier. Il s'avère, au regard du déséquilibre de charge de 23 %, que le temps de rendu d'un seul esclave ait pénalisé l'ensemble du rendu.

Action	1 calculateur	5 calculateurs	Imb.	Mémoire	Diff.	Débit
1ère exéc.	408 s	106,3 s	8%	4 688 mo	4 688 mo	44 mo/s
2nde exéc.	-	128,5 s	23%	9 322 mo	4 634 mo	36 mo/s
3ème exéc.	-	-	-	-	-	-
Zoom	-	75,3 s	12%	15 858 mo	15 858 mo	-
Déc. droite	-	88,8 s	11%	19 295 mo	3 437 mo	39 mo/s

TAB. VI.11 – Test n°7. *LOC*.

Le test n°8 (voir le tableau VI.12) montre une efficacité de parallélisation de 120% pour une surcharge de lecture par rapport au rendu mono-machine de 85%. Cette performance de parallélisation, qui est la meilleure de tous les modèles, provient probablement de la quantité de données nécessaire à l'exécution de ce modèle. Nécessitant beaucoup de chargement de blocs, le phénomène initial de double chargement des blocs est amorti, et permet à ce modèle de profiter pleinement de la faible latence et meilleure stabilité du réseau. La distribution *REM* permet une fois de plus d'obtenir le meilleur temps de rendu avec 61,6 s.

Action	5 calculateurs	Imb.	Mémoire	Diff.	Débit
1ère exéc.	68,8 s	2%	7 226 mo	7 226 mo	105 mo/s
2nde exéc.	73,3 s	3%	13 411 mo	6 185 mo	84 mo/s
3ème exéc.	72,5 s	2%	19 830 mo	6 419 mo	89 mo/s
Zoom	68,7 s	2%	25 770 mo	5 940 mo	86 mo/s
Déc. droite	61,6 s	-	-	-	-

TAB. VI.12 – Test n°8. *REM*.

2.5.2.4 Modèle *Engine 512* (4,8 Go) En raison d'une bonne utilisation de l'ensemble des mémoires caches, la distribution du test n°9 (voir le tableau VI.13) ne génère pas de surcharge de lectures par rapport au rendu mono-machine, mais permet au contraire d'en diminuer le nombre de 33%. Cette réduction permet alors d'avoir une efficacité de parallélisation surlinéaire de 109%.

Action	1 calculateur	5 calculateurs	Imb.	Mémoire	Diff.	Débit
1ère exéc.	1 843 s	338,3 s	29%	10 591 mo	10 591 mo	31 mo/s
2nde exéc.	-	278,3 s	12%	21 204 mo	10 613 mo	38 mo/s
3ème exéc.	-	328,5 s	25%	31 882 mo	10 678 mo	33 mo/s
Zoom	-	304,8 s	16%	42 966 mo	11 084 mo	36 mo/s
Déc. droite	-	287,8 s	14%	53 681 mo	10 715 mo	37 mo/s

TAB. VI.13 – Test n°9. *LOC*.

2.6 Discussion

Dans cette section, nous avons présenté une distribution du rendu par lancer de rayons pour les très grands modèles selon une topologie maître/esclaves. Le principe de la pile de rayons est réutilisé en dissociant deux façons de charger les blocs. La première reprend la lecture des blocs depuis le disque, utilisée dans la version mono-machine, en dupliquant le modèle sur chaque esclave. La seconde façon profite de la plus faible latence du réseau, pour récupérer les blocs chargés dans les mémoires des autres esclaves. Cette configuration se limite alors aux modèles dont la taille, ou plus exactement le *working set*, peut être intégralement réparti dans l'ensemble des

mémoires distribuées. La solution globale est alors implémentée de façon à supporter les deux configurations.

Pour un meilleur confort d'utilisation, nous avons présenté une technique d'affichage progressif du rendu qui permet de naviguer interactivement dans les très grands modèles. Cette technique exploite une notion de priorité au niveau du rayon, qui permet ensuite de sélectionner, lors du dépilement, les rayons qui contribuent le mieux au rendu global de l'image.

D'une manière générale, l'efficacité de la distribution varie entre 50% et 120% en privilégiant les modèles qui génèrent beaucoup d'accès aux données. Les meilleurs temps d'exécution sont obtenus lors de l'utilisation des mémoires distantes qui offrent un débit de lecture plus rapide et plus constant, malgré une surcharge de lectures allant jusqu'à de 80% en raison d'une taille du cache réduite.

Cette distribution souffre cependant d'un problème d'équilibre de charge, qui est particulièrement prononcé pour les configurations *LOC*. Ce déséquilibre doit être mieux contrôlé en apportant des mécanismes supplémentaires pour guider l'affectation des rectangles aux esclaves. Nous pensons qu'il est possible d'y remédier en affectant les rectangles en fonction des données chargées en mémoire. Cette solution pourrait être réalisée en comptant par région, et pour chaque esclave, le nombre de blocs chargés par projection sur le plan de l'image ; le maître choisirait alors d'affecter à chaque esclave la région contenant le plus de blocs chargés.

Avec une résolution d'image de 400x300 pixels, les modèles *chevrolet512* et *chevrolet4096* peuvent être rendus en 2 images/s. Ces derniers résultats montrent la capacité globale du système à exploiter le parallélisme d'exécution au niveau des processeurs (flots d'exécution) et des calculateurs (distribution).

Conclusion

Nous avons proposé dans ce chapitre une solution pour traiter les très grands modèles par l'utilisation conjointe d'une pile de rayons pour masquer les latences d'accès aux données, ainsi que de la distribution pour bénéficier des capacités mémoire et de calculs d'une grappe de calculateurs. Pour travailler partiellement sur un modèle, nous avons organisé celui-ci en blocs issus de la construction d'un arbre octal. Chaque bloc est structuré de manière à être directement chargeable en mémoire, et à contenir les informations suffisantes à l'exécution d'une requête d'intersection.

A l'aide d'un seul calculateur, nous avons montré, qu'en remplacement d'un lancer de rayons synchrone qui charge les blocs en mémoire selon les besoins immédiats de chaque rayon, que l'utilisation d'une pile de rayons permet de :

- différer le traitement des rayons nécessitant des blocs non présents en mémoire ;
- réordonner la séquence des blocs à charger pour maximiser le débit de lecture ;
- traiter en parallèle les rayons exploitant les blocs déjà chargés, dans le but de masquer les temps de chargement.

Cet asynchronisme permet alors d'accélérer par un facteur x1,8 le rendu de très grands modèles par rapport à un lancer de rayons synchrone.

Dans une solution distribuée de la pile de rayons, nous avons montré que la meilleure façon d'accéder aux données dépendait de la taille du modèle par rapport à la taille des mémoires distribuées. Ainsi, quand ce rapport est > 1, la meilleure solution consiste à reprendre la configuration mono-machine pour chaque calculateur en dupliquant le modèle. Pour diriger les calculs, un serveur affecte dynamiquement les parties d'image à calculer, et récupère ensuite les rendus partiels pour affichage. Quand le rapport de taille ≤ 1, il devient possible de répartir le modèle équitablement dans l'ensemble des mémoires des machines. Les blocs nécessaires au rendu, et non possédés par un calculateur, sont alors récupérés depuis les mémoires des autres calculateurs, via le réseau plus rapide d'accès.

La solution distribuée offre une efficacité de parallélisation qui peut être surlinéaire pour les modèles les plus complexes. Pour des modèles plus modestes, ou dont le *working set* est restreint, la solution permet de rendre jusqu'à 2 images/s avec 5 calculateurs (voir la section B à la page 192).

Chapitre VII

Conclusions et perspectives

1 Synthèse des principales contributions

Chapitres II et III

Dans le chapitre II, nous avons proposé un intersecteur géométrique en réponse aux besoins de performances présents dans de nombreux domaines de la simulation de la propagation d'ondes. Cet intersecteur s'appuie sur des structures de recherche performantes : un kdTree et un arbre BVH. Les méthodes de construction utilisent, pour les deux structures de recherche, des techniques de construction paresseuse et de tri par intervalle, gages de rapidité et de qualité de construction. Le traitement des rayons simples, et des rayons SIMD pour bénéficier des architectures modernes, permet alors d'atteindre les performances actuelles de l'état de l'art.

Pour accroître ces performances, le chapitre III montre deux implémentations de ces groupes selon les structures de recherche supportées, après avoir présenté des techniques rapides de construction de groupes de rayons. L'exécution des groupes de rayons avec le kd-Tree s'inspire de plusieurs techniques existantes, et propose une adaptation afin de déterminer le meilleur ensemble de points d'entrée dans la structure de recherche. Par rapport aux méthodes concurrentes, cette méthode est la plus efficace si l'on ne considère pas l'utilisation d'objets occultants, difficiles à mettre en place, et limités à certains types de modèles. L'implémentation des groupes de rayons avec l'arbre BVH reprend les travaux de Wald *et al.* [77] en y apportant quelques modifications, notamment pour accélérer le rejet rapide d'une boîte par rapport à un faisceau.

Les performances des groupes de rayons offrent un facteur d'accélération moyen par rapport aux rayons simples entre x3,6 pour le kd-Tree, et x10,5 pour l'arbre BVH.

Chapitre IV

Dans le chapitre IV, nous avons montré l'exploitation des groupes de rayons à origine commune pour deux cas concrets d'application : les sources ponctuelles et l'occlusion ambiante. L'origine commune autorise une construction plus rapide des groupes de rayons, et apporte naturellement une meilleure cohérence. Avec une source ponctuelle, nous avons montré qu'il était possible avec l'arbre BVH d'obtenir des meilleures performances en réorganisant les rayons d'ombre les plus incohérents. En mémorisant, dans une hiérarchie de grilles centrée sur la source ponctuelle, les points d'entrée issus de la propagation d'un faisceau dans le kd-Tree, nous avons également montré que les rayons SIMD pouvaient égaler les performances des groupes de rayons en apportant une plus grande souplesse d'utilisation.

Dans le contexte de l'occlusion ambiante, nous avons proposé une requête de visibilité pour détecter les zones vides de surfaces autour des points d'évaluation. L'utilisation conjointe de cette requête, et de la structure d'un arbre octal, permet alors d'accélérer par un facteur x2 les calculs d'occlusion ambiante en tout point. En remarquant que l'occlusion ambiante est un phénomène qui varie lentement, nous avons adapté l'arbre octal pour limiter les points d'évaluation aux sommets des feuilles. Lors d'un rendu, le calcul paresseux de ces valeurs, et leur interpolation, permet d'obtenir un affichage interactif.

Chapitre V

Dans le chapitre V, nous nous sommes intéressés à la problématique de la cohérence des rayons, élément indispensable à la performance des groupes de rayons. Dans une première approche, et en partant du constat que les groupes de rayons sont plus efficaces avec l'arbre BVH, et que les rayons simples ou SIMD sont plus efficaces avec le kd-Tree, nous avons proposé d'exploiter ces deux structures de recherche de manière couplée. Une stratégie globale, s'appuyant sur la connaissance empirique de la cohérence des rayons, permet de sélectionner dynamiquement la structure de recherche la plus adaptée. Nous avons montré que le couplage de ces deux structures de recherche, dans un contexte de rendu générant tous types de rayons, apporte un gain de performance jusqu'à un facteur x1,5.

Dans une seconde approche, nous avons présenté une méthode simple pour déterminer a priori la cohérence d'un groupe de rayons. Cet indicateur permet d'une part d'accroître la performance en profitant de la cohérence partielle dans un groupe de rayons, et d'autre part de fournir un critère d'optimisation quand il s'agit de regrouper automatiquement un ensemble de rayons. Des résultats préliminaires de regroupement automatique de rayons, ont permis de valider la possibilité d'accélérer de manière transparente des codes mono-rayon, ou de reléguer à un processus automatisé la responsabilité de regrouper des rayons non aisément identifiables. Bien que les performances n'égalent pas les groupes de rayons constitués et optimisés "manuellement", les résultats obtenus sont encourageant d'autant plus que le temps pris par la réorganisation des rayons reste améliorable.

Chapitre VI

Le chapitre VI a montré une solution de réorganisation des rayons pour traiter les très grands modèles qui ne peuvent tenir dans la mémoire centrale d'une seule machine. La solution s'appuie sur un découpage de la géométrie en blocs issus de la construction d'un arbre octal. Ces blocs peuvent alors être chargés paresseusement dans la mémoire selon les besoins des rayons. Pour masquer les grandes latences d'accès au support de masse, l'utilisation d'une pile de rayons permet de traiter en parallèle la propagation des rayons, et le chargement des blocs utiles aux calculs. L'apport de la pile de rayons, par rapport à un lancer de rayons synchrone, permet alors d'accélérer par x1,8 les performances sur une seule machine.

La distribution de cette pile de rayons sur une grappe de calculateurs permet un rendu interactif d'un modèle constitué de plus de 100 millions de triangles. Nous avons proposé deux topologies de distribution en fonction de la taille des modèles. La première, et la plus performante, s'adresse aux modèles pouvant résider dans l'ensemble des mémoires distribuées. Chaque calculateur est alors responsable d'une partie du modèle qu'il garde dans sa mémoire centrale ; l'autre partie du modèle est alors récupérée, via le réseau plus rapide d'accès, depuis les mémoires distantes. La deuxième topologie répond aux modèles de plus grandes tailles en dupliquant le fichier des blocs sur chaque calculateur ; ceux-ci gèrent alors localement leur accès aux données sans perturber les autres calculateurs.

L'ensemble de la solution a été validée dans un prototype qui permet de naviguer interactivement dans des modèles de toutes tailles, en exploitant une notion de priorité dans la pile de rayons pour raffiner progressivement les images rendues.

2 Perspectives

Temps de construction des structures de recherche

Les structures de recherche sont à la base de la performance des requêtes d'intersection. Cependant, leur temps de construction est un frein à l'interactivité quand l'on considère des modifications de la géométrie. Plusieurs niveaux de modifications peuvent être différenciés :

- Si la géométrie évolue essentiellement sous la forme d'objets rigides, l'utilisation d'instances permet de limiter la reconstruction des arbres.
- Si la géométrie se transforme, alors des techniques de mises à jour partielles des arbres peuvent être utilisées [46, 80].
- Si le nombre de surfaces varie, il devient indispensable de reconstruire les arbres ; des solutions performantes doivent alors être trouvées.

En profitant des architectures multi-coeurs, il est possible de paralléliser la construction de l'arbre. Bien que cette tâche ne soit pas évidente dans les premiers niveaux de l'arbre, certaines approches [69] ont récemment démontré que la construction parallélisée pouvait atteindre une bonne "scalabité" en concédant en contrepartie une

certaine qualité de construction. Nous pensons alors que l'approche paresseuse peut améliorer ces techniques performantes. En gardant une parallélisation explicite du premier niveau de l'arbre, qui est de toute évidence nécessaire de construire intégralement, le niveau le plus fin peut être délégué à la construction paresseuse qui sera alors implicitement parallélisé par les requêtes d'intersection. L'approche apporterait alors des propriétés de plus faible consommation mémoire, et de plus grande rapidité de construction.

La même stratégie peut être envisagée pour réduire le temps de prétraitement des modèles complexes. En utilisant un arbre octal à deux niveaux, nous pensons qu'il est possible de ne construire que le premier niveau pour former des blocs de quelques milliers de triangles, et de déléguer la construction des sous-arbres à chaque calculateur et ce de manière paresseuse. Ceci permettrait de réduire le temps de prétraitement ainsi que la taille des fichiers de blocs ; le débit de chargement des blocs dans le cache, que ce soit depuis le disque ou depuis les mémoires distantes, serait alors amélioré.

Nous avons montré que le couplage des structures de recherche permet d'améliorer l'efficacité des groupes de rayons. Cependant, pour des scènes dynamiques, ce couplage augmente la problématique du temps de construction. Il est alors intéressant d'étudier dans quelle mesure, une des structures de recherche puisse être utilisée pour construire plus rapidement l'autre. Nous supposons qu'une approche montante de construction de l'arbre BVH à partir des feuilles du kd-Tree est une première piste d'exploration.

Efficacité des groupes de rayons

L'efficacité des groupes de rayons est due en grande partie à une réduction drastique du nombre d'étapes de traversées, et à un relatif maintien du nombre de calculs d'intersection. Ce maintien relève d'un équilibre entre le rejet rapide des triangles par rapport à un faisceau, et une augmentation générale du nombre de tests d'intersection rayon/triangle. Contrairement à ce que mentionne Reshetov [65], nous pensons qu'il n'est pas intéressant de contraindre davantage les critères de subdivision afin de construire des arbres moins profonds et dont les feuilles contiennent plus de triangles. D'une part, le bénéficie acquis de la réduction du nombre d'étapes de traversée est perdu, et d'autre part, en réduisant la profondeur de l'arbre, on oppose des complexités algorithmiques qui ne sont pas avantageuses. En effet, en raison de la complexité logarithmique de la traversée, une augmentation par un facteur deux du nombre de triangles dans les feuilles ne permet de réduire la propagation des rayons que d'une seule étape de traversée. Pour améliorer l'efficacité des groupes de rayons, il reste donc nécessaire d'optimiser les méthodes d'intersection entre un groupe de rayons et une surface, bien que potentiellement chaque rayon puisse être associé à une seule surface limitant alors inévitablement le gain.

Nous avons présenté un indicateur qui permet d'exploiter la cohérence partielle dans un groupe de rayons en utilisant alternativement la propagation du faisceau pour la partie cohérente de la traversée, et en utilisant les rayons SIMD pour les

sous-parties incohérentes. Bien que cette adaptation soit efficace, son mode de fonctionnement binaire ne semble pas optimal. Il est alors intéressant d'étudier dans quelle mesure une réduction progressive de la taille des groupes permettrait de mieux exploiter la cohérence. Bien que cette stratégie soit équivalente à celle proposée par Wald *et al.* [78], elle offrirait l'avantage d'éviter la coûteuse évaluation de proportion de rayons actifs en utilisant l'heuristique de surface du faisceau.

Temps additionnel pris par la réorganisation des rayons

De nombreuses techniques ont tenté de réorganiser les rayons par opposition aux groupes résultant de l'organisation naturelle des rayons primaires. Nous avons proposé de réorganiser les rayons d'ombre, de regrouper des grands ensembles de rayons automatiquement, et d'utiliser une pile de rayons dans le contexte des très grands modèles. Ces techniques partagent un but commun : accroître la cohérence. Nous avons alors montré que chacune d'elle permettait de réduire significativement le nombre d'opérations. Cependant, cette réorganisation représente un temps additionnel au traitement direct des rayons qui empêche d'atteindre la performance optimale. Nous pensons que les stratégies exploitées sont bonnes, mais qu'il est nécessaire de revoir leur implémentation pour les optimiser fortement, à l'image de toutes les optimisations apportées à la recherche d'intersection.

Interface pour une réorganisation automatique des rayons

Les groupes de rayons apportent un gain de performances indéniable par rapport aux rayons simples et SIMD. Leur pleine exploitation dans une application n'est cependant possible que si toute la chaîne de traitement des données a été pensée dans ce sens, et au prix d'efforts d'optimisation parfois incompatibles avec la productivité souhaitée. La réorganisation automatique des rayons répond partiellement à cette problématique en simplifiant la tâche de regroupement des rayons. Cependant pour que cette réorganisation soit efficace, il est nécessaire de disposer d'un grand ensemble de rayons afin d'accroître la probabilité de trouver des sous-ensembles cohérents. Cette contrainte devient alors incompatible avec l'utilisation classique du lancer de rayons récursif, ou plus généralement avec l'utilisation de la requête d'intersection dans des fonctions isolées. Il devient alors nécessaire de définir une nouvelle interface de programmation qui allie la souplesse d'utilisation du rayon simple à la performance des groupes de rayons.

Nous avons recensé trois types d'interfaces qui pourraient satisfaire ces besoins :

– *En deux passes.* Dans la section 2 à la page 132 qui traite du regroupement automatique des rayons, nous avons proposé une interface en deux passes. La première permet de définir les paramètres d'un grand nombre rayons, et la seconde permet de récupérer les résultats de recherche d'intersection. La réorganisation automatique des rayons intervient alors, de manière transparente, entre l'exécution de ces deux passes.

– *Evénementielle.* Lors du traitement des modèles complexes (voir la section VI à la page 153), nous avons proposé la mise en place d'une pile de rayons dont l'interface de programmation est événementielle : dès qu'un rayon est traité, une fonction événementielle est alors appelée, permettant ainsi au programmeur de réaliser des calculs physiques, et d'ajouter d'autres rayons dans la pile. Ce type d'interface permet alors de réorganiser en toute liberté les rayons de la pile afin de créer des groupes de rayons cohérents.
– *Par langage de programmation dédié.* Parker *et al.* [54] proposent un langage de programmation dédié au rendu par lancer de rayons. Ce langage dispose, parmi d'autres fonctions de haut niveau, d'une requête d'intersection. L'interprétation de ce langage génère un arbre de syntaxe abstraite qui permet, de manière générique, de produire du code compatible avec l'intersecteur souhaité ; il est alors possible d'intégrer des fonctions de réorganisation des rayons.

3 Conclusion générale

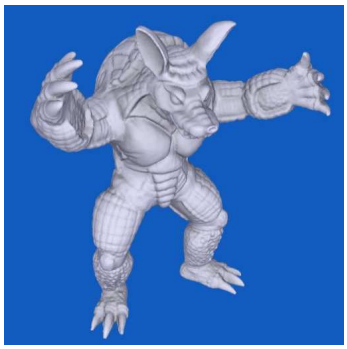
A partir des algorithmes de recherche d'intersections les plus efficaces pour des rayons simples et SIMD, nous avons montré dans cette thèse, comment, et dans quelle mesure, les groupes de rayons apportent un gain de performance additionnel. Nous avons constaté que ce gain est variable selon les structures de recherche utilisées, et dépend fortement de la cohérence présente dans les groupes de rayons. En réponse à cette problématique de cohérence, nous avons proposé des techniques de couplage des structures de recherche, ainsi que des techniques de réorganisation des rayons.

Les applications exploitant des requêtes d'intersection utilisent, par habitude et par simplicité, une interface de type "rayon simple" incompatible avec la notion de groupes de rayons. Puisqu'il n'est pas souhaitable de perdre cette simplicité, gage de productivité, l'interface de la requête d'intersection doit évoluer vers une forme qui puisse permettre la réorganisation des rayons, sans en faire apparaître la complexité. Définir une interface souple, et démontrer son efficacité, demeure alors un sujet de recherche.

Annexe A

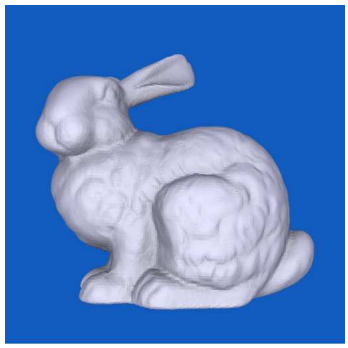
Modèles

La figure A.1 recense l'ensemble des modèles utilisés pour toutes les expérimentations de cette thèse. Ceux-ci ont été choisis afin de couvrir un large éventail de cas d'utilisations en distinguant des scènes intérieures et extérieures, des modèles scannés, des scènes architecturales et des modèles complexes. Nous avons limité certaines expérimentations, pour des raisons pratiques et de simplicité de présentation, aux modèles *Conference*, *Fairy Forest* et *Soda Hall* ; nous avons alors jugé que ces modèles, qui sont par ailleurs très bien connus de la communauté scientifique, suffisent à généraliser les résultats.

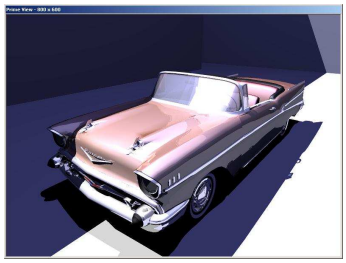


(a) Armadillo, 345 944 triangles.

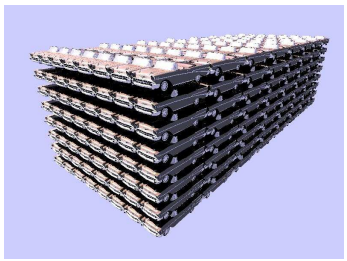
(b) Buddha, 1 087 716 triangles.



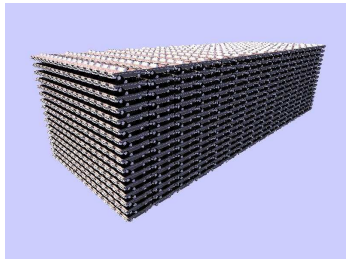
(c) Bunny, 69 451 triangles.



(d) Chevrolet, 29 662 triangles.



(e) Chevrolet512, 15 186 944 triangles. Taille du fichier de blocs : 1 Go.



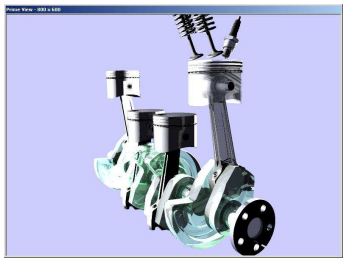
(f) Chevrolet4096, 121 495 552 triangles. Taille du fichier de blocs : 8 Go.

FIG. A.1 – Modèles utilisés lors des expérimentations.

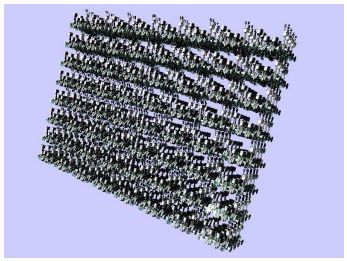
(g) Clipper, 584 155 triangles.

(h) Sully College, 804 196 triangles.

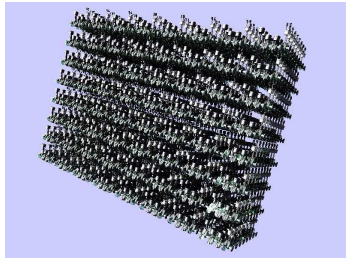
(i) Villa Digy, 80 914 triangles.



(j) Engine, 114 797 triangles.



(k) Engine256, 29 388 032 triangles. Taille du fichier de blocs : 2,4 Go.

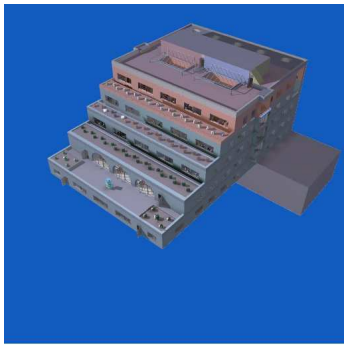


(l) Engine512, 58 776 064 triangles. Taille du fichier de blocs : 4,8 Go.

(m) Conference, 282 664 triangles.

(n) ERW6, 804 triangles.

(o) Fairy Forest, 174 117 triangles.



(p) Soda Hall, 2 169 132 triangles.

Fig. A.1 - suite.

Annexe B

Configuration matérielle

La configuration matérielle suivante a été utilisée pour l'ensemble des expérimentations présentées dans la thèse. L'ordinateur portable a été exclusivement employé dans les quatre premiers chapitres avec un seul coeur. Dans le dernier chapitre, les intitulés d'expérimentations différencient l'utilisation d'un ou plusieurs calculateurs du cluster.

Portable

- Marque : Dell Inspiron 9400
- Processeur : Intel Core 2 Duo
- Fréquence : 2 Ghz
- Mémoire centrale : 2 Go
- Carte graphique :
  - Processeur : NVidia GeForce Go 7900 GS
  - Mémoire : 256 mo
- Système d'exploitation : Windows XP SP2

Cluster

- Nombre de calculateurs : 6
- Réseau :
  - Type : Gigabit Ethernet
  - Latence moyenne : 130 μs
  - Bande passante : 55 Mo/s
  - Switch : 1000 Base-T

Calculateur

- Carte mère :

  - Marque : Tiger MPX (S2466N-4M)
  - Chipset : 760MPX
  - Mémoire centrale : 1 Go
- Processeur :
  - Nombre : 2
  - Marque : AMD Athlon MP 2000 +
  - Fréquence : 1667 MHz
  - Cache L1 : 64 ko
  - Cache L2 : 256 ko
- Carte graphique :
  - Processeur : NVIDIA GeForce 4 Ti 4600
  - Mémoire : 128 mo
- Disque dur :
  - Marque : IBM Deskstar 60 GXP
  - Capacité : 60 go
  - Vitesse de rotation : 7200 tours/min.
  - Cache : 2 mo
  - Latence moyenne : 4,17 ms
  - Accès moyen : 8,5 ms
  - Débit de lecture : 40 Mo/s
- Système d'exploitation : Windows 2000 Professionnel SP4

Annexe C

Publications associées à la thèse

Coupled Use of BSP and BVH Trees in Order to Exploit Ray Bundle Performance. *Gilles Cadet, Bernard Lécussan.* Article accepté à RT'07, Symposium on Interactive Ray Tracing 2007, Allemagne, 10-12 Septembre.

Fast Approximate Ambient Occlusion. *Gilles Cadet, Bernard Lécussan.* Poster accepté à Siggraph'07. San Diego, Californie, USA, 7-9 Août 2007.

Un lancer de rayons distribué pour traiter de très grands modèles sans précompilation. *Gilles Cadet, Bernard Lécussan.* Proceedings of Technique et science informatiques. 2006, vol. 25, no2, pp. 197-22.

Rendering Complex Scenes on Clusters with Limited Precomputations. *Gilles Cadet, Sebastian Zambal, Bernard Lécussan.* Proceedings of HPCSE-04, Workshop, Toulouse, France, 27 Août, 2004.

Une architecture distribuée pour simuler de très grands modèles de données. *Gilles Cadet, Bernard Lécussan.* Proceedings of Revue Scientifique et Technique de la Défense, RSTD 2004, Vol. 63, pp. 49 to 59.

Bibliographie

[1] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *SJCC*, 1967.

[2] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[3] D. Kirk J. Arvo. Fast ray tracing by ray classification. volume 21, pages 55–64, 1987.

[4] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. volume 5, pages 9–22, 2000.

[5] Didier Badouel. An efficient ray-polygon intersection. In Andrew S. Glassner, editor, *Graphics Gems*, pages 390–393, San Diego, 1990. Academic Press. includes code.

[6] Carsten Benthin. *Realtime Ray Tracing on current CPU Architectures*. PhD thesis, Saarland University, 2006.

[7] Sébastien Bermes. *Les arbres octaux paresseux : une méthode dynamique de subdivision spatiale pour le lancer de rayons*. PhD thesis, Supaero, 1998. Thèse.

[8] Jakko Bikker. Real-time ray tracing through the eyes of a game developer. In *Interactive Ray Tracing, 2007. RT '07. IEEE Symposium on*, pages 1–1, 2007.

[9] Solomon Boulos, Dave Edwards, J Dylan Lacewell, Joe Kniss, Jan Kautz, Peter Shirley, and Ingo Wald. Interactive distribution ray tracing. Technical Report UUSCI-2006-022, 2006.

[10] Solomon Boulos, Dave Edwards, J Dylan Lacewell, Joe Kniss, Jan Kautz, Ingo Wald, and Peter Shirley. Packet-based whitted and distribution ray tracing. In *Proceedings of Graphics Interface 2007*, pages 177 – 184, 2007.

[11] Solomon Boulos, Ingo Wald, and Peter Shirley. Geometric and arithmetic culling methods for entire ray packets. Technical Report UUCS-06-010, 2006.

[12] Michael Bunnell. Dynamic ambient occlusion and indirect lighting. In *GPU Gems 2*, chapter 14. 2005.

[13] L.A. Butler and A. Stephens. Bullet ray vision. In *Interactive Ray Tracing, 2007. RT '07. IEEE Symposium on*, 2007.

[14] Gilles Cadet and Bernard Lécussan. Une architecture distribuée pour simuler de très grands modèles de données : application à la modélisation de la propagation d'ondes par lancer de rayons. *RSTD*, 63 :49–59, 2004.

[15] Gilles Cadet, Sebastian Zambal, and Bernard Lécussan. Rendering complex scenes on clusters with limited precomputation. In *Proceedings of the International Symposium on High Performance Computational Science and Engineering 2004*, 2004.

[16] Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, December 1974.

[17] Frédéric Cazals, George Drettakis, and Claude Puech. Filtering, clustering and hierarchy construction : a new solution for ray-tracing complex scenes. *Comput. Graph. Forum*, 14(3) :371–382, 1995.

[18] David E. DeMarle, Steven G. Parker, Mark Hartner, Christiaan P. Gribble, and Charles D. Hansen. Distributed interactive ray tracing for large volume visualization. In Anton H. J. Koning, Raghu Machiraju, and Claudio T. Silva, editors, *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003 (PVG'03)*, pages 87–94, Seattle, WA, USA, October 2003. IEEE Computer Society (Los Alamitos, CA).

[19] Advanced Micro Devices. *3DNow ! Technology Manual*, 2000.

[20] M. Dippe and J. Swensen. An adaptive subdivision algorithm and parallel architecture for realistic image synthesis. In H. Christiansen, editor, *SIGGRAPH '84 Conference Proceedings (Minneapolis, MN, July 23-27, 1984)*, pages 149–158. ACM, July 1984.

[21] Kirill Dmitriev, Vlastimil Havran, and Hans-Peter Seidel. Faster ray tracing with simd shaft culling. Research Report MPI-I-2004-4-006, Max-Planck-Institut für Informatik, Saarbrücken, Germany, December 2004.

[22] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by A priori tree structures. *Conf. Proc. of SIGGRAPH '80*, 14(3) :124–133, July 1980.

[23] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. Arts : Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, pages 16–26, April 1986.

[24] Michael Gigante. Accelerated ray tracing using non-uniform grids. In *Proc. of Ausgraph '90*, pages 157–63, Melbourne, Australia, September 1990.

[25] A. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press, 1989.

[26] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5) :14–20, 1987.

[27] Gene S. Greger. The irradiance volume. Master's thesis, Cornell University, Ithaca, NY, August 1996.

[28] Eric A. Haines. The light buffer : A ray tracer shadow testing accelerator. Master's thesis, Cornell University, 1986.

[29] V. Havran and L. Bittner. On improving kd trees for ray shooting. In V. Skala, editor, *Journal of WSCG*, volume 10, 2002.

[30] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University, Praha, Czech Republic, April 2001. Available from http ://www.cgg.cvut.cz/ havran/phdthesis.html.

[31] Vlastimil Havran and Jirí Bittner. LCTS : Ray shooting using longest common traversal sequences. *Comput. Graph. Forum*, 19(3), 2000.

[32] Vlastimil Havran, Robert Herzog, and Hans-Peter Seidel. On the fast construction of spatial hierarchies for ray tracing. In *Proceedings of RT06 conference*, pages 71 – 80, 2006.

[33] HPC-SA. *RayBooster : Ray Tracing Acceleration Library.* www.raybooster.com.

[34] Jim Hurley, Alexander Kapustin, Alexander Reshetov, and Alexei Soupikov. Fast ray tracing for modern general purpose cpu. In *International Conference Graphicon 2002*, 2002.

[35] id Software. Wolfenstein 3d, 1992. http://fr.wikipedia.org/wiki/Wolfenstein_3D.

[36] Mental Images. Mental ray. http://www.mentalimages.com/2_1_0_mentalray/index.html.

[37] Mathematics Application Group Inc. Nuclear radiation exposure evaluation, 1966. http://design.osu.edu/carlson/history/tree/magi.html.

[38] Intel. Intel multi-core technology. http://www.intel.com/multi-core/index.htm.

[39] Intel. *IA-32 Intel Architecture Software Developer's Manual*, volume Volume 2 : Instruction set reference. 2003.

[40] Thiago Ize, Ingo Wald, and Steven G Parker. Asynchronous BVH Construction for Ray Tracing Dynamic Scenes on Parallel Multi-Core Architectures. In *Proceedings of the 2007 Eurographics Symposium on Parallel Graphics and Visualization.*

[41] Alexander Keller. Instant radiosity. In *Computer Graphics (ACM SIGGRAPH '97 Proceedings)*, volume 31, pages 49–56, 1997.

[42] P. Kensler, A. ; Shirley. Optimizing ray-triangle intersection via automated search. In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 33 – 38, 2006.

[43] Krzysztof S. Klimaszewski and Thomas W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17(1) :42–51, January 1997. bounding volume hierarchy with grids at each level & more.

[44] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In Anselmo Lastra, Marc Olano, David P. Luebke, and Hanspeter Pfister, editors, *SI3D*, pages 41–48. ACM, 2005.

[45] Hayden Landis. Production-ready global illumination. In *In Proc. SIGGRAPH, San Antonio, USA, Course 16 Notes, 2002.*, 2002.

[46] Christian Lauterbach, Sung-Eui Yoon, David Tuft, and Dinesh Manocha. RT-DEFORM : Interactive ray tracing of dynamic scenes using BVHS. Technical Report TR06-010, Department of Computer Science, University of North Carolina - Chapel Hill, April 26 2006. Wed, 26 Apr 2006 19 :51 :52 UTC.

[47] J. David MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3) :153–66, June 1990.

[48] Mattias Malmer, Fredrik Malmer, Ulf Assarsson, and Nicolas Holzschuch. Fast precomputed ambient occlusion for proximity shadows, December 2005.

[49] ANL Mathematics and Computer Science. *The Message Passing Interface (MPI) standard*. http://www-unix.mcs.anl.gov/mpi/.

[50] Erik Månsson, Jacob Munkberg, and Tomas Akenine-Möller. Deep coherent ray tracing. In *Symposium on Interactive Ray Tracing 2007*, 2007.

[51] Moller. A fast triangle-triangle intersection test. *JGTOOLS : Journal of Graphics Tools*, 2, 1997.

[52] Motorola. *AltiVec Technology Programming Interface Manual*, 1999.

[53] D.S. ; Lin C. ; Mark W.R. Navratil, P.A. ; Fussell. Dynamic ray scheduling to improve ray coherence and bandwidth utilization. In *Interactive Ray Tracing, 2007. IEEE Symposium on*, pages 95 – 104, 2007.

[54] S. Bigler J. Robison A. Parker, S.G. Boulos. Ray tracing shading language. In *Interactive Ray Tracing, 2007. RT '07. IEEE Symposium on*, pages 149–160, 2007.

[55] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In *Computer Graphics Proceedings, Annual Conference Series, 1997 (SIGGRAPH 97)*, pages 101–108, August 1997.

[56] Bui Tuong Phong. *Illumination for computer-generated images.* PhD thesis, Dept. of Electrical Engineering, University of Utah, 1973.

[57] J. ; Seidel H.-P. ; Slusallek P. Popov, S. ; Gunther. Experiences with streaming construction of sah kd-trees. In *Interactive Ray Tracing 2006, IEEE Symposium on*, 2006.

[58] L'équipe POV. Persistence of vision raytracer.

[59] Thierry Priol and Kadi Bouatouch. Lancer de rayon sur des architectures paralleles : une etude de performance. Technical Report RR-0973, Inria, Institut National de Recherche en Informatique et en Automatique.

[60] Walt Disney Productions. Tron, 1982. http://www-unix.mcs.anl.gov/mpi/.

[61] Realtime raytracing project of the University of Saarbruecken. The openrt real-time ray-tracing project. http://www.openrt.de/.

[62] Erik Reinhard and Frederik W. Jansen. Rendering large scenes using parallel ray tracing. *Parallel Computing*, 23(7) :873–885, 1997.

[63] Erik Reinhard, Arjan J. F. Kok, and Frederik W. Jansen. Cost prediction in ray tracing. In *Rendering Techniques '96*, pages 41–50. Springer-Verlag, June 1996.

[64] Reshetov. Omnidirectional ray tracing traversal algorithm for kd-trees. In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 57 – 60, 2006.

[65] Reshetov. Faster ray packets - triangle intersection through vertex culling. In *ACM SIGGRAPH 2007 posters*, 2007.

[66] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2005)*, 24(3) :1176–1185, 2005.

[67] S. M. Rubin and T. Whitted. A 3-dimensional representation for fast rendering of complex scenes. *Proceedings of SIGGRAPH'80*, pages 110–116, 1980.

[68] Jörg Schmittler, Daniel Pohl, Tim Dahmen, Christian Vogelgsang, and Philipp Slusallek. Realtime ray tracing for current and future games. In Peter Dadam and Manfred Reichert, editors, *GI Jahrestagung (1)*, volume 50 of *LNI*, pages 149–153. GI, 2004.

[69] Maxim Shevtsov, Alexei Soupikov, and Alexander Kapustin. Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. In *Computer Graphics Forum*, volume 26, pages 395–404, 2007.

[70] Ken Shoemake. Plücker coordinate tutorial. In *Ray Tracing News*, volume 11. 1998.

[71] Inc. Sun Microsystems. Remote method invocation. http://java.sun.com/j2se/1.5.0/docs/guide/rmi/index.html.

[72] Kelvin Sung and Peter Shirley. Ray tracing with the BSP tree. In David Kirk, editor, *Graphics Gems III*, pages 271–274. Academic Press, San Diego, 1992. includes code.

[73] László Szécsi. *An Effective Implementation of the kd-Tree*, pages 315 – 326. Charles River Media, Inc, 2003.

[74] Maurice van der Zwaan, Erik Reinhard, and Frederik W. Jansen. Pyramid clipping for efficient ray traversal. In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 1–10, New York, 1995. Springer-Verlag.

[75] Carsten Wächter and Alexander Keller. Instant ray tracing : The bounding interval hierarchy. In Tomas Akenine-Möller and Wolfgang Heidrich, editors, *Eurographics Workshop/Symposium on Rendering*, pages 139–149, Nicosia, Cyprus, 2006. Eurographics Association.

[76] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004.

[77] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, 26(1), 2007.

[78] Ingo Wald, Christiaan P Gribble, Solomon Boulos, and Andrew Kensler. SIMD Ray Stream Tracing - SIMD Ray Traversal with Generalized Ray Packets and On-the-fly Re-Ordering. Technical Report UUSCI-2007-012, 2007.

[79] Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 61–69, 2006.

[80] Ingo Wald, William R Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G Parker, and Peter Shirley. State of the Art in Ray Tracing Animated Scenes. In *Eurographics 2007 State of the Art Reports*.

[81] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive distributed ray tracing of highly complex models. In *RENDERING TECHNIQUES'01*, pages 277–288.

[82] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. In A. Chalmers and T.-M. Rhyne, editors, *EG 2001 Proceedings*, volume 20(3) of *Computer Graphics Forum*, pages 153–164. Blackwell Publishing, 2001.

[83] Chris Wassenius. Accelerated ambient occlusion using spatial subdivision structures. 2005.

[84] Kyu-Young Whang, Ju-Won Song, Ji-Woong Chang, Ji-Yun Kim, Wan-Sup Cho, Chong-Mok Park, and Il-Yeol Song. Octree-R : An Adaptive Octree for Efficient Ray Tracing. *IEEE Transactions on Visualization and Computer Graphics*, 1(4) :343–349, December 1995.

[85] T. Whitted. An improved illumination model for shaded display. volume 13, pages 1–14, August 1979.

[86] Sergej Zhukov, Andrej Inoes, and Grigorij Kronin. An ambient light illumination model. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98*, Eurographics, pages 45–56. Springer-Verlag Wien New York, 1998.

Accélération de la requête d'intersection par la réorganisation des rayons

La performance de la requête d'intersection, nécessaire à la simulation intensive de la propagation d'ondes, et indispensable aux rendus interactifs à base de lancer de rayons, est située au cœur du travail de cette thèse. En partant des meilleures structures de recherches et algorithmes de calculs d'intersections rayon/surface présents dans l'état de l'art, ce travail s'est concentré sur l'efficacité des groupes de rayons, sources de performance supplémentaire.
Cependant, à l'inverse des rayons simples, la cohérence nécessaire aux groupes de rayons pose la problématique de leur organisation spatiale pour approcher la performance optimale. Après avoir montré comment construire et traiter efficacement les groupes de rayons avec deux structures de recherche particulières, le cas des rayons à origine commune a été étudié, et optimisé, pour l'évaluation des sources ponctuelles et de l'occlusion ambiante.
Pour répondre à la problématique des rendus générant aussi bien des rayons cohérents que des rayons incohérents, une solution de couplage des structures de recherche est proposée en s'appuyant sur une stratégie globale d'affectation des rayons. Une autre solution, plus générale et destinée à exploiter la cohérence dans les cas les plus difficiles, est proposée pour regrouper automatiquement les rayons selon un critère a priori de cohérence. En se plaçant dans le contexte des modèles ayant une très forte complexité géométrique, la mise en place d'une pile de rayons, ainsi que sa distribution sur une grappe de calculateurs sont proposées afin d'améliorer la localité d'accès aux données, et de fournir une solution de visualisation interactive.
Pour permettre la réorganisation automatique des rayons, et donc faciliter l'exploitation des groupes de rayons, cette thèse conclut qu'il est impératif de définir une nouvelle interface de programmation, en remplacement de la simple requête d'intersection.

Mots clés : Lancer de rayons – Rayon SIMD – Groupe de rayons – Kd-Tree – Arbre BVH – Occlusion ambiante – Grappe de calculateurs – Très grands modèles

Ray query acceleration with bundle of rays

The core of this thesis is focused on raytracing performance, which is generally useful for intensive simulation of wave propagation and specially indispensable to render interactive image synthesis. Built on the best state-of-the-art acceleration structures and ray triangle intersection tests, this work has explored many ways of bundling rays together in order to improve this performance.
However, the coherency needed to obtain optimal ray bundle performance raises the specific and difficult problem of spatial organization compared to the classical use of single rays. This problem is decomposed and analysed in many levels in this thesis.
First of all, this work shows how to build efficiently ray bundles and how to use them with two kind of acceleration structures. Next, the special case of rays with common origin has been studied and optimized in the well-known contexts of point light sources and ambient occlusion.
In a more general context, an original coupled use of two acceleration structures is proposed to overcome the problem of mixed use of coherent and incoherent rays. Another solution, based on an a priori estimation of coherency, is proposed to reorganize automatically bundle of rays when coherency is not known or difficult to determine.
This thesis concludes that ray organization, essential to improve raytracing performance, is a difficult task that should be automated. However, this automation is only possible by redesigning the application programming interface and becomes a new challenge.

Keywords : Raytracing – SIMD Ray – Ray group – Kd-Tree – BVH Tree – Ambient occlusion – Computers cluster – King size