CHAPITRE III

Classification et Définitions

Mots clés

Lancer de rayon, volumes englobants, voxels, arbres CSG, arbres binaires, arbres octaux, parallélisation, vectorisation, graphes, classification, lancer de rayon, parallélisme

Résumé

La première partie de ce chapitre est consacrée à un survol concernant l'état de l'art des techniques d'accélération en lancer de rayon en donnant les grandes idées qui y sont rattachées. Cet état de l'art classique est en partie inspiré de l'article de J. ARVO & D. KIRK [Gl89].

La seconde partie de ce chapitre présente les éléments théoriques de base nécessaires à l'exposé de notre méthode d'optimisation. Ces éléments sont un ensemble d'entités algorithmiques, de relations entre ces entités ainsi que quatre graphes importants. Cette approche permet également de proposer une nouvelle classification des méthodes de lancer de rayon

Summary

The first part of this chapter is a survey of ray tracing speedup techniques and its main ideas. This classical state of the art is based upon J. ARVO and D. KIRK paper [Gl89].

In second part of this chapter, basic theoretical elements are described : algorithmic elements, relationships between them and four important graphs. They introduce the presentation of our ray-tracing speedup technique and a new classification for classical methods.

I) Classification classique des méthodes du lancer de rayon

La plus grande difficulté liée aux programmes de lancer de rayon est de produire un algorithme efficace en regard du temps d'exécution et de la qualité finale de l'image. Malgré les recherches entamées sur le sujet depuis les tout premiers travaux (fin des années 60), aucune étude n'a jamais pu réellement apporter une solution définitive. Aussi, nous nous proposons de faire un rapide survol des nombreuses méthodes d'optimisation connues à ce jour.

I.1) Considérations sur le choix d'une méthode d'optimisation

Il ne suffit pas d'optimiser un programme, encore faut-il que la version optimisée ne réduise pas trop fortement le champ d'application originel de ce programme. Aussi, la première partie de cet état de l'art sera consacrée aux questions qu'il faut se poser avant de rechercher une optimisation.

I.1.a) Critères d'applicabilité

* L'optimisation s'applique-t-elle à tous les rayons ou seulement à une partie de ceux-ci (rayons primaires, secondaires, d'ombrage...) ?
* Est-elle applicable dans le contexte CSG (Constructive Solid Geometry) ?
* Sur quels types d'objets s'applique-t-elle (facette, équation de surface...) ?
* Est-elle toujours applicable si la scène évolue dans le temps (animation) ?

I.1.b) Critères de performance

* Le résultat sera-t-il obtenu aussi rapidement que l'application le demande ?
* Comment les performances évoluent-elles en fonction de la taille de la base de données (complexité du problème) ?
* Quel est le coût de la phase de pré-échantillonnage (si il y en a une) ?
* L'algorithme exploite-t-il toutes les notions de cohérence ?

I.1.c) Critères sur les ressources

* Quel est le coût mémoire de la méthode (complexité mémoire) ?
* L'algorithme fait-il un bon usage de l'espace mémoire disponible par rapport au temps de calcul global ?
* Quel est le coût des opérations flottantes par rapport à celles en entier ?
* Veut-on utiliser une machine multiprocesseur, vectorielle ou dédiée ?
* Si l'on utilise une architecture parallèle, quel est le coût des communications ? De l'amorçage du réseau ?

I.1.d) Critère de simplicité

* L'algorithme est-il difficile à implémenter ?

* Est-il fortement lié à l'architecture de la machine utilisée (généralité du principe) ?

* Peut-on modifier du code déjà existant ou doit-on tout récrire ?

I.2) Une classification rapide des méthodes d'optimisation

La classification usuelle des méthodes d'optimisation [Gl89] fait apparaître trois stratégies (Cf. Figure III.1):

(1) Réduire le temps moyen d'intersection d'un rayon avec l'environnement,

(2) Réduire le nombre total de rayons lancés dans l'espace,

(3) Remplacer la notion de rayon par une notion plus générale et plus efficace.

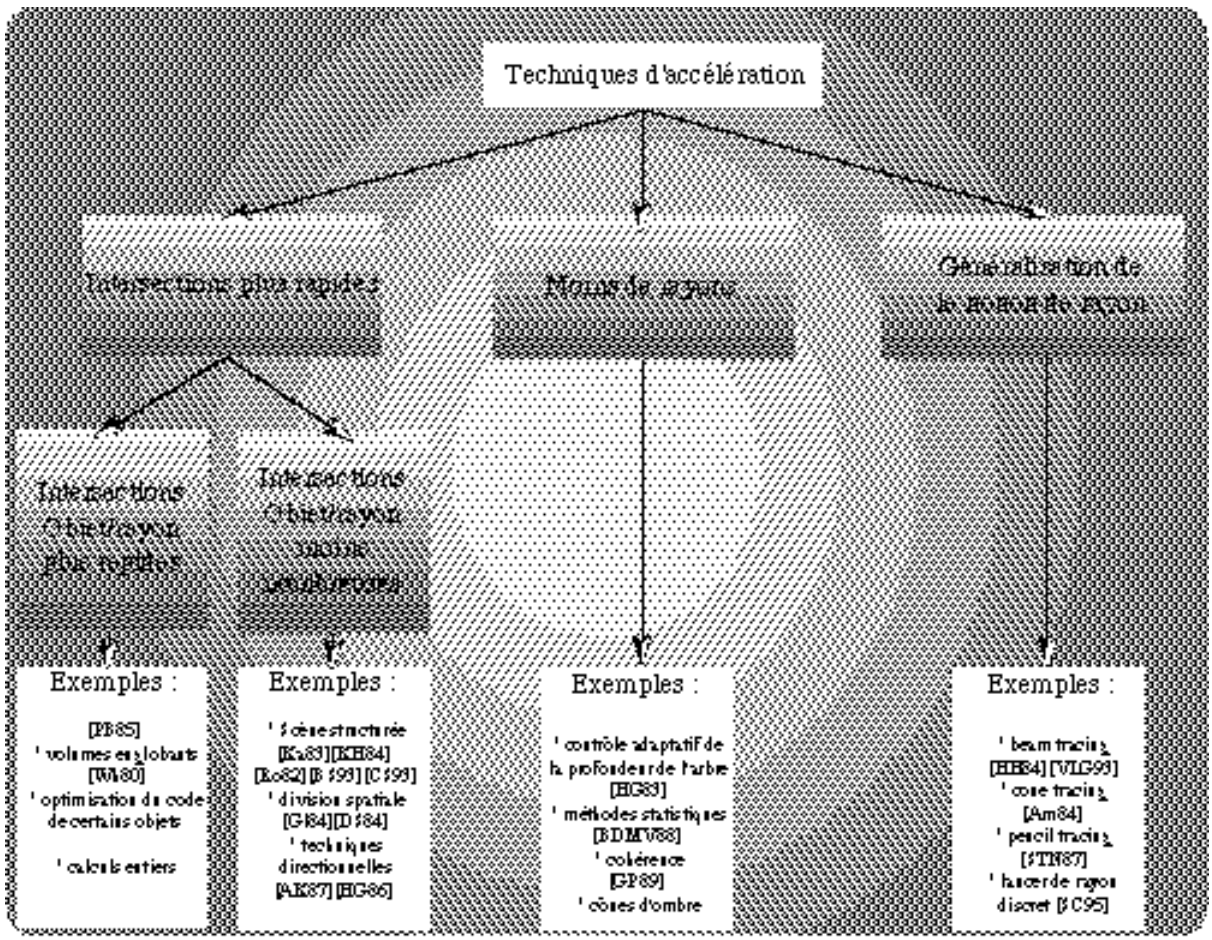


Figure III-1 : classification usuelle des optimisations ([Gl89] complétée)

Cette classification, complétée par des exemples nouveaux publiés après 1989, nous servira de plan pour présenter les différentes méthodes séquentielles d'optimisation. Notre présentation diffère cependant de l'article original sur de nombreux points structurels, sur certains paragraphes nouveaux (liés à des méthodes non présentées en 1989) et nous faisons apparaître en filigrane les éléments de base de la classification que nous introduirons dans la seconde partie de ce chapitre. De plus, on peut émettre quelques remarques :

- le parallélisme n'apparaît pas explicitement dans cette classification,

- les volumes englobants sont classés dans la même branche que l'optimisation simple du code alors que cela ne procède pas du tout de la même approche.

- comment pourrait-on classer une méthode de tracer de cônes couplée avec une notion similaire aux boîtes englobantes par exemple ?

- le tracer de cône par exemple, n'exprime-t-il pas intrinsèquement des notions de cohérence ?

Les optimisations parallèles sont en partie dérivées des optimisations séquentielles, mais l'utilisation de plusieurs processeurs offre des pistes d'optimisation supplémentaires. Lucas [Lu92] propose la classification suivante pour le lancer de rayon parallèle :

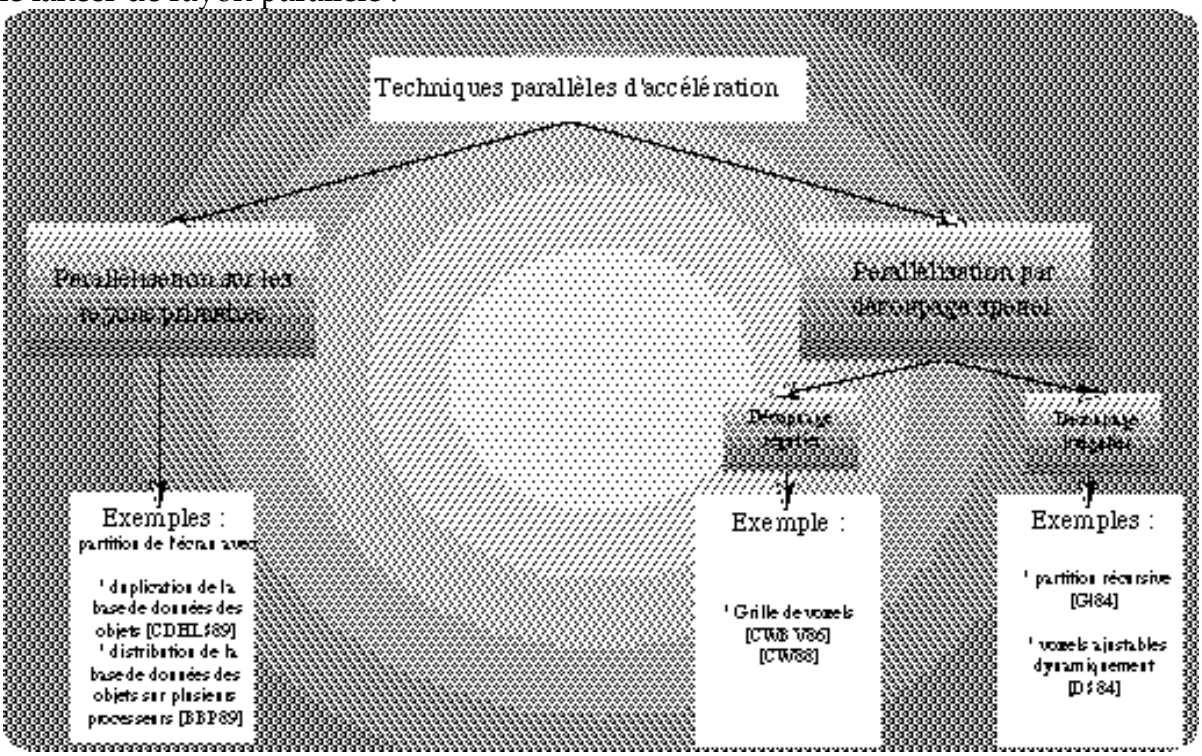


Figure III-2 : classification des méthodes parallèles

On peut également remarquer sur cette classification des méthodes parallèles, que même si des méthodes utilisant un processeur par objet peuvent difficilement être implantées à cause du nombre d'objets d'une scène, il ne serait pas absurde de faire apparaître une branche distincte prenant comme base de parallélisation les objets de la scène (ici elle est implicitement confondue avec les deux autres).

Nous reviendrons sur les problèmes soulevés par ces classifications après avoir présenté les différentes méthodes d'optimisation.

I.3) Intersections plus rapides : structuration des données

I.3.a) Les volumes englobants

I.3.a.1) Principe

Dès ses premiers travaux sur le lancer de rayon, Whitted constata que plus de 75% du temps de calcul était consacré aux intersections avec les objets. Il eut l'idée d'ajouter des volumes dont l'intersection est très simple pour englober un objet dont l'intersection est, elle, compliquée. Ainsi, si un rayon intercepte le volume englobant alors on regardera s'il intercepte l'objet contenu par celui-ci, sinon on passe à un autre objet (il y a plus d'intersections à calculer, mais les intersections inutiles sont beaucoup moins coûteuses). Cette technique est très efficace si la surface du volume englobant "colle" bien à celle de son contenu. Si ce n'est pas le cas, alors le fait d'avoir à calculer des intersections supplémentaires s'avère vite pénalisant [Wh80]. Cette méthode ne diminue pas la complexité (linéaire à chaque niveau de récursivité). Aussi, a-t-on eu l'idée d'introduire des volumes englobants hiérarchisés [RW80] qui permettent théoriquement d'obtenir une complexité logarithmique en fonction du nombre d'objets de la scène (si l'on ne considère par une boîte comme un objet). L'idée est d'englober (récursivement) un certain nombre de volumes englobants par un volume plus gros. Les données sont organisées en arbre, chaque noeud étant un volume pointant sur les objets qu'il contient.

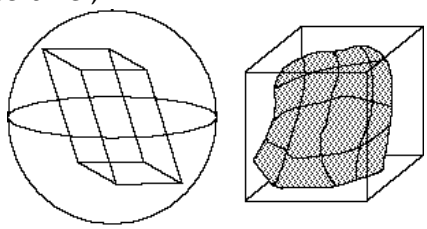


Figure III-3

Traverser une boîte englobante c'est traverser une zone de l'espace où l'on a des connaissances particulières. Le fait d'entrer ou de sortir d'une zone de l'espace clairement délimitée permet de déduire des optimisations. Cela est vrai pour les boîtes englobantes, mais on verra lors de la présentation des règles A_n (Cf. I du chapitre suivant), qu'un objet transparent en particulier, apporte le même genre d'information sur son espace intérieur et permet de déduire des optimisations du même type que les boîtes. La méthode des volumes englobants est très efficace mais pose le problème du nombre optimal de volumes que l'on doit ajouter à la scène originelle. En effet, plus il y a de volumes et plus finement est décomposée la scène mais hélas, plus le nombre d'intersections qui ne sont pas directement utiles pour le calcul final est important. En pratique, plus on a de boîtes englobantes et moins on risque de calculer inutilement l'intersection (complexe) avec l'objet mais les coûts calculs d'intersection avec les boîtes ne doivent pas être supérieurs au temps d'intersection avec l'objet final, d'où la recherche d'un compromis qui n'a pas de solution exacte dans le cas général (il dépend entre autres du type de la scène). A ce

problème, il faut ajouter celui du volume inutile contenu dans l'objet englobant. Il a été dit que pour que la méthode soit très efficace, il fallait que le volume englobant "colle" bien à l'objet. Or si ce dernier à une forme très complexe, il va falloir compliquer celle de son volume englobant et donc augmenter le coût du calcul d'intersection. Encore une fois, un compromis est à trouver (Cf. Figure III.4).

Des calculs de probabilités peuvent aider à trouver une certaine optimalité alors que d'autres méthodes cherchent à produire une meilleure hiérarchie en arrangeant au mieux les volumes ou encore en essayant de calculer l'enveloppe convexe des volumes (difficile [KK86]).

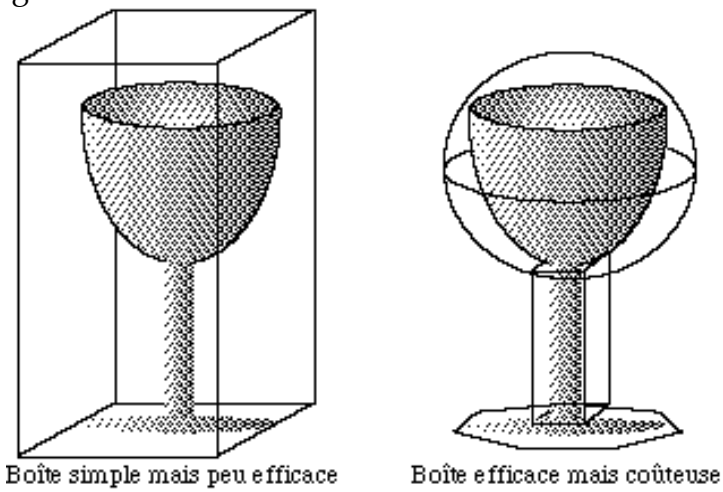


Figure III-4 : exemples de boîtes englobantes

I.3.a.2) Optimisation pour les arbres CSG

Appliquer le lancer de rayon à une scène définie par un arbre CSG (les feuilles de l'arbre binaire sont des primitives et les nœuds des opérateurs logiques d'union, intersection, différence, …) consiste, dans un premier temps, à rechercher classiquement les intersections entre chaque rayon et chacun des objets de la scène. Pour cela, on propage le rayon dans l'arbre CSG définissant la scène suivant l'algorithme de profondeur d'abord. Lorsque le rayon atteint une feuille, on calcule l'intersection entre l'objet mémorisé dans cette feuille et le rayon pour obtenir une liste des points d'intersections classées par ordre croissant de distance par rapport à l'œil. Lors de la remontée dans l'arbre, pour chaque nœud interne, on connaît la liste de ces points entre le rayon et les deux objets représentés par les sous arbres gauches et droits. On réalise le traitement correspondant à l'opérateur booléen mémorisé

dans le nœud interne sur ces deux listes pour construire une nouvelle liste des points d'intersections entre le rayon et la combinaison des deux sous- objets. Une fois la remontée terminée, on connaît l'intersection entre le rayon et l'objet complet en prenant le premier point de la liste d'intersections.

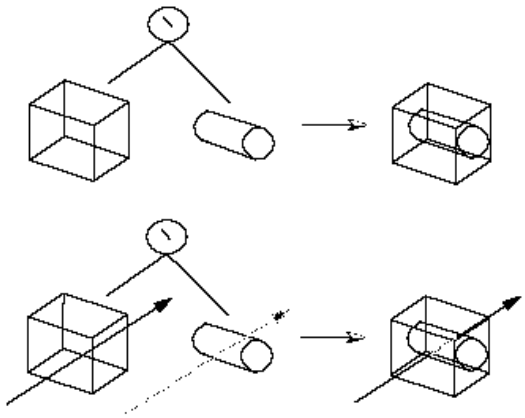


Figure III-5 : optimisation CSG

L'algorithme de lancer de rayon de Roth [Ro82] repose sur cette idée et il a utilisé les boîtes englobantes pour optimiser son algorithme (Cf. IV du chapitre VI). Au niveau de chaque nœud interne, on accède à deux boîtes, une contient son sous-arbre gauche et l'autre son sous-arbre droit. Avant de progresser en direction de l'un d'entre eux, on réalise l'intersection avec sa boîte associée (on utilise les connaissances données par la structure de l'arbre et le calcul rapide sur les boîtes). Il est possible d'utiliser des boîtes 2D applicables uniquement aux rayons primaires (projection sur l'écran) et des boîtes sphériques applicables à tous les niveaux. De plus, une optimisation sur les distances permet d'éliminer des sous-arbres CSG du calcul. Il est également possible d'utiliser au mieux les propriétés des opérateurs CSG (exemple, dans le cas de l'union, l'arbre peut être ré-équilibré sans affecter le résultat final, etc...).

Pour du lancer de rayon de premier niveau, Bronsvoort et al. [BWJ84], plutôt que d'utiliser des volumes englobants, utilisent les propriétés de cohérence d'un algorithme de lignes de prospection (Scan-line). En fait, ces notions de cohérence essaient de retrouver et d'utiliser des informations données par la structure CSG et qui peuvent fournir une optimisation. Des découpages de l'espace (souvent utilisés en parallélisme) ont été associés à la structure CSG par Wyvill et al. par exemple. Ils proposèrent d'associer octrees et arbres CSG [WKS86]. Arnaldi et al. [APB87] proposèrent une décomposition non régulière de l'espace associée à l'arbre CSG. Le problème sous-jacent de ces associations d'un découpage particulier de l'espace et de l'arbre CSG, est de trouver la bonne adéquation entre les structures de données (ici imposées) et l'architecture du programme qui va le traiter.

Cette méthode s'applique à tous les types de rayons et d'objets, mais pose le problème du choix du type de volume englobant et augmente sensiblement la taille de la base de données.

I.3.b) Division spatiale

À la différence des méthodes englobantes qui reviennent à regrouper de façon cohérente les objets de la scène puis à traiter le problème, on désire ici découper l'espace d'abord en réfléchissant sur la structure du programme puis en considérant éventuellement les objets composant la scène.

L'idée générale est de considérer l'espace de la scène dans sa globalité puis de le découper en sous-espaces ; c'est donc une approche descendante au contraire des volumes englobants qui partaient de chaque objet. Cette division de l'espace a donné naissance au concept de voxel, analogue 3D des pixels en 2D (Cf. Figure III-6).

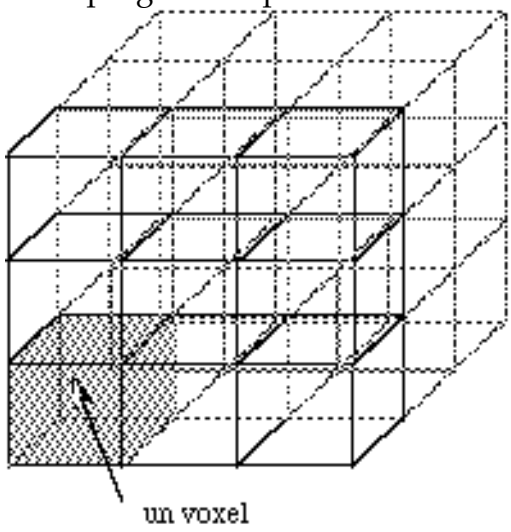


Figure III-6 : voxelisation

Un rayon parcourant l'espace, passera successivement d'un voxel à l'autre et l'algorithme n'aura à tester que les objets à l'intérieur de chacun des voxels considérés. Ce passage d'un voxel à l'autre se faisant dans l'ordre naturel de parcours du rayon, dès que l'on a résolu toutes les intersections à l'intérieur d'un voxel et retenu une solution, l'algorithme a trouvé la solution cherchée (si cet objet ne réfléchit ou ne transmet rien).

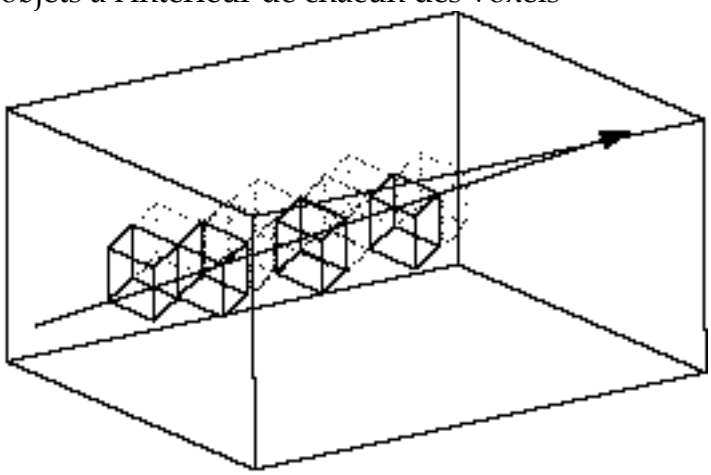


Figure III-7 : Voxels parcourus par un rayon.

I.3.b.1) Division spatiale non uniforme

Cette méthode essaie de tenir compte de la densité d'objets dans une région (donc de la "densité de calculs" que va effectuer le programme dans une zone donnée) en diminuant autant que possible la taille des voxels pour obtenir le moins d'objets à l'intérieur. La limite est atteinte lorsque l'on a zéro ou un objet ou encore

lorsque le nombre d'objets ne diminue plus par divisions successives. Les données sont alors organisées en octrees [YMS83].

Une autre idée du même genre est d'effectuer des divi-sions binaires récursives de l'espace (Binary Space Partition) [Ka85]. Les données sont alors organisées en arbre BSP (Cf. Figure III-8).

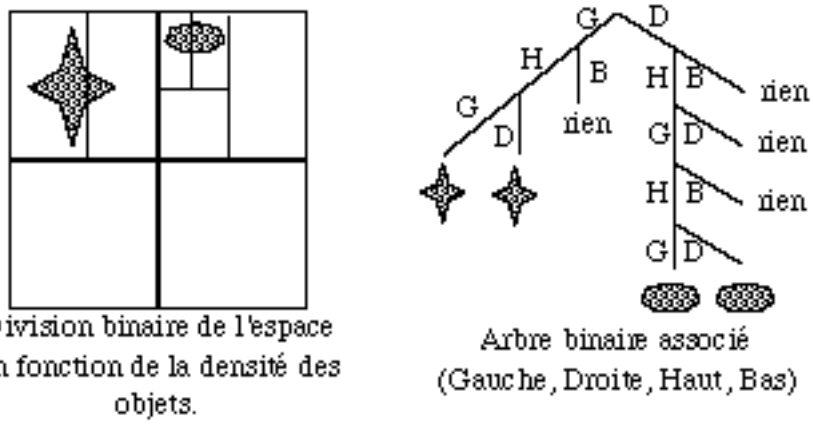


Figure III-8 : Partition binaire de l'espace

On notera de nouveau que plus il y a de voxels et plus efficace est le calcul d'intersection au niveau du voxel, mais qu'il est en contrepartie beaucoup plus difficile de gérer un arbre de forte profondeur et il faut tenir compte du fait que l'on ne dispose que d'une place mémoire limitée (adéquation entre la simplicité du traitement et la difficulté à retrouver les données). Là également, un compromis est à trouver.

I.3.b.2) Division spatiale guidée par les données

L'approche précédente considérait d'abord l'espace et le divisait selon des critères pré-définis. Une autre approche adoptée dans [CDP95] consiste à créer une partition en considérant d'abord les données selon le schéma :

- regroupement d'objets en sous-ensembles de taille similaire,
- pour chacun de ces sous-ensembles, regroupement des voisins en grappes,
- construction d'une grille 3D pour chaque grappe,
- construction d'une hiérarchie de ces grappes.

Cette approche est bien adaptée aux scènes très complexes et il est plus simple pour le programme de traiter les données, car la structure du programme et celle des données ont été traitées ensemble dès le départ. Elle tient à la fois de la division spatiale et de l'organisation des données du I.3.a).

I.3.b.3) Division spatiale uniforme

Le bénéfice cherché en utilisant une division spatiale régulière (maillage par exemple) est de simplifier la gestion des structures de données et par là même, d'essayer d'obtenir un gain de temps. En effet, si l'on ne prend plus en compte l'anisotropie de l'espace, on peut espérer simplifier l'algorithme de traversée de celui-ci par une méthode incrémentale, donc peu coûteuse en lignes de code [FTI86]

mais extrêmement coûteuse en mémoire (on résout le problème d'adéquation entre les données et le traitement par des duplications massives d'objets - doublons).

I.3.b.4) Ressources et applicabilité

Ces méthodes s'appliquent à tous les types d'objets et de rayons mais l'adjonction d'une structure arborescente sur les données peut poser des problèmes de mémoire.

I.3.c) Les techniques directionnelles

Cette approche date de la fin des années 80. L'idée est de prendre en compte des informations directionnelles (connaissance dans une direction donnée de l'espace) pour un groupe de rayons plutôt qu'un rayon individuel en construisant, pour une zone donnée de l'espace, une représentation de ce qui est vu dans différentes directions. Ces méthodes réutilisent des approches déjà vues comme les divisions spatiales.

I.3.c.1) Outils de base : le cube directionnel (direction cube)

Les méthodes directionnelles essaient de représenter l'espace à partir d'un point particulier. Pour cela, on a recours à des projections sur les faces d'un cube dont le point particulier est le centre, et qui est aussi l'origine d'un repère dont les axes passent par les centres des faces. Les faces de ce cube peuvent ensuite être discrétisées (uniformément ou non) pour avoir une vue par zone de l'espace.

Un cube directionnel est donc un cube aligné sur ces axes nommés +X, -X, +Y, -Y, +Z, -Z. On peut ainsi traduire les coordonnées 3D de l'espace en termes de coordonnées 2D sur une face du cube (par projection). Sur chaque face du cube, on définit deux vecteurs U et V pour exprimer ces coordonnées 2D (Cf Figure III-9).

Pour un rayon donné, on va chercher quelle face du cube il intercepte et trouver les coordonnées du point d'interception exprimé en fonction de u et v, coordonnées qui sont ramenées dans [-1..1].

Cette vue de l'espace permet d'effectuer les subdivisions spatiales de façon simple. Ainsi, chaque portion de l'espace est vue depuis le cube à travers une petite pyramide qui joue un rôle similaire aux voxels (ou plutôt aux faisceaux), bien que cette fois, la division spatiale soit attachée à chaque objet, ou encore à des points particuliers de l'espace comme les sources lumineuses.

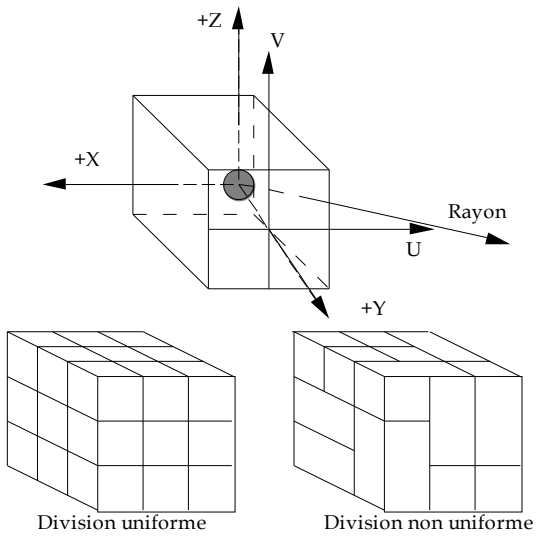


Figure III-9 : cubes directionnels

I.3.c.2) Le tampon de lumière (light buffer)

Le tampon de lumière de Haines & Greenberg [HG86] a pour but d'accélérer le calcul des ombres. On construit autour de chaque lumière, une connaissance des objets susceptibles de faire de l'ombre par le biais du cube directionnel (il y a donc une phase de pré-traitement). Tous les objets vus d'une source donnée, sont projetés par tampon de profondeur (Z-Buffer) sur les faces du cube directionnel.

Le calcul d'ombre pourra facilement se faire en déterminant quelle zone du cube directionnel traverse un rayon qui part d'un point donné de l'espace (ce rayon ne cherche plus alors à intercepter un ensemble d'objets, mais un seul cube, et, selon la zone d'interception avec ce cube, on obtient une liste d'objets susceptibles de faire de l'ombre).

Le tampon de lumière réutilise pour les rayons d'ombre une connaissance (grossière) pré-calculée de l'ombrage.

I.3.c.3) L'algorithme de cohérence des rayons (ray coherence algorithm)

Cette optimisation permet d'accélérer le calcul de "l'objet suivant" tout au long du parcours d'un rayon dans l'espace. Elle est utilisée avec des cubes directionnels associés à toutes les sources d'émission de rayons (œil, lumières, objets réfléchissants et transparents...) et optimise l'algorithme du tampon de lumière. Elle a été développée par Ohta & Maekawa [OM87] et s'exprime sous la forme d'une relation mathématique. Tout rayon originaire de l'intérieur de la sphère S1 et aboutissant dans la sphère S2 définit un angle θ avec la ligne

joignant les centres des deux sphères et si S1 et S2 ne s'intersectent pas, nous avons alors l'inégalité :

$$\cos(\theta) > \sqrt{1 - \frac{r1 + r2}{d(O1, O2)}}$$

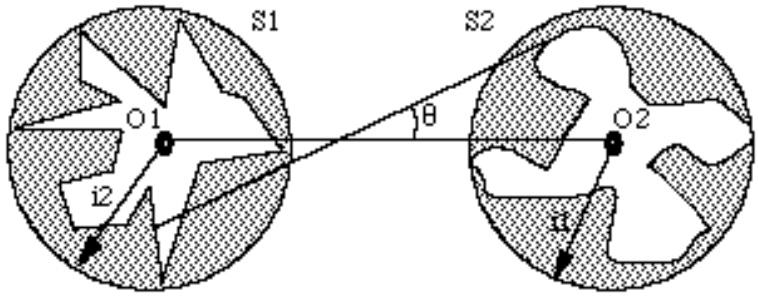


Figure III-10

Ohta & Maekawa ont étudié l'appliquation d'une telle relation à des objets englobants non sphériques (polyèdres convexes).

I.3.c.4) L'algorithme de classification des rayons (ray classification)

L'algorithme de classification des rayons d'Arvo & Kirk [AK87] est basé sur le fait qu'un rayon dans un espace 3D peut être vu par un formalisme à cinq dimensions correspondant aux points de $R^3 * S^2$ où S^2 est la sphère unité dans l'espace 3D et R^3 l'espace où sont exprimées les coordonnées du point de départ du rayon (le centre de la sphère). L'utilisation d'angles n'étant pas commode, la sphère est remplacée par un cube dont les faces correspondent aux directions dominantes. Un rayon avec une direction dominante est donc entièrement décrit par 5 composantes (x,y,z,u,v) où les trois premières coordonnées sont celles du point origine et les deux dernières celles de la direction associée à une face du cube directionnel. Tout rayon dans un espace 3D peut être décrit de cette façon. L'algorithme procède en partitionnant récursivement et adaptativement (suivant la densité d'objets) l'espace 5D des rayons en petits voisinages et en leur associant la liste complète des objets candidats à l'intersection. Le calcul d'intersection se fait alors en localisant la zone qui contient le rayon (dans sa représentation x,y,z,u,v) et en testant uniquement les objets qui y sont associés. Pour construire la liste des objets candidats à une intersection, on construit le faisceau décrit par le point de départ et la bonne zone de la face du cube adéquat. Une liste de candidats contient tous les objets touchés par ce faisceau. Ces objets peuvent être triés selon leur distance par rapport au point d'émission du faisceau.

Cet algorithme original ne semble pas donner de résultats nettement meilleurs que les autres méthodes directionnelles [Pi91].

I.3.c.5) Ressources et applicabilité

Ces méthodes sont assez gourmandes en mémoire et nécessitent des pré-traitements.

I.3.d) Des organisations de données particulières

Ces dernières années, les nouvelles méthodes d'optimisation ont porté assez largement sur des données dont l'organisation spécifique permettait d'accélérer les calculs. En effet, de façon analogue aux arbres CSG, une organisation adaptée des données induit des optimisations :

- traitement des splines et des surfaces de Bezier [BS93],
- traitement d'objets définis par leur densité [BSS93],
- rendu de relief naturel ou la visualisation géographique ("champs de hauteurs") [CS93] ,

I.4) Comment avoir moins de rayons ?

I.4.a) Exploiter la cohérence

Les optimisations font souvent appel à la notion de cohérence. Cependant cette notion est définie de façon assez vague. En effet, on explique principalement la cohérence par ses propriétés sur un aspect donné, mais il semble difficile de la définir en tant que telle. Cela est probablement dû au fait que ces notions expriment des relations complexes dans l'organisation des données ou du programme. Nous verrons qu'une partie du travail de cette thèse a consisté à définir cette notion sous forme de règles logiques et de graphes.

[FDFH93] donnent une définition générale de la cohérence (pour sa composante algorithmique) :

Définition : on exploite la cohérence lorsque l'on réutilise des calculs faits pour une partie de l'environnement, ou l'image, ou encore des morceaux proches, soit sans aucun changement ou avec des changements incrémentaux plus efficaces qu'un recalcul total de la solution.

Sutherland & al. [SSS74] ont identifié plusieurs types de cohérence pouvant être exploitées par les algorithmes de parties cachées.

Quatre d'entre elles peuvent l'être par le lancer de rayon et parmi celles-ci, celle de la cohérence des objets est la plus fondamentale ; elle exprime le fait qu'un objet n'est pas un nuage de particules distribuées aléatoirement dans l'espace mais un ensemble topologiquement cohérent. Algorithmiquement, [FDFH93] expriment cette propriété par : si un objet est entièrement séparé d'un autre, les comparaisons peuvent n'avoir besoin d'être faites qu'entre ces objets, et non entre leurs faces ou leurs arêtes (on retrouve une idée de boîte englobante).

La cohérence de l'image est dépendante du point de vue de la scène et exprime le fait que cette cohérence est conservée lors de la projection en 2D sur l'écran. Algorithmiquement : les propriétés des surfaces varient régulièrement sur celle-ci (modification incrémentale tout au long de la face). On distinguera :

* *la cohérence des arêtes* : la visibilité d'une arête change si et seulement si elle passe derrière une arête visible ou si elle traverse une face.

* *la cohérence d'arêtes d'intersection* : si deux faces planaires se coupent, l'arête d'intersection peut-être déterminée avec deux points d'intersection.

* *la cohérence des zones* : un groupe adjacent de pixels affiche souvent la même surface.

* *la cohérence de la profondeur* : les morceaux adjacents d'une même surface sont "bornés" en profondeur. Des surfaces différentes sont généralement séparées en profondeur.

La cohérence des rayons caractérise la propriété qu'ont deux rayons "similaires" d'avoir un parcours "similaire" dans l'espace (définition intuitive des choses - Cf. Figure III-11). Cette notion est plus difficile à exploiter que celle de la cohérence des objets par exemple. (On notera cependant que cette cohérence des rayons disparaît si les rebonds sont nombreux).

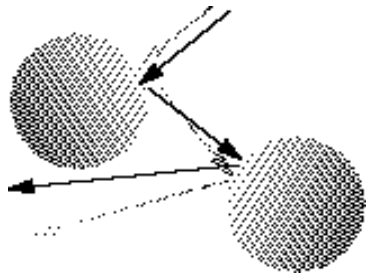


Figure III-11

La cohérence des déformations temporelles ("frame coherence") exprime le fait qu'une séquence d'images se calcule par déformations continues et non par sauts dans le temps et que deux images successives dans le temps sont généralement peu différentes.

Enfin, la cohérence des données est une cinquième propriété totalement algorithmique liées au fait que les calculs effectués à un moment donné ont besoin, de façon plus probable, des variables utilisées pour les calculs immédiatement précédents qu'à d'autres données [GP89].

En utilisant correctement ces propriétés, on arrive à simplifier bon nombre de calculs. Nous reviendrons sur ce sujet en donnant un ensemble précis de règles exprimant ces notions de cohérence.

I.4.b) Optimisations statistiques

L'introduction des statistiques dans le lancer de rayon a deux buts différents :

- générer des effets nouveaux de rendu (flou, pénombre, anti-aliassage…) en utilisant des intégrations de Monté-Carlo comme Bouville et al. [BDMV88] ou par d'autres approches [CPC84],[CO86],
- optimiser les calculs.

Seul le second cas nous intéresse ici. Il faut noter que l'optimisation statistique dans le cas des boîtes englobantes a déjà été traitée.

Le calcul d'une image est en fait le résultat d'un échantillonnage (généralement celui de l'écran). Des études ont été menées pour évaluer la précision nécessaire de ce calcul (et donc de la qualité finale de l'image) et plusieurs approches ont été adoptées pour fixer au mieux l'échantillonnage [LRU85], [PU86].

I.5) Notion de rayons généralisés

Un rayon est une demi-droite mathématique, ce qui a différentes conséquences négatives :

- effets d'aliassage (crénelage classique par exemple),
- besoin d'un très grand nombre de rayons si l'on veut introduire des effets comme la pénombre,
- difficulté d'exprimer les notions de cohérence pour les rayons.

Pour résoudre ce problème, une idée est de considérer non pas un rayon, mais un ensemble de rayons liés entre eux, et qui sera vu comme un faisceau, un cône ou encore un "pinceau". Cette approche cherche à généraliser la notion de rayon. Cependant, il faut abandonner l'espoir d'avoir des intersections "exactes" ; d'un autre côté, on obtient une exécution plus rapide de l'algorithme, un effet de lissage de l'image ainsi que quelques effets d'optiques.

I.5.a) Lancer de cône

Le rayon en tant que demi-droite mathématique a été abandonné par Amanatides [Am84] au profit des cônes circulaires décrits par un sommet, une droite centrale et un angle ("cone tracing", Cf. Figure III-12). Pour calculer la valeur d'un pixel (avec ombre et pénombre) et le lissage, on a alors besoin de savoir si le cône intercepte un objet, mais également dans quelles proportions (l'objet est-il totalement ou partiellement dans le cône ?).

La réflexion et la transmission sont calculées en déterminant un nouveau cône réfléchi ou transmis à partir de la surface d'impact du cône. Cependant, cette méthode oblige à ne considérer que des objets de type sphères, plans ou encore polygones.

Tracer de cône

Figure III-12

I.5.b) Lancer de faisceau

La géométrie des cônes est toujours la même. Heckbert & Hanrahan [HH84] ont introduit la notion de faisceau ("beam tracing") qui généralise celle de cône, puisqu'au lieu d'avoir une section circulaire, on considère une section polygonale quelconque (Cf. Figure III-13). Cependant, avec cette méthode, les objets doivent être

entièrement définis par des facettes polygonales et planaires pour conserver la nature polygonale de la section du faisceau. Les effets de réflexion ou de transmission ne respectant pas nécessairement cette nature polygonale, on réalise une approximation de ceux-ci. Le grand avantage des faisceaux par rapport au rayon est qu'ils peuvent être partiellement occultés par une surface (on découpe alors le faisceau avec la silhouette du volume rencontré).

Le lancer de faisceau génère trois types de problèmes : intersection d'objets, tri d'objets et détourage ("clipping"). Dadoun et Kirkpatrick [DK85] ont montré que ceux-ci pouvaient être simplifiés par une représentation hiérarchique de la scène.

Figure III-13

Robin et al. [VLG93] ont implanté une méthode entre le tracer de faisceau et le lancer de rayon classique.

I.5.c) Tracer de crayon

Enfin, une dernière généralisation a été proposée par Shinya & al. [STN87], à savoir le tracer de crayon ("pencil tracing"). Un crayon est un ensemble de rayons qui sont dans le voisinage d'un rayon particulier appelé le rayon axial. Chacun d'eux peut être représenté comme un vecteur 4D décrivant la déviation par rapport au rayon axial. Shinya et al. résolvent les intersections de ce groupe de rayons avec l'environnement par diverses techniques mathématiques, et dans le cas où l'on ne considère que les rayons ayant une faible déviation par rapport au rayon axial, on peut faire l'approximation de la linéarité du crayon et il est possible de se ramener à un système matriciel 4*4. Cette approximation est valable seulement si l'on rencontre des surfaces continues et sans arête vive. Dans ces derniers cas, il est toujours possible d'utiliser le lancer de rayon classique.

I.5.d) Points communs de ces trois extensions

Ces trois méthodes peuvent parfaitement trouver leur équivalent dans le lancer de rayon classique. Il faudrait pour cela, lancer autant de rayons que nécessaire pour remplir le cône par exemple. Implicitement, ces méthodes donnent un formalisme qui permet de lier tous ces rayons au rayon axial. Nous avons ici une formulation analytique de la cohérence d'un groupe de rayons.

I.5.e) Le lancer de rayon discret

Par rapport aux trois paragraphes précédents, le lancer de rayon discret est une généralisation un peu différente de la notion de rayon dans le sens où l'on ne travaille plus sur un espace mathématique continu, mais sur un espace discret. Les rayons sont des demi-droites au sens de Bresenham et les objets sont créés par voxelisation. On recherche par cette approche, à accélérer les calculs par la simplicité des opérations à effectuer. Cette méthode présente un certain nombre de difficultés comme la définition discrète des normales, des effets importants d'aliassage et la génération de grosses bases de données [BA94][SC95].

I.6) Les techniques parallèles et vectorielles

Malgré la diversité des optimisations, le problème de la quantité de calcul à effectuer n'a pu qu'être temporairement repoussé. Aussi, en plus des optimisations algorithmiques, il faut s'attacher à examiner des solutions plus matérielles, à savoir la vectorisation et la parallélisation des algorithmes. Ces solutions s'inspirent bien souvent des techniques que l'on vient de décrire.

I.6.a) Approches vectorielles

Max [Ma81] a vectorisé un programme de lancer de rayon capable de simuler des vagues et des îles. Ce programme tourne sur Cray-1. Les rayons primaires, les calculs d'ombre ainsi que les rebonds étaient vectorisés. Plunket & Bailey [PB85] proposèrent un algorithme plus général tournant sur CDC Cyber 205. Celui-ci était de la forme :

```
Tant qu'il reste des pixels non calculés faire
Ajouter des rayons primaires, réflexions et ombres dans la queue
jusqu'à ce qu'elle soit pleine.
Réaliser les calculs d'intersection de chaque rayon dans la queue
avec chaque surface de la scène utilisant le code vectoriel.
Déterminer la surface visible pour chaque rayon en utilisant une
évaluation de type CSG.
Générer des rayons supplémentaires pour les effets spéciaux et les
mettre dans la queue.
Calculer la couleur finale de chaque pixel.
Fait ;
```

Hélas, la vectorisation complique quelque peu l'algorithme et nécessite beaucoup plus de mémoire pour pouvoir résoudre les problèmes de calculs sur l'arbre CSG alors que d'un autre côté, on ne peut pas augmenter à l'infini le nombre d'informations envoyées dans la queue.

D'une manière générale, ces algorithmes permettent d'obtenir un gain de temps sensible.

I.6.b) Les machines dédiées

Des machines spécialement conçues pour l'infographie ont été étudiées. LINKS-1 [LOKSO83] est une de ces machines. Elle est de type vectoriel mais également parallèle. En effet, elle est composée de 64 processeurs plus un, qui sert de contrôleur et peut dynamiquement reconfigurer tout le réseau (pipelines en parallèle). Une unité est spécialement dédiée au transfert de données entre processeurs.

Ceci n'est qu'un exemple et il en existe quelques autres comme Pixel-Planes [FP81] ou la machine Ray-Casting [KE84]. On notera toute fois que les algorithmes s'exécutant sur ces machines sont spécialement conçus dans cette optique et sont peu généralisables. De plus, il reste encore beaucoup de travail pour avoir un lancer de rayon complet (avec plusieurs niveaux de récursivité) calculé par une machine spécialisée.

I.6.c) Des machines parallèles plus générales

De nombreux travaux ont été publiés sur les programmes de lancer de rayon utilisant le parallélisme asynchrone avec des topologies classiques style hypercube ou autres. On pourra se rapporter à [GS85][CPC84][GP89][PRIOL][KH95] pour avoir une bonne idée de la question. D'un point de vue général, on peut distinguer trois méthodes : une parallélisation processeur par pixel, une parallélisation processeur par voxel et un processeur par objet.

I.6.c.1) Approche processeur par pixel

Cette approche a été principalement explorée avec Pixel-Planes [FP81] et rejoint la branche de la parallélisation sur les rayons primaires selon la classification de Lucas. L'idée générale est très simple : à chaque pixel , on va associer un processeur. C'est donc un parallélisme massif où l'on compte essentiellement sur le nombre pour obtenir un résultat rapide mais on sous-utilise fortement la plupart des processeurs car la charge de calcul est très mal répartie avec cette approche. D'autres études ont été menées en partant de cette idée [CDHLS89].

I.6.c.2) Approche processeur par voxel

La seconde de ces méthodes consiste essentiellement à découper l'espace en voxels (division spatiale adaptative ou non [DS84]) et à affecter un processeur à chaque zone (seconde branche de la classification de Lucas). Un rayon partant de l'œil va traverser l'espace et chaque fois qu'il entrera dans une zone, il sera pris en compte par le processeur concerné qui effectuera les tentatives d'intersection avec

les objets qu'il contient. S'il réussit à calculer une intersection, il relance dans l'espace des rayons de calcul d'ombre, des rayons transmis, réfléchis s'il en est besoin et obtient ainsi le résultat cherché.

Cette stratégie pose parfois des problèmes de partage de base de données plus ou moins bien résolus et des détections de fin de calculs parfois difficiles [Pr89].

I.6.c.3) Approche processeur par objet

La dernière stratégie possible est d'associer chaque objet à un processeur particulier. Ainsi, si un rayon est lancé, il est envoyé à tous les processeurs qui renvoient les résultats d'interception si elles existent.

Cette méthode demande un très grand nombre de processeurs et génère énormément de communications (nous avions avancé cette hypothèse pour expliquer l'absence de cette branche dans la classification de Lucas).

I.6.d) Conclusion sur le parallélisme

On notera que les algorithmes sont souvent très dépendants de la machine sur laquelle ils tournent et que le parallélisme est parfois assez brutal dans le sens ou les processus tournant en parallèle ne coopèrent pas réellement entre eux (par exemple, ils transmettent un rayon d'une zone à une autre sans transmettre des informations qui pourraient être utiles aux calculs suivants). Nous verrons au chapitre suivant comment changer cet aspect des choses.

II) Conclusion sur cette classification classique

Les classifications de Lucas ainsi que d'Arvo et Kirk rangent les nombreuses optimisations du lancer de rayon dans des branches distinctes mais le parallélisme semble apparaître comme une méthode à part. Nous verrons qu'il est possible de mieux distinguer les différents éléments du lancer de rayon et que ce sont ces éléments qui guident la démarche d'optimisation que l'on retrouve dans ces deux classifications, tant en séquentiel, qu'en parallèle.

Ces dernières années, les travaux en lancer de rayons se sont ralentis. Des domaines connexes ont également été étudiés :

- étude quantitative de méthodes existantes [GA93],

- développements du lancer de rayon dans le cadre de l'animation [MDC93] en introduisant le temps comme paramètre de calcul.

En près de vingt ans, les algorithmes de lancer de rayon ont fortement progressé sur le point de l'optimisation. Cependant, et malgré ces progrès notables, le calcul d'images complexes en moins d'une seconde avec cette méthode est toujours impossible. Aussi, et malgré le grand nombre d'idées d'optimisation déjà publiées, de nouvelles voies doivent être découvertes et explorées.

III) Définitions et nouvelle classification

La classification classique des méthodes d'accélération en lancer de rayon que nous venons de présenter n'est pas totalement suffisante pour plusieurs raisons :

- elle ne fournit pas vraiment un ensemble de critères précis qui permettraient de classer parfaitement une optimisation dans une catégorie donnée, surtout si cette approche mixte différentes optimisations,
- certaines méthodes semblent se retrouver dans différentes branches de cette classification,
- elle est basée sur des techniques existantes et peut se révéler incomplète pour d'autres à venir,
- elle n'indique pas la possibilité de nouvelles pistes de recherche,
- elle distingue trop nettement parallélisme et traitement séquentiel.

Dans cette seconde partie, nous proposons une formalisation du lancer de rayon qui d'une part est utile pour la présentation générale de notre méthode et d'autre part peut permettre de classer plus rigoureusement les différentes optimisations. L'idée centrale présentée dans ce mémoire s'appuie sur ces définitions théoriques importantes qui sont ensuite associées à l'environnement informatique disponible.

Cette séparation de l'aspect théorique et de la réalité physique est fondamentale par rapport à l'hypothèse de portabilité maximale de la méthode d'optimisation proposée.

III.1) Description du lancer de rayon à l'aide de graphes

Les définitions suivantes seront illustrées par l'exemple d'une méthode de lancer de rayon ayant les caractéristiques suivantes (Cf. Figure III-14) :

- la scène est structurée par une hiérarchie de boîtes englobantes,
- le calcul d'ombre est optimisé par un tampon de profondeur (recours à un cube directionnel sur les sources)
- l'écran est décomposé en quatre zones selon une topologie carrée
- on considère une scène composée de trois objets et de cinq boîtes englobantes.

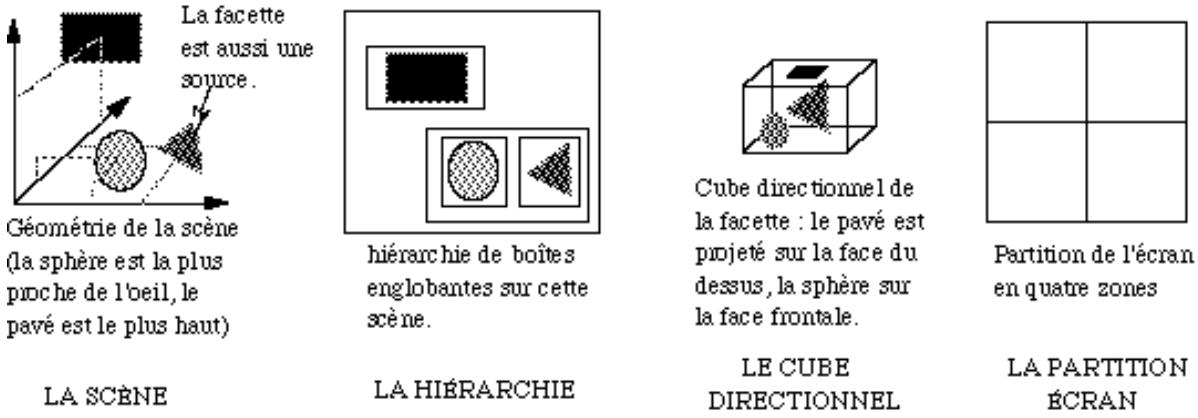


Figure III-14 : exemple de base

Une autre petite série d'exemples sera ensuite donnée.

III.1.a) Définition de trois entités algorithmiques pour le lancer de rayon

III.1.a.1) Les objets réels

Les objets réels se définissent comme les éléments géométriques de base qui composent la scène et qui sont associés à une procédure de calcul d'intersection élémentaire. Les objets réels de base sont généralement connus par l'équation de leur surface. À chaque objet est associée de façon univoque une description photométrique. Ils peuvent être simples comme des facettes, des volumes quadriques, appartenir à une hiérarchie comme les feuilles d'un arbre CSG, ou les boîtes englobantes d'une hiérarchie de boîtes englobantes. Les boîtes englobantes dont on est amené à calculer l'intersection avec un rayon sont donc considérées comme des objets réels. L'espace pourrait être vu comme un objet réel d'une géométrie fixée a priori, et dont la couleur serait la couleur ambiante, qui serait transparent et qui contiendrait tous les autres objets.

On notera E_r l'ensemble des objets réels.

On distinguera dans E_r, $O = \{O_1, O_2, \dots O_n\}$ qui est l'ensemble des objets et $B = \{B_1, B_2, \dots B_m\}$ qui est l'ensemble des boîtes englobantes associées à l'ensemble O des objets de la scène ($m \neq n$ en général), $E_r = O + B$.

On définit le graphe des objets réels noté G_r comme suit :

- les sommets s de G_r sont des sous-graphes $G_r'(s)$ construits sur des sous-ensembles de E_r ; si $G_r'(s)$ n'a pas d'arête, il sera réduit à un sous-ensemble de E_r

- les arêtes de G_r (resp. de $G_r'(s)$) définissent une relation entre deux sommets (par exemple la contiguïté entre les voxels contenant les objets associés aux sommets, l'appartenance à un même volume englobant, …). G_r (resp. $G_r'(s)$) peut être réduit à un ensemble déconnecté d'objets, mais de façon générale il sera plus ou moins structuré,

- il est éventuellement possible de considérer plusieurs graphes G_r correspondants chacun à une vision particulière du problème.

On peut déjà constater que plus G_r est structuré au lancement du programme, plus le logiciel de rendu dispose d'informations au début de ses calculs.

Les objets de notre exemple de base (Cf. Figure III-14) peuvent être regroupés de différentes façons. Par exemple :

Solution 1)

La première solution pour construire G_r avec notre exemple de départ, est de considérer les quatre zones écrans et la projection des objets sur ces zones sans tenir compte des boîtes englobantes. Les sommets du graphe vont marquer l'appartenance des objets contenus à une même zone et les arêtes la connexité (Nord, Sud, Est, Ouest) des différentes zones. On remarquera le doublon lié à la sphère qui indique une des faiblesses de cette approche. À l'intérieur d'un sommet s, les éléments ne sont pas structurés, $G_r'(s)$ est un graphe totalement déconnecté.

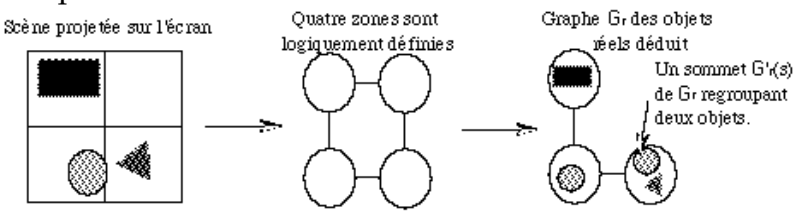


Figure III-15

Le graphe ainsi créé fait apparaître des informations exploitables algorithmiquement : ici, pour un sommet donné, on ne traitera qu'un ensemble réduit d'objets et les primitives d'intersection à utiliser par voxels apparaissent dans les sommets de G_r. Il est aussi possible de considérer des aspects quantitatifs comme des disparités dans la répartition car ici on avait a priori quatre sommets possibles et seulement trois ont eu besoin d'être créés (ce qui peut se traduire ensuite dans l'algorithme par une mauvaise répartition de la charge de calcul).

Solution 2)

Si on considère en priorité la hiérarchie d'englobement, on crée une autre vision de G_r qui exprime d'autres propriétés sur ces données (Cf. Figure III-16). G_r est une arborescence et une telle organisation des données devrait normalement aboutir à rechercher des

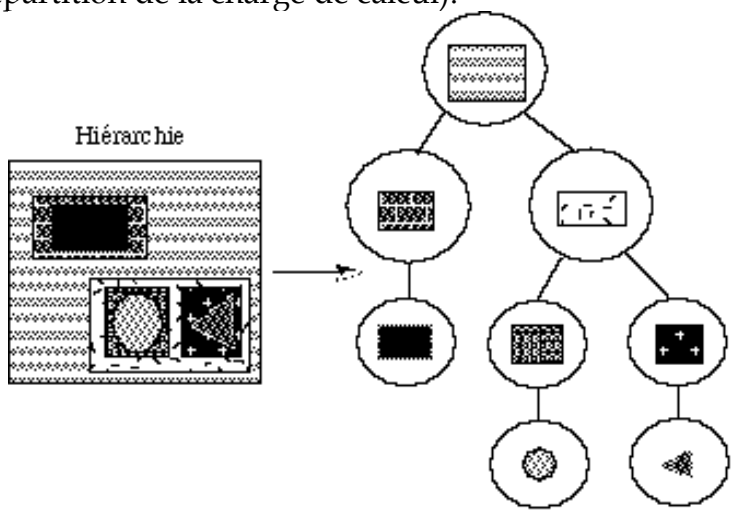


Figure III-16

algorithmes bien adaptés à cette structure et on pourra chercher à extraire des critères comme l'équilibrage de l'arbre pour guider la recherche d'optimisation.

Solution 3)

Les deux visions de la scène ont abouti a deux graphes différents. L'algorithme pourra commencer avec un G_r ayant deux parties totalement disjointes, mais on peut aussi chercher une solution qui regroupe les deux descriptions. Et dans ce cas, on disposera en même temps d'une double source d'information pour essayer de déduire des optimisations. La solution est ici relativement simple, il suffit de structurer chaque sommet de la solution 1 par une arborescence issue de la solution 2 (Cf. Figure III-17).

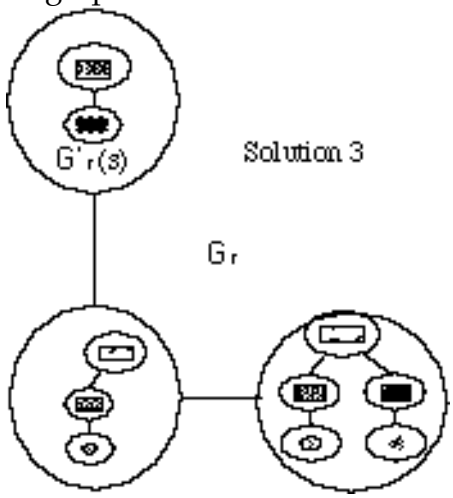


Figure III-17

De manière générale, G_r peut selon les cas être fixé définitivement au début de l'algorithme ou évoluer dynamiquement (par exemple en complétant la base de données si celle-ci est répartie). Cette évolution ainsi que la connaissance supplémentaire acquise (création de sous-graphes, d'arêtes, ...) caractérise également le niveau d'information utilisable par l'algorithme.

Solution 4)

Enfin, il existe un dernier graphe possible qui peut être créé à partir de la projection des objets sur le cube directionnel. Cependant, ce graphe ne sera pas utilisé seul, mais en association avec le graphe des objets virtuels.

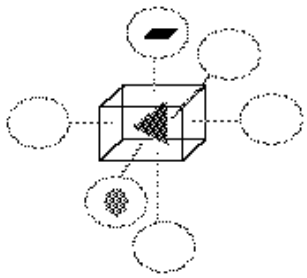


Figure III-18

Dans le cas d'une méthode utilisant plusieurs optimisations très différentes, G_r peut être composé de sous-graphes disjoints n'ayant aucun rapport entre eux. Sur notre exemple de base (Cf. Figure III-14), il existe un graphe des objets réels intégrant les informations de la hiérarchie de boîtes englobantes et la partition de l'écran (Cf. Figure III-17) qui donne une vision totalement différente du graphe créé pour le cube directionnel (Cf. Figure III-18).

Cela peut dénoter un problème structurel de l'implémentation d'une telle méthode.

III.1.a.2) Les objets virtuels

Ces objets sont ceux qui servent à calculer les informations lumineuses selon le modèle utilisé. Dans le cas le plus simple, ce sont les rayons (primaires, secondaires -

réfléchis ou transmis - ou d'ombres). Un rayon est généralement connu par un point de départ et un vecteur directeur mais il faut également ajouter à cette liste des objets virtuels, les rayons généralisés utilisés en tracer de cône, faisceau ou de crayon (pencil tracing), ainsi que les rayons secondaires créés stochastiquement ou statistiquement.

On notera E_v l'ensemble des objets virtuels.

On définit le graphe des objets virtuels noté G_v comme suit :

- les sommets s de G_v sont des sous-graphes $G_v'(s)$ construits sur des sous-ensembles de E_v ; si $G_v'(s)$ n'a pas d'arête, il sera réduit à ce sous-ensemble de E_v
- les arêtes de G_v (resp. de $G_v'(s)$) définissent une relation entre deux sommets (comme la contiguïté entre les deux voxels regroupant les rayons qui les traversent),
- il est éventuellement possible de considérer plusieurs graphes G_v .

Le graphe G_v traduit le type et la qualité des informations préalables dont on dispose (rayons dans la même zone de l'espace, associés à la même lumière, …). Par exemple, dans un cas défavorable, on peut considérer qu'il n'y a aucune relation entre les rayons (G_v a pour sommets des éléments de E_v et est totalement déconnecté), mais on peut tenir compte du fait qu'un rayon réfléchi ou transmis possède un père, ou que deux rayons passant par deux pixels voisins sont eux aussi voisins, ou qu'un ensemble de rayons appartienne à un même cône, etc… Là aussi, il existe un gisement préalable d'information qui peut être traduit dans la structure de G_v et exploité pendant les calculs.

NB : Chaque rayon primaire passant par un pixel de l'écran, l'ensemble de ces pixels est donc associé à une partition naturelle de E_v. Les pixels ne sont donc pas des entités algorithmiques indépendantes. Si on choisit pour sommets de G_v le regroupement des rayons issus de chaque rayon primaire, alors les sommets de G_v sont naturellement associés aux pixels.

Sur notre exemple, nous pouvons encore distinguer trois structures possibles pour G_v.

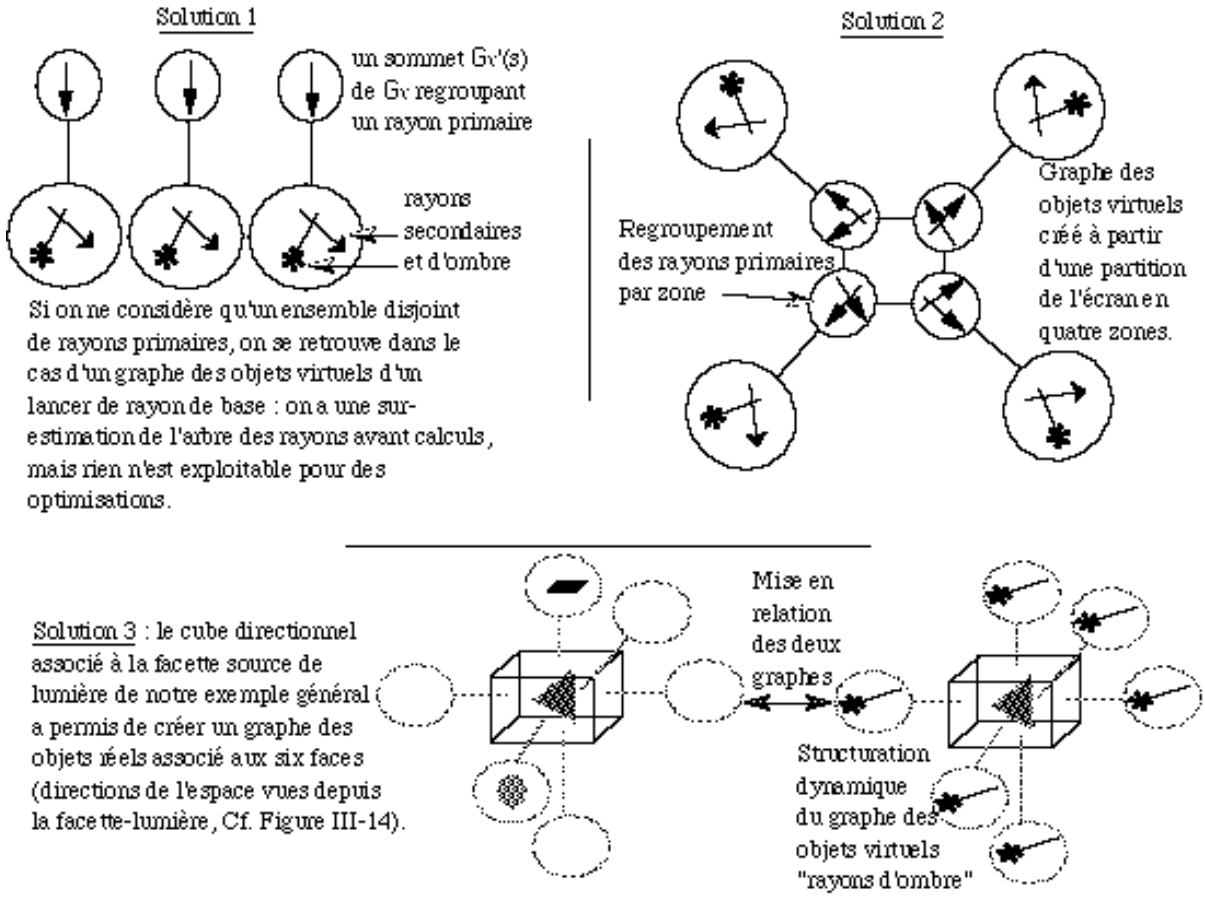


Figure III-19

Sur l'exemple de la Figure III-19, on considère trois solutions possibles :

- la première a un degré d'information utilisable nul (pour le lancer de rayon de base, on considère les rayons primaires de façon totalement indépendante et les rayons secondaires et d'ombrage qui en sont issus ne bénéficient pas d'information utilisable)

- dans la deuxième, le graphe créé à partir de la partition de l'écran, bénéficie d'information comme l'appartenance à une zone écran, la connexité entre deux zones, etc...

- enfin la troisième solution est un peu particulière. Pour pouvoir la créer, on a d'abord considéré les sources de lumière, créé un graphe des objets réels associé aux faces du cube de chaque source (Cf. Figure III-18) et, avec la même structure liée aux faces du cube, on a construit, pour une lumière donnée, une organisation des rayons d'ombre associée à ces six faces du cube directionnel. Puis, pour un rayon d'ombre donné, une fois que l'on a déterminé auquel des six groupes appartient ce rayon, on n'a plus à considérer que les objets réels liés à la même face du cube : les sommets des deux graphes sont donc naturellement et fortement liés entre eux.

Les deux dernières solutions fournissent des voies d'optimisation. Cependant, à la différence des solutions 1 et 2 sur G_r, l'association des deux approches (partition écran et cube directionnel) est moins immédiate et demande un peu plus de travail.

En effet, pour réaliser cette fusion, on va structurer les sous-graphes contenant les rayons secondaires et d'ombre en isolant un groupe de rayons secondaires et six groupes de rayons d'ombre pour chaque cube directionnel. La réorganisation des rayons d'ombre implique pour chaque rayon d'ombre, de calculer à quel sommet (lié à une face du cube) il appartient. Pour les rayons secondaires, on n'a pas à faire ce traitement, car tous les rayons réfléchis ou transmis du même arbre de calcul appartiennent au même sommet. Les rayons d'ombre sont donc plus structurés, mais cela a un coût algorithmique.

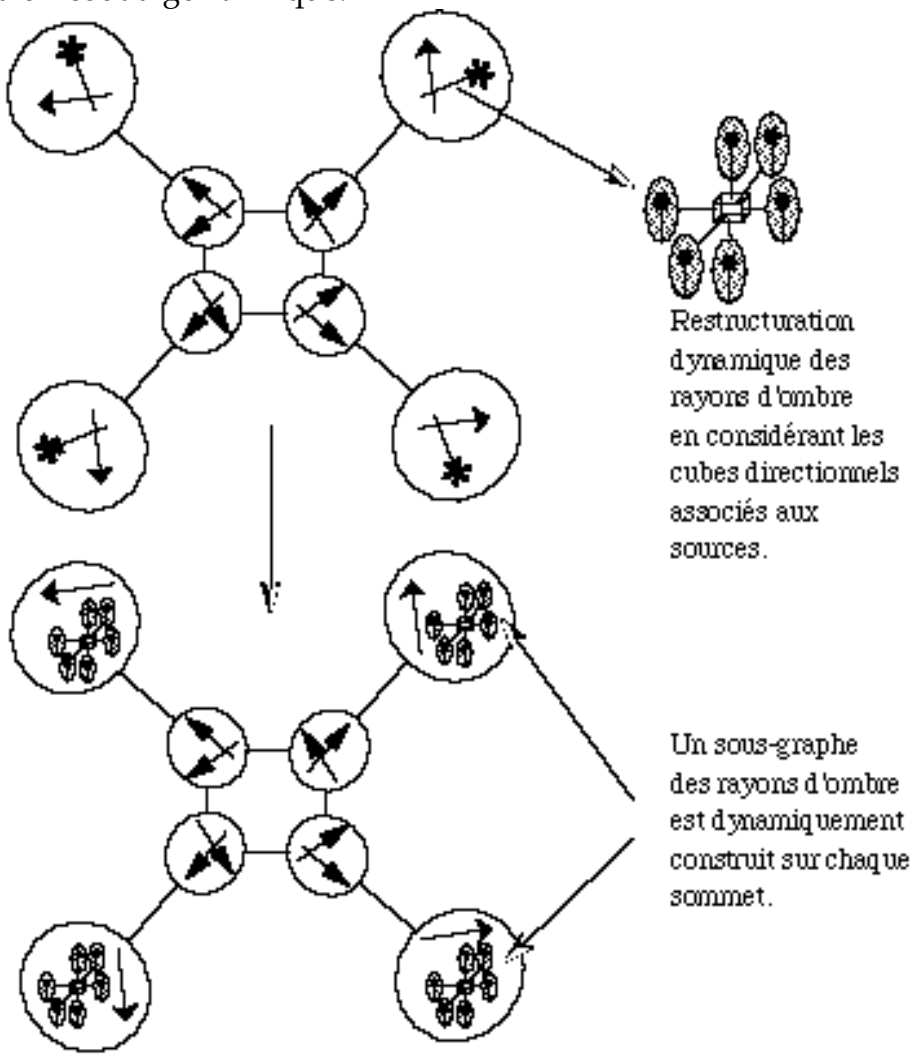


Figure III-20 : réorganisation en fonction du cube directionnel

De manière générale, le ou les graphes G_v peut évoluer en cours d'algorithme. Dans certains autres cas comme celui d'une parallélisation selon les voxels de l'espace, ce graphe est presque inconnu au début de l'algorithme et est construit au fur et à mesure des calculs (ici par échanges de rayons). On notera que les rayons secondaires ou d'ombrage arrivent plus difficilement à être structurés en début de programme : seul des algorithmes comme ceux utilisant des rayons généralisés (un

cône = un ensemble de rayons) ou celui présenté à la solution 3, introduisent de l'information par un pré-traitement.

III.1.a.3) Les processus

Un processus (on utilisera de façon synonyme le mot tâche) est un élément algorithmique qui regroupe un ensemble de calculs ou de données. Un processus peut ne contenir que les éléments de déclaration d'un objet par exemple, ou inversement ne contenir aucune donnée de façon permanente. Ainsi, on peut réduire une tâche à un objet ou encore à une fonction d'intersection.

On notera P l'ensemble des processus.

On définit le graphe des processus comme le graphe dont les sommets sont les tâches et dont les arêtes symbolisent une communication bidirectionnelle entre deux sommets (processus). Il sera noté G_p et comme il exprime la structure d'un programme il est unique.

Soit p P. Pour tout p_i tel (p,p_i) est un arc de G_p, on supposera en général :

- si (p,p_i) existe alors (p_i,p) existe aussi (liaison bidirectionnelle),
- p_i sera appelé voisin de p,
- l'ensemble des voisins de p sera noté V(p),
- les arcs a^+ et a^- associés à une arête sont respectivement associés à une étiquette d^+ et d^- (étiquette que l'on nommera aussi direction topologique : on dira alors que l'on va de p à p_i par la direction d^+).
- on note L l'ensemble des directions du graphe.
- on notera L(p) l'ensemble des directions entrant ou sortant du sommet p. On appellera aussi topologie de p cet ensemble L(p).

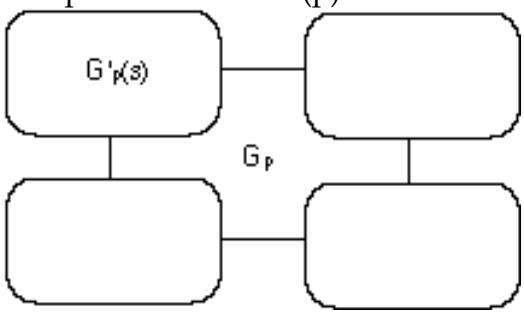


Figure III-21 : Graphe des processus lié à la partition écran de l'exemple de base

III.1.b) Association de ces trois entités

Nous avons donc caractérisé le lancer de rayon par une approche en termes de graphes :

1) Les données concernant les objets de la scène sont décrites par un ou plusieurs G_r.

2) Les données concernant les calculs d'optique géométrique (y compris la visualisation des pixels) sont décrites par un ou plusieurs G_v.

3) Les autres données et la dynamique des calculs sont décrites par G_p.

Le résultat final (l'image) est issu de l'association de ces trois graphes (pour l'exécution algorithmique, on considère également un graphe des machines qui est décrit plus bas). Pour exposer clairement cette association, nous avons besoin de définir un certain nombre de relations.

III.1.b.1) (processus, objet réel)

Cette relation booléenne sera définie comme vraie si le processus considéré possède dans ses données l'objet dont il est question. Le cas symétrique n'a à ce jour pas de sens.

Cette relation résulte de l'association de G_r et de G_p. Dans le cas d'une parallélisation définie à partir des objets de la scène par exemple, cette association est naturellement construite au début de l'algorithme.

Sur l'exemple de la Figure III-22, un sommet d'un graphe est associé à un sommet de l'autre. Dans le cas général, il peut y avoir création de doublons ou un sommet d'un graphe peut n'être associé à aucun sommet de l'autre graphe (ce qui peut caractériser une inadéquation entre les structures des graphes).

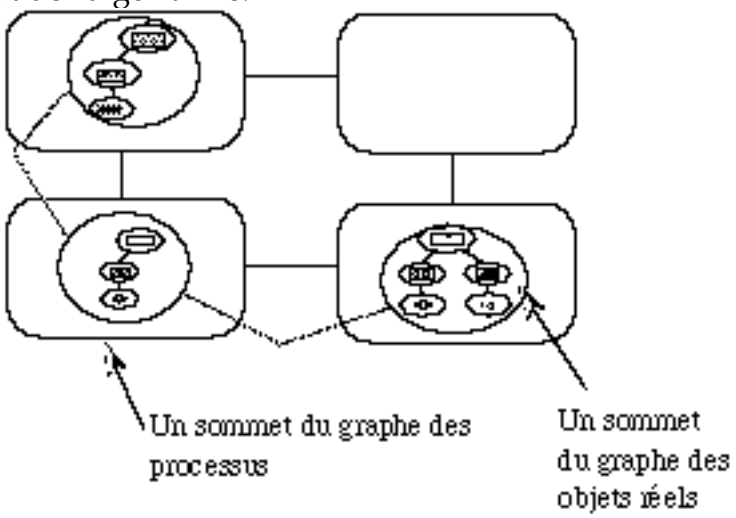


Figure III-22

III.1.b.2) (processus, objet virtuel)

Cette relation booléenne sera définie comme vraie si le processus considéré possède dans ses données l'objet dont il est question. Le cas symétrique n'a à ce jour pas de sens.

Cette relation résulte de l'association de G_v et de G_p.

Sur l'exemple de la Figure III-23, on remarque que l'association entre le graphe de la solution 2 de G_v et G_p est naturelle. Cepen-dant, en ce qui concerne les informations sur le cube directionnel, il y avait un choix à faire : ici, chaque processus prend en charge le surcoût de calcul lié au cube directionnel. Cependant, on aurait pu envisager de créer un processus spécial pour traiter cette partie de G_v. Des messages échangeant des rayons auraient alors été nécessaires.

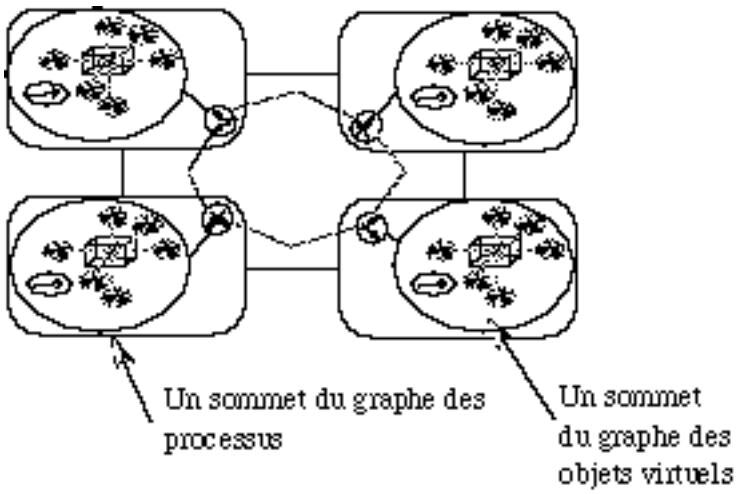


Figure III-23

L'association d'un sommet d'un graphe avec un sommet de l'autre met en évidence l'adéquation entre les différentes méthodes utilisées pour structurer les graphes (et donc optimiser le programme).

III.1.b.3) (objet virtuel, objet réel)

Cette relation booléenne définit l'association d'un objet virtuel (i.e. un rayon) avec un objet réel (par exemple une facette). Le but du lancer de rayon est de trouver pour un rayon donné, la bonne association avec l'objet réel (le plus proche du point de départ) puis d'effectuer des calculs photométriques et géométriques. Ce calcul sera noté dans la suite inter(objet virtuel, objet réel) et sera défini comme vrai si cette intersection a une solution. Le symétrique n'a pas de sens en lancer de rayon.

Cette relation résulte de l'association des objets réels et virtuels de G_v et de G_r. Dans notre exemple de base, si l'on applique les solutions des Figure III-17 pour les objets réels et III-20 pour les objets virtuels, un rayon d'un sommet de G_v sera testé avec les objets d'un sommet de G_r en utilisant les optimisations liées à la hiérarchie de boîtes englobantes (décrite par G_r). La structure des graphes guide cette association et par conséquent optimise les calculs. Chaque processus réalise ce calcul d'intersection.

L'intérêt de structurer les données en graphe est de diminuer la taille du problème en recherchant préalablement des relations entre ces données. Si G_v et G_r sont détaillés et que leurs sommets peuvent être associés de façon adéquate, alors l'intersection n'a plus qu'à être réalisée sur un ensemble réduit d'objets avec des informations préalables données par les graphes.

III.1.c) Relations internes aux trois entités

III.1.c.1) (processus, processus)

Cette relation indique qu'il y a une communication entre les deux processus. On symbolisera le fait qu'un message est échangé par la notation (processus,processus)[message]. Ces communications sont directionnelles (du premier processus au second).

Ces communications sont de différentes natures. Soient P_1 un processus, et P_x un voisin de P_1. On notera (P_1, P_x)[message] une communication de P_1 vers P_x et (P_x, P_1)[message] une communication de P_x vers P_1.

Il est possible de caractériser quatre types de communications en général :

1) (P_1, P_x)[objet réel] : communication P_1 vers P_x dans le cas des bases de données distribuées par exemple. Ces communications servent à construire ou modifier G_r.

2) (P_1, P_x)[objet virtuel] : communication utilisée par exemple dans le cas des algorithmes utilisant une division spatiale. Elles servent à construire ou modifier G_v.

3) (P_1, P_x)[inter(rayon, objet réel)] : communication dans le cas d'un processus ayant localement toutes les données nécessaires à un calcul et demandant à un autre processus spécialisé de réaliser ce calcul (on peut imaginer un processus spécialisé dans le calcul des intersections des sphères par exemple). Cela peut aussi caractériser un rééquilibrage dynamique de charge (c'est alors une façon de modifier G_p).

4) (P_1, P_x)[information] : communication exprimant des cas plus divers comme la terminaison d'un calcul, la synchronisation ou, dans notre cas, l'échange d'information de nature topologique (Cf. chapitre suivant).

Dans un cas plus général, les informations peuvent être détenues et envoyées par un processus qui n'est pas un voisin. Dans ce cas, il faut router l'information de P_x vers P_1 en passant par d'autres processus. Ceci sera noté :

routage(P_x,..., P_1)[message]

III.1.c.2) (objet virtuel, objet virtuel)

Cette relation symbolise les relations qu'il peut y avoir entre deux rayons (par exemple cohérence, notion de rayon généralisé, relation père/fils des rayons réfléchis par exemple - génère_fils(père,fils)).

III.1.c.3) (objet réel, objet réel)

Cette relation n'est rien d'autre qu'une récriture des arbres CSG par exemple (union(A,B), différence(C,D), etc…). Elle peut également exprimer des propriétés topologiques de G_r.

III.1.d) Récapitulation des relations entre les graphes

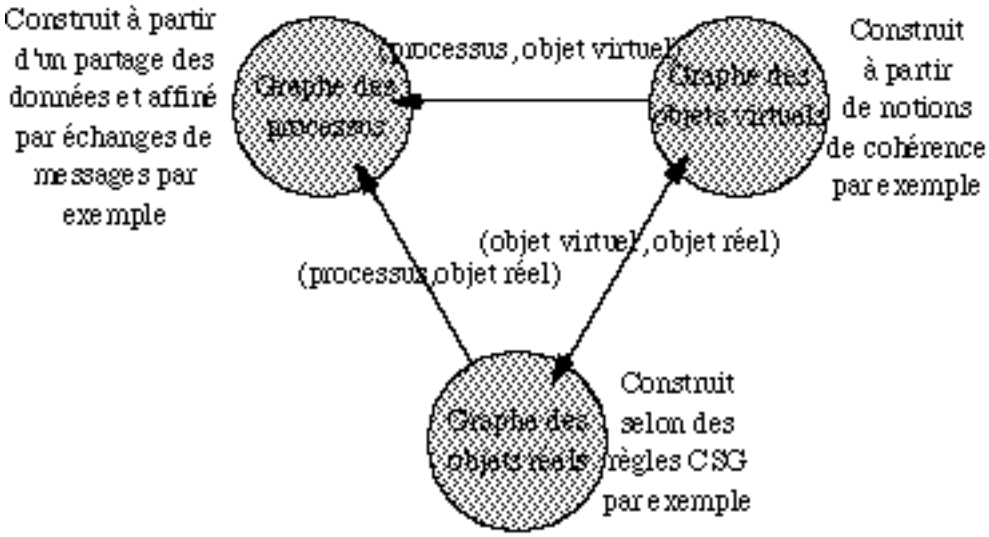


Figure III-24 : relation entre les trois graphes

III.1.e) Notion de "bon graphe"

On pourra dire que G_r est un bon graphe s'il exprime des propriétés (topologiques voire géométriques) importantes et exploitables (par exemple, un pavage carré et régulier de cubes peut être totalement défini par la connaissance de la géométrie d'un cube et d'un graphe qui sera un maillage carré ; ici G_r exprime complètement la structure de la scène modélisée).

Le "meilleur" graphe G_v, est plus facile à définir : c'est l'arbre des rayons qui donnent effectivement lieu à une intersection utile (calcul qui renvoie une information photométrique - on reviendra sur cette notion d'utile au chapitre consacré aux tests). Une telle approche avec comme point de départ le meilleur G_v a été tentée par Humbert & al. [HG94] mais est handicapée par des problèmes de mémoire. Le but était d'optimiser des calculs d'animation mais on peut imaginer des applications plus restreintes comme le recalcul d'une nouvelle scène en ne changeant que les paramètres photométriques ambiant, de diffusion lambertienne et spéculaire. La géométrie ne changeant pas, il est alors très efficace de réutiliser le graphe des objets réels et virtuels déterminé lors d'un précédent calcul de l'image comme l'ont montré Sequin et al. [SS90].

Il peut également être envisagé de tenir compte d'informations géométriques comme la zone d'espace traversée, etc…

Enfin, un bon graphe G_p doit avoir des propriétés classiques en parallélisme (extensibilité, granularité adaptée, routage simple, etc…).

III.1.f) Autres exemples

III.1.f.1) Exemple 1 : lancer de rayon de base

Dans le cas du lancer de rayon le plus simple par exemple, le graphe des objets réels est réduit à un ensemble totalement déconnecté d'éléments (ensemble non ordonné d'éléments). Pour le graphe des objets virtuels, on sait seulement qu'il aura une structure aborescente qui est inconnue au départ et dont on ne tire aucune information utile pour l'optimisation (les rayons sont utilisés sans aucun lien entre eux, les rayons primaires sont lancés indépendamment et les autres rayons sont traités au fur et à mesure de leur génération puis perdus). Enfin, il n'y a qu'un processus. Calculer une image revient à faire, dans ce cas, une recherche combinatoire de la solution.

Sur la Figure III-25 on constate que rien a priori ne permet d'exploiter des informations : les différents graphes sont fortement déconnectés et on ne dispose que d'un processus (en parallélisme on disposerait de n processus déconnectés sur lesquels on duplique G_r et on distribue G_v). La seule analyse de la structure des différents éléments permet de conclure que cette approche est grossière. Si l'on ajoute des boîtes englobantes, on retrouve l'algorithme proposé par Whitted [Wh80].

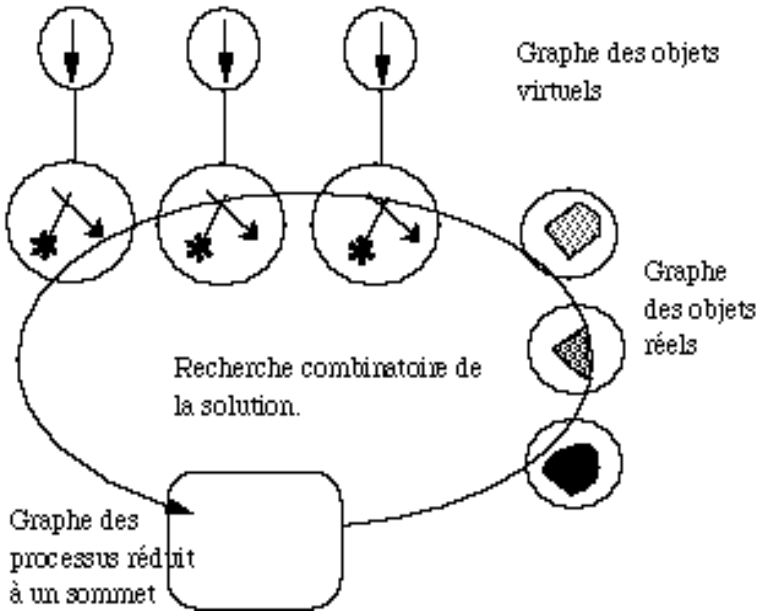


Figure III-25

III.1.f.2) Exemple 2 : cas d'une base de données répartie associée à une division de l'écran par regroupement de rayons primaires

Hypothèses : G_p fixé, G_r estimé (par une méthode à décrire) lors de la distribution de la base de données, G_v estimé (on a les rayons primaires donc on dispose de la structure de l'arbre au premier niveau) et associé à G_p (chaque processus a en charge un certain nombre de rayons primaires et calculera tout l'arbre de calcul). On considère un processus P_1 et P_x un voisin de P_1 ou un processus

accessible par routage. Si on ne dispose d'aucune optimisation permettant de réduire le nombre d'objets testés par rayon, on se retrouve dans un cas particulier de l'exemple précédent où tous les objets sont testés contre tous les rayons. Mais si la base de données est distribuée en fonction de la projection des objets sur l'écran, le traitement des rayons primaires est amélioré. Enfin, si l'on introduit une fonction d'évaluation estime_intersection(rayon, objet réel) qui est fausse lorsque l'on est sûr qu'il n'y a pas d'intersection est vraie sinon, on a un algorithme du type :

```
Pour tout rayon tel que (P1, rayon) Faire
            Pour tout objet réel tel que
               estime_intersection(rayon, objet réel) Faire
                  Si non (P1, objet réel)
                  Alors Trouver un PX tel que (PX, objet réel)
                        routage(PX,..., P1)[objet réel]
                  Fsi /* on modifie Gr */
                  Si inter(rayon, objet réel) /* association Gv Gr */
                  Alors garder la solution si c'est la meilleure
                  Fsi
            Fait
            génère_fils(rayon,rayon secondaire) /* on construit Gv */
            lancer récursivement le programme (rayon secondaire)
            calcul couleur
Fait
```

Ici, on voit (Cf. Figure III-26) que le graphe des processus G_p est au départ complètement connu, que le graphe des objets réels a été estimé (par un pré-traitement par exemple, ce qui peut générer des doublons) et associé à G_p, puis il est effectivement modifié par échange de messages en cours de calculs comme l'ont proposé Badouel & al. [BBP90]. Le graphe des objets virtuels (évalué au départ selon l'appartenance des rayons primaires à une zone écran) est construit au fur et à mesure après avoir rejeté les rayons primaires se perdant dans le vide et calculé les rayons secondaires ou d'ombre.
L'analyse des graphes peut qualifier la répartition des sommets et amener à rechercher des optimisations selon des règles de cohérence

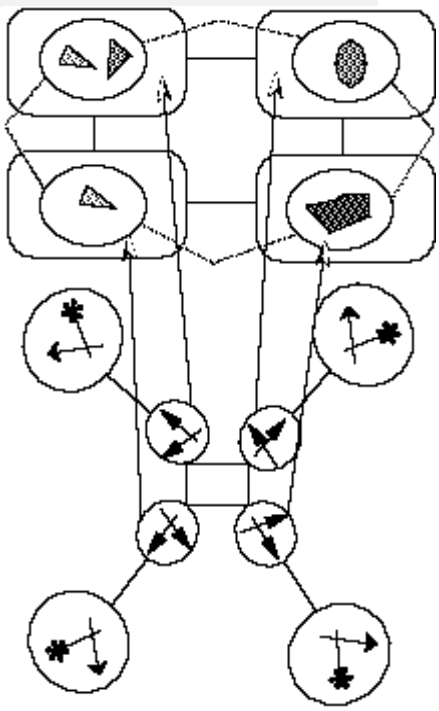


Figure III-26

sur les rayons primaires. Cependant on constate qu'un site a besoin de tester plus d'objets réels qu'il n'en possède. Même avec des optimisations permettant de ne pas tester tous les objets réels, il n'est pas possible d'être assuré de ne pas avoir de défaut de page et donc de prévoir une stratégie optimale sans ces défauts de page. Des informations intéressantes peuvent encore être extraites si l'on sait dire à quel moment les graphes terminaux sont construits (on sait alors si les échanges de données sont pénalisants ou pas, si le cache mémoire servant aux échanges d'objets réels est d'une dimension suffisante en fonction du nombre total d'objets réels dans la scène).

III.1.f.3) Exemple 3 : cas de la division spatiale volumique (voxels)

Hypothèses : G_p connu, G_r surestimé lors de la distribution de la base de données par voxel et donc associé à G_p, G_v inconnu.

Algorithme type :

```
Tant qu'il reste des calculs à faire par un processus Faire
            (Px, P1)[rayon entrant]            /* construction de Gv*/
            Pour tout objet réel tel que (P1, objet réel) Faire
                  Si inter(rayon entrant, objet réel) /*associe Gv Gr*/
                  Alors garder la solution si c'est la meilleure
                  Fsi
            Fait
            génère_fils(rayon entrant, rayons sortants)
            Tant qu'il y a des rayons sortants Faire
            /* avant de ressortir du voxel, on regarde si ces rayons */
            /* ne génèrent pas d'intersections dans celui-ci */
                  Pour tout objet réel tel que (P1, objet réel) Faire
                   Si inter(rayon sortant, objet réel) /*associe GvGr*/
                   Alors garder la solution si c'est la meilleure
                   Fsi
                  Fait
                  Si on n'a pas trouvé de solution pour ce rayon
                  Alors (P1, PY)[rayon sortant]
                  Sinon génère_fils(rayon sortant, rayons sortants)
                  Fsi
            Fait
Fait
```

Ce type de programme décrit par Cleary & al. [CWBV86] se caractérise par un graphe des objets virtuels très peu connu au lancement de l'algorithme. On remarquera également l'existence d'un test pour détecter la terminaison (qui n'est pas un problème simple en parallélisme) et un plus grand nombre potentiel de communications par rapport à l'exemple précédent.

III.1.f.4) Exemple 4 : méthodes vectorielles

On peut avec cette classification, caractériser des méthodes vectorielles comme celle de Plunkett & Bailey[PB85].

Les différents graphes se présentent schématiquement comme :

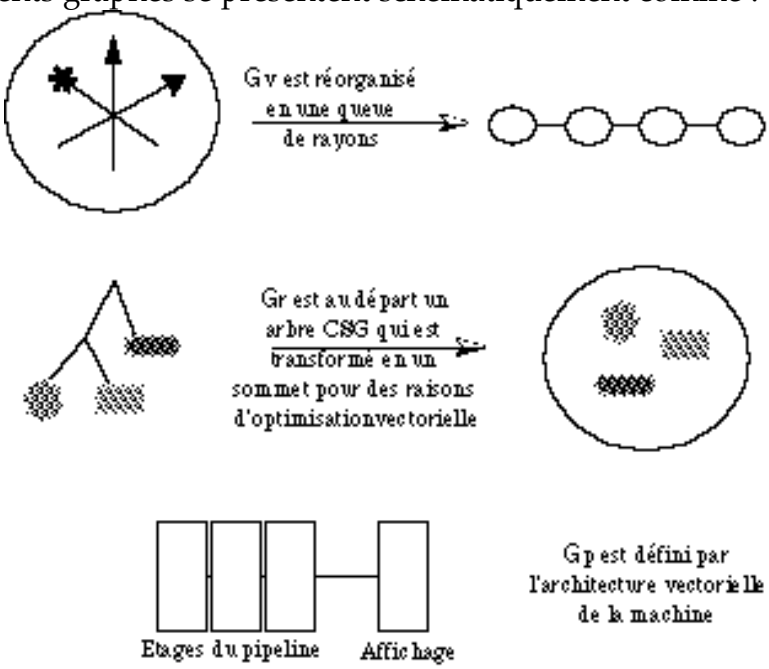


Figure III-27

Algorithme type :

```
Envoyer les rayons primaires dans la queue du pipeline
Pour tous les objets Faire
            Pour tous les rayons dans la queue du pipeline Faire
                  Si inter(rayon, objet) Alors
                  Alors garder la solution si c'est la meilleure
                        Envoyer les rayons secondaires dans la queue
                  Fsi
            Fait
Fait
```

G_r est au départ un arbre CSG, mais pour des raisons d'optimisation vectorielle (désamorçage du pipeline), les optimisations liées à la structure CSG ne sont pas considérées et les objets sont traités les uns après les autres.

G_v est réduit à une file de sommets dont l'ordre est quelconque (les rayons forment un ensemble linéairement structuré mais non ordonné).

On peut donc avec cette classification expliquer directement les difficultés rencontrées par cette approche vectorielle puisque les calculs se font à la chaîne sans possibilité d'optimisation autre que le pipelinage des calculs sur un ensemble considéré comme non structuré d'objets réels.

III.2) Nouvelle classification et classifications classiques

Si l'on retrouve ces trois graphes dans tous les programmes de lancer de rayon, leur niveau de complexité est cependant très différent. Plus il y aura d'information au niveau d'un graphe et plus les possibilités d'optimisation seront grandes. Par conséquent, un programme se caractérise par l'ordre dans lequel il considère les graphes, le niveau d'information initial de ceux-ci et des méthodes mises en œuvre pour affiner ces informations :

- a-t-on structuré les graphes les uns en fonction des autres ou de façon totalement disjointe (i.e. les optimisations du programme formeront-elles un ensemble homogène) ?

- adéquation entre les différents graphes : peut-on à partir de deux des trois graphes, en reconstruire un troisième qui exprime de nouvelles relations ou tous les sommets d'un graphe devront-ils être testés avec tous les sommets d'un autre. -
taille et informations sur le contenu des sommets de G_v ou G_r : les sommets sont-ils des petits ou des grands sous-ensembles de E_v ou E_r?

- pourra-t-on modifier ultérieurement l'un de ces trois graphes ?

- en cours de calcul, quels sont les mécanismes mis en œuvre pour ajouter de l'information aux différents graphes ?

Le résultat final peut également être caractérisé par les trois graphes qui auront, ou pas, été transformés et améliorés. Il est même envisageable de pouvoir quantifier la qualité des graphes de départ si l'on est capable d'établir des relations entre la structure des graphes (nombre de nœuds, d'arêtes, de sous-graphes connexes, etc…) et des critères d'efficacité (granularité, distribution, cohérence, équilibre…).

En conclusion : construire une méthode d'optimisation, revient à commencer par choisir sur quel graphe on commence par réfléchir parmi G_p, G_r, G_v, à le créer puis à le structurer pour exprimer des relations entre les objets qui lui sont propres, puis à essayer de construire les autres graphes à partir de ce premier graphe (Cf. Figure III-28). La construction peut se faire en cascade, graphe après graphe.

Les paragraphes suivants sont à lire en même temps que la Figure III-28.

III.2.a) Méthodes traitant d'abord les objets réels

Certaines scènes sont naturellement structurées. L'algorithme de Roth [Ro82] travaille sur une base de données qui est un arbre CSG. D'autres s'appliquent à traiter des surfaces particulières (Bezier pour [BS93], terrain modélisé [CS93], etc...). Travailler sur G_r consiste à extraire des informations sur les objets réels qui serviront à optimiser les calculs.

III.2.a.1) Parallélisation du traitement de ces scènes structurées

Les méthodes que l'on vient de décrire peuvent être parallélisées. Cette parallélisation peut se résumer à l'application de n fois le même algorithme mais l'on peut aussi chercher à trouver une structure du graphe des processus qui tire le meilleur parti du travail fait sur G_r, comme par exemple Wyvill et al. [WKS86] travaillant sur une division spatiale adaptée au CSG, ou Dippé et al. [DS84] réalisant une division adaptée à la scène. Beaucoup d'autres méthodes ont été évaluées [APB87], [KE82], [CDP95].

III.2.a.2) Traitement particulier des rayons sur des scènes structurées

On a vu que les méthodes directionnelles par exemple, liaient étroitement G_v et G_r par le biais du cube directionnel. En effet, pour utiliser ce cube qui optimise les intersections d'ombre (Cf. par exemple [HG86]), comme nous l'avons montré par le traitement sur notre scène de base, il faut d'abord travailler sur G_r puis sur G_v et ensuite les associer.

Le lancer de rayon discret (traité par exemple par [BSS93] ou [SC95]) procède selon une approche très différente. Dans un premier temps, on retraite les objets pour leur donner une représentation en voxels. Dans un second temps, on cherche

une représentation des rayons sur l'espace discret des objets. Le problème du lancer de rayon revient dans ce cas à effectuer un "et" logique entre la représentation d'un rayon et de la scène voxelisée.

III.2.a.3) Structuration successives de ces trois graphes

On peut envisager l'existence d'un algorithme qui avant de commencer les calculs du lancer de rayon proprement dit, aurait réussi à structurer les trois graphes. Ce pourrait être le cas par exemple, d'un lancer de rayon travaillant sur une scène CSG, avec des rayons généralisés et qui aurait été parallélisé sur une machine donnée. Cependant, nous n'avons pas trouvé de référence de travaux de ce type. Peut-être de telles méthodes ont-elles été implantées mais qu'elles n'ont pas été suffisamment originales pour être publiées (il s'agit d'un mélange d'approches classiques).

III.2.b) Méthodes traitant d'abord les objets virtuels

Au moins deux approches très différentes traitent principalement les rayons :

- les rayons généralisés comme le lancer de cône [Am84], de faisceau [DK85] ou [HH84], et le tracé de crayon [STN87], …

- les méthodes statistiques qui génèrent aléatoirement un certain nombre de rayons supplémentaires, selon une méthode de Monté-Carlo [BDMV88], des échantillonnage stochastiques [Co86], une distribution des rayons [CPC84] ou [LRU85], …

III.2.b.1) Parallélisation

Des applications parallèles de ces méthodes (déduction de G_p) ont problement été implantées mais nous n'avons pas trouvé d'articles les décrivant.

III.2.b.2) Traitement structuré

Pour ce qui est de l'association de G_v puis de G_r, certaines méthodes d'animation travaillent sur un arbre de rayons pré-calculé en cherchant des invariants temporels en considérant le déplacement des objets réels [MDC94] ou essaient de réutiliser des arbres de rayons déjà évalués [HG94].

Comme nous l'avons dit au III.2.a.2), il est a priori envisageable d'utiliser des rayons généralisés sur une scène structurée. Dans ce cas particulier, traiter G_v avant G_r, ou vice-versa, semble équivallent (mais dans le cas général, l'ordre de structuration est important car il induit des contraintes pour une bonne association des graphes). Cependant, étant donné que nous n'avons pas trouvé le détail de l'implantation d'une telle méthode, il est possible que l'ordre de traitement induise également des algorithmes différents (même faiblement).

III.2.c) Méthodes travaillant en priorité sur les processus

On peut considérer au moins deux types de travaux sur G_p:

- travail sur l'optimisation du code,
- parallélisation.

Si l'on ne considère que des aspects algorithmiques pour la parallélisation, on cherchera à avoir une méthode qui est algorithmiquement simple et efficace. Cette approche induit une voxelisation régulière de l'espace par exemple, car le traitement d'un maillage régulier est très simple [CW88] [FTI86] [CWBV86] [Pi91]

III.2.c.1) Association des rayons à la parallélisation

Les méthodes de vectorisation [Ma81] [PB85] tirent parti d'une architecture particulière en envoyant les rayons dans un pipeline.

Des méthodes utilisant des machines plus générales regroupent des rayons primaires puis affectent les paquets de rayons à un processeur libre d'une machine parallèle. Ici, l'architecture est fixée, mais l'affectation dynamique rayon/processus permet d'ajuster la charge de calcul [KH95].

III.2.c.2) Association des objets réels à la parallélisation

L'une des solutions implantées pour adapter G_r à un G_p donné est la distribution de la base de données comme l'ont fait Badouel et al. [BBP89]. On remarquera que traiter G_r puis G_p (Cf. III.2.a.1) donne un algorithme très différent du traitement de G_p puis G_r. Dans un cas, le graphe des processus "est adapté" aux irrégularités de densité d'objets de la scène alors que dans l'autre on répartit les objets sans tenir compte de cette même densité (cas de la voxelisation régulière). Cela est dû au fait que les graphes sont construits avec des contraintes initiales très différentes (simplicité algorithmique ou machine donnée pour G_p, répartition homogène pour G_r).

Des études ont également été menée pour associer un G_p donné lié à une machine parallèle et un graphe G_r très structuré comme par exemple une division en arbre octal (octree) [KNS87]. Dans ce cas, on pourrait considérer que l'ordre de structuration est assez indifférent, mais en fait c'est bien l'architecture de la machine qui est la première contrainte car il est plus facile d'envisager de changer de représentation pour la scène que de changer d'ordinateur parallèle. Cependant, il s'agit là d'une nuance.

III.2.d) Considérations générales sur la structuration des trois graphes

On remarquera que la grande majorité des méthodes structure fortement un graphe, parfois deux, mais que peu de méthodes arrivent à bien construire les trois

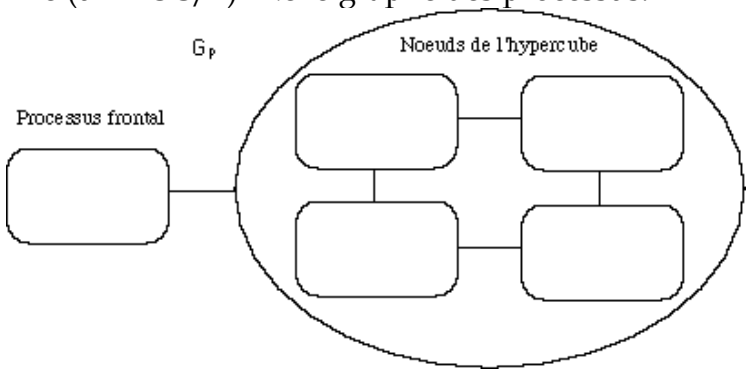
graphes avant de lancer les calculs. Pour les approches considérant en dernier le graphe des processus, on peut déduire une parallélisation (éventuellement par simple duplication du problème sur n sites). Pour les autres, étant donné qu'à chaque niveau on structure le problème, lorsque l'on arrive au dernier niveau il est très difficile de structurer encore le dernier graphe sans commencer le calcul (on n'a plus d'information a priori cohérente avec les contraintes déjà liées aux graphes déjà construits). La nécessité de respecter ces contraintes peut être illustrée par une voxelisation en arbre octal : dans ce cas G_p et G_r sont associés efficacement mais hélas le parcours d'un rayon à travers un arbre octal n'est pas trivial. Si en plus, on ajoute un lancer de faisceau par exemple, on imagine alors la grande difficulté qu'il y aurait à traiter un tel algorithme.

En fait une seule méthode autre que la parallélisation est issue du troisième niveau et est commune aux trois branches : un lancer de rayon qui travaillerait a priori sur des graphes exacts mais en partie non instanciés, comme [HG94]. Ces méthodes sont cependant très gourmandes en mémoire.

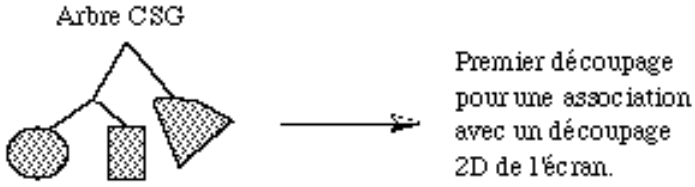
III.2.e) Un exemple structurant les trois graphes

Nous avons choisi de donner un dernier exemple développé par Priol et Bouatouch [PB89]. Cet exemple est intéressant car il travaille sur les trois graphes mais la structuration en une première phase construit les graphes sans lancer le lancer de rayon (dite phase statique) et une seconde qui a besoin d'un pré-calcul du lancer de rayon (dite phase dynamique).

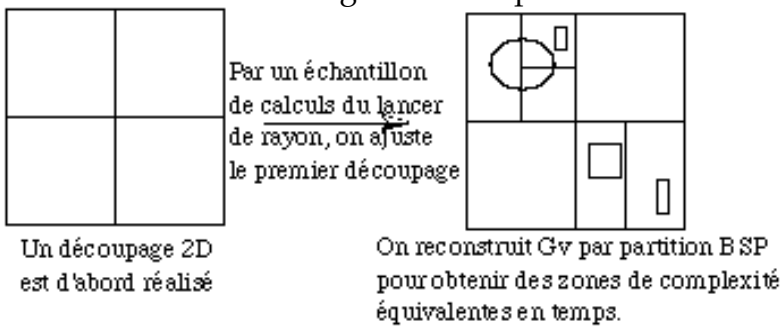
(1) La machine (un IPSC/2) fixe le graphe des processus.



(2) La scène est décrite initialement par un arbre CSG qui est redécoupé pour être associé à un découpage 2D de l'écran.



(3) Une division régulière de l'écran en zones 2D donne une première structuration de G_v. À partir de G_r défini en (2), on ajuste cette division par une partition BSP pour tenir compte de la densité d'objets par une phase dynamique de pré-traitement (on effectue un échantillonnage du calcul du lancer de rayon). Ce pré-calcul donne une estimation de la charge de calcul par zones écran.



(4) Enfin, on effectue un re-découpage de G_r en sous-arbres qui sont liées au G_v calculé en (3) (association aux zones écrans).

Les étapes (1) et (2) sont définies sans recourir au calcul du lancer de rayon alors que les deux dernières ont besoin d'effectuer une partie des calculs (ici sur les rayons primaires) pour obtenir plus d'information. On notera également que G_r est construit dans un premier temps puis réajusté après avoir obtenu des informations sur G_v.

III.2.f) Classification statique

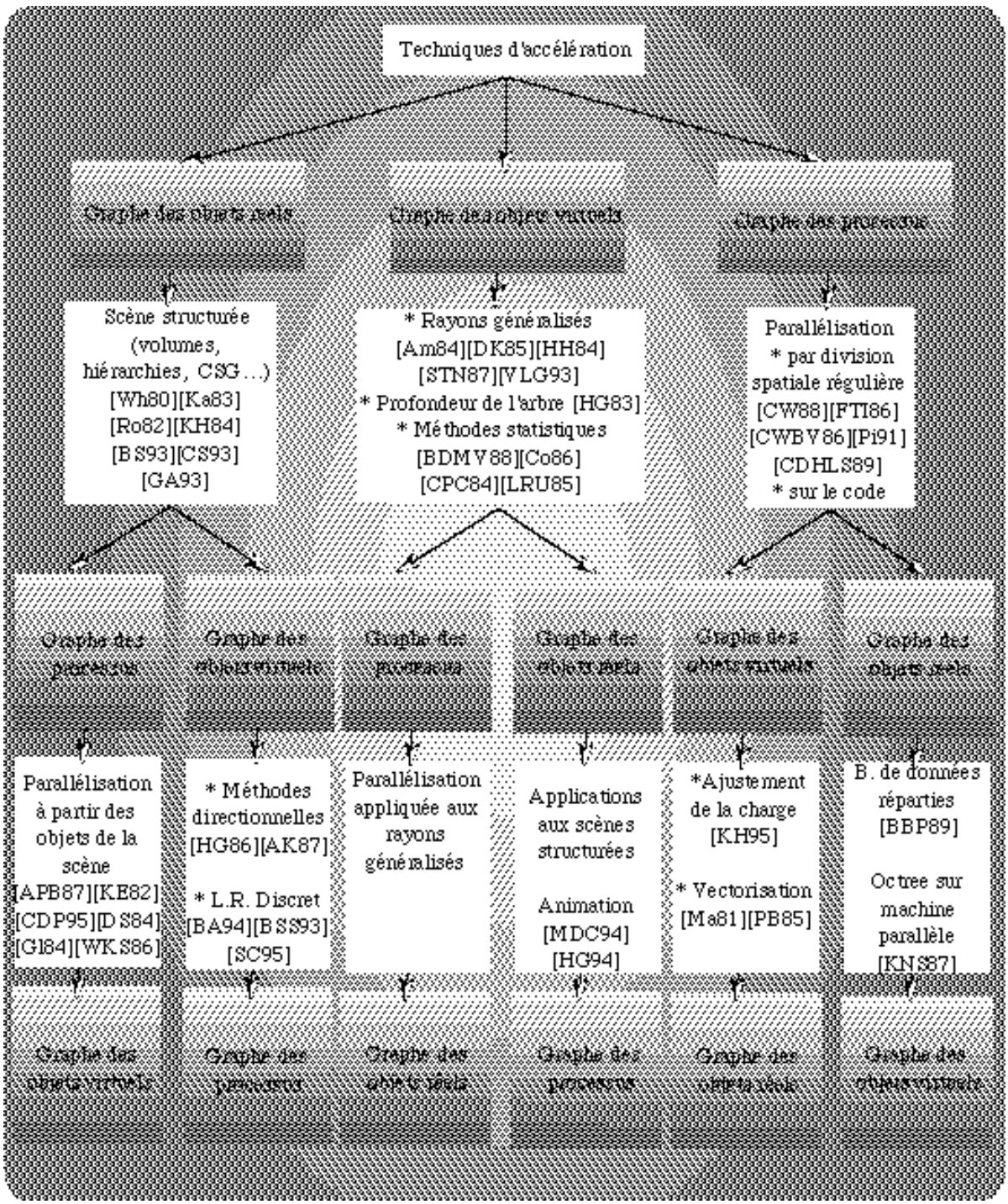


Figure III-28 : Nouvelle classification statique

Cette présentation de notre classification ne prend pas en compte l'évolution éventuelle des différents graphes au cours des calculs (d'où son nom de statique). Ce schéma a une approche un peu différente d'Arvo & Kirk, puisque nous cherchons à caractériser la construction d'une méthode, mais pas le champ d'application de celle-ci (tests d'intersection, rapidité). Les combinaisons de méthodes apparaissent

clairement et le parallélisme n'est plus traité comme un cas particulier. On voit également clairement que M. Lucas n'avait pas fait apparaître le traitement sur le graphe des objets réels comme un choix particulier (en fait, certains graphes sont souvent construits l'un en fonction de l'autre et apparaissent confondus, mais on choisit toujours l'un des deux en premier).

Si nous n'avions pas eu à introduire un grand nombre de définitions nouvelles pour le formalisme que nous venons de présenter, l'état de l'art de ce mémoire aurait été présenté avec celui-ci. Il nous a paru cependant nécessaire de faire d'abord une présentation classique. Les paragraphes suivant repassent rapidement en revue les méthodes déjà présentées au début de ce chapitre en les situant par rapport à notre formalisme.

III.2.g) Classification d'Arvo & Kirk

III.2.g.1) Généralisation de la notion de rayon

La généralisation de la notion de rayon travaille d'abord sur le graphe des objets virtuels. Chaque sommet de ce graphe est composé du rayon support du cône, du faisceau ou du pinceau implicitement lié à tous les rayons à l'intérieur de ce cône, faisceau ou pinceau. Ces méthodes ne fournissent pas d'approche particulière pour les graphes des objets réels ou le graphe des processus.

III.2.g.2) Avoir moins de rayons

L'approche consistant à générer moins de rayon travaille également en priorité sur le graphe des objets virtuels : le contrôle adaptatif de la profondeur est la seule méthode qui à notre connaissance tire une information d'optimisation de l'arborescence père/fils de l'arbre de calcul.

Les méthodes de *cohérence* peuvent, selon les cas, commencer par travailler sur ce graphe (les sommets regroupant les rayons "ayant un comportement similaire") mais également sur le graphe des objets réels. En fait, ces notions de cohérence sont trop vagues pour être immédiatement utilisables : nous donnerons dans ce mémoire trois séries de règles A_n, B_n et C_n qui rendent cette approche immédiatement exploitable et plus clairement classifiable.

Les méthodes statistiques travaillent sur l'association du graphe des objets réels et du graphe des objets virtuels en considérant par exemple la probabilité qu'un rayon touche un objet après avoir touché sa boîte englobante (le graphe des objets réels est bien structuré). Ceci nécessite d'avoir un graphe exprimant la relation entre cette boîte et son contenu (G_r). Dans le cas où l'on utilise le fait qu'un rayon a plus de

probabilité de toucher un objet si un rayon "proche" l'a aussi touché, on estime dynamiquement une association entre les deux graphes par une approche probabiliste se basant sur G_v. Ces méthodes ne fournissent pas d'approche particulière pour le graphe des processus.

III.2.g.3) Intersections plus rapides

Enfin, la branche des optimisations par intersections plus rapides regroupe un mélange assez hétérogène d'approches, ce que nous avions noté.

Toutes les méthodes travaillant sur les volumes englobants, cherchent à construire d'abord un graphe des objets réels générant d'importantes informations pour les optimisations.

Les méthodes de divisions spatiales travaillent en général à la fois sur le graphe des objets réels et sur le graphe des processus, en donnant la priorité à l'un ou l'autre selon les cas (exemple 2 du I.6.b par exemple).

Les techniques directionnelles cherchent à mieux structurer le graphe des objets virtuels. Souvent, cela est fait par une passe distincte du calcul (tampon de lumière par exemple). On associe une organisation pré-calculée du graphe des objets réels (projection Z-Buffer sur un cube directionnel) et la construction dynamique du graphe des objets virtuels lors de leur génération.

Enfin, la voie consistant à améliorer le code lui-même est un peu en marge de notre classification, car si on travaille dans ce cas sur le graphe des processus c'est pour améliorer la qualité d'une partie du code et non l'architecture du programme. Or nous ne mesurons pas cet aspect précis. En fait, nous sommes capables d'évaluer les qualités d'une approche, non sa mise en œuvre.

III.2.h) Classification des méthodes parallèles

Pour optimiser le lancer de rayon par des méthodes parallèles, deux grandes approches ont été distinguées [Lu92] :

Approche 1) Parallélisation par regroupement de rayons primaires (G_p fixé, G_r estimé lors de la distribution de la base de données, G_v est estimé car on ne connaît que les rayons primaires et il est associé à G_p). La scène projetée sur l'écran est découpée en zones 2D de pixels [CDHMS89]. A chaque zone, on associe un processus de calcul qui prend en charge la totalité de l'arbre de calcul pour chaque pixel de la zone. Deux stratégies sont envisageables pour l'accès à la base de données

(travail sur G_r) : (1) on la duplique autant de fois qu'il y a de processus, ou (2) on la distribue sur tous les processus et l'on introduit un protocole d'échange de données entre les processus pour les cas où l'un d'entre eux aurait besoin d'informations ne se trouvant pas sur la base qu'il possède [BP90].

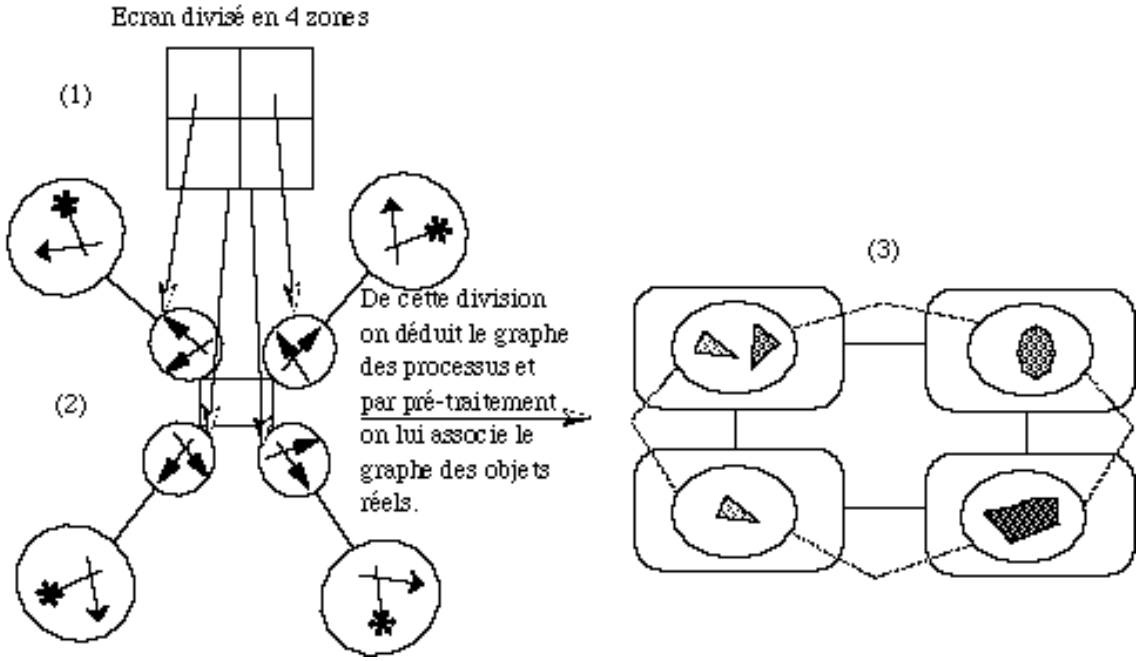


Figure III-29 : Parallélisation par regroupement de rayons primaires

Approche 2) Partage en voxels de l'espace 3D (G_p connu, G_r estimé - en rajoutant parfois des doublons - lors de la distribution de la base de données par voxel et donc associé à G_p, pour G_v, on sait seulement quels processus vont traiter quels rayons primaires). Chaque processus est associé à un voxel. La part de base de données détenue par chaque processus contient les objets englobés en totalité ou partie par son voxel. Durant les calculs, les communications servent essentiellement à échanger des rayons ((P_1,P_x)[objet virtuel]) entre voxels [DS84], [CWBV86] et donc à construire G_v. Pour une répartition de la charge de travail homogène, un pré-échantillonnage peut être effectué [BBP89]. Cette approche conduit à créer des voxels de différentes tailles. D'autres partitions peuvent être déduites des optimisations séquentielles utilisant les volumes englobants (hiérarchisés ou non) [Le92] ou une division récursive de l'espace [Gl84], [CW88], [AK89]. Ces méthodes font une évaluation de temps ou de densité d'objets (donc des différents graphes), et leur efficacité dépend grandement de celle-ci. L'intérêt de cette seconde approche est de pouvoir distribuer la base de données sur tous les processus. Ainsi, on réduit le nombre de calculs d'intersections et la mémoire nécessaire sur chaque site. Elle est cependant gourmande en nombre de messages (chaque rayon fait l'objet d'envoi de messages lors de sa transmission et pour le rapatriement des couleurs des

intersections locales) et crée parfois des problèmes de charge de communication, voire de terminaison [Pr89].

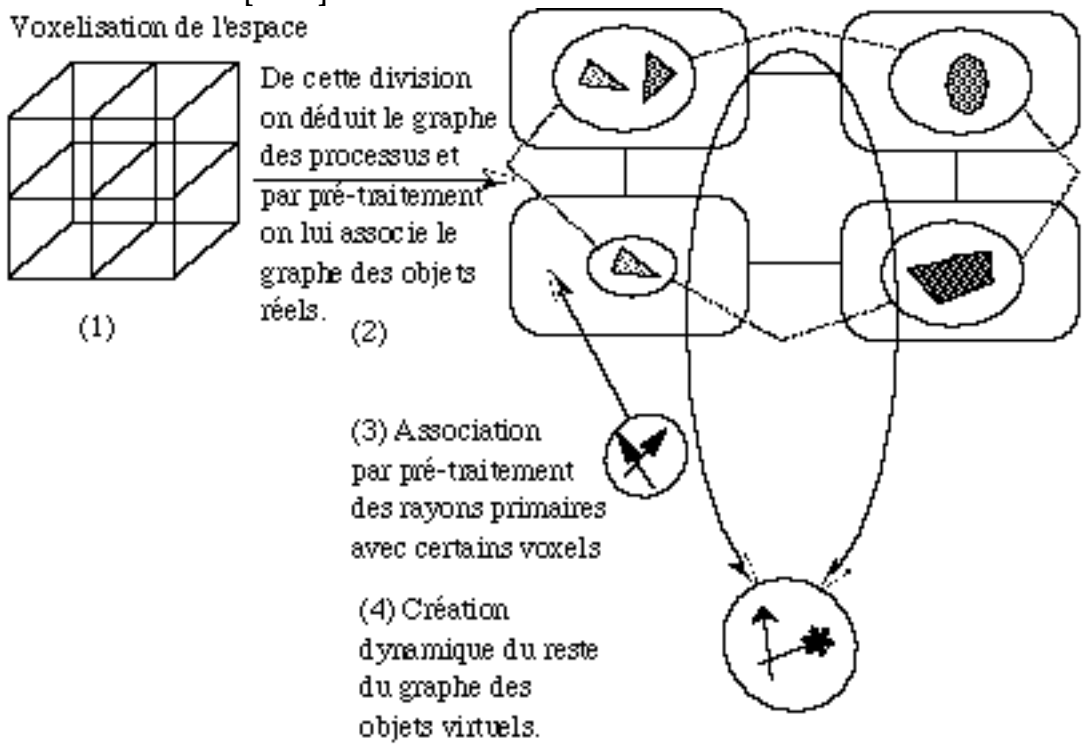


Figure III-30 : parallélisation par division spatiale

Enfin, notre classification permet de décrire clairement des méthodes hybrides comme celle de Kobayashi et al. [KNS87] qui travaillent d'un côté sur le graphe des processus en réalisant un mélange de parallélisme et de traitement pipeliné (Cf. Figure III-31) et de l'autre sur la création et la manipulation d'un arbre octal (G_r).

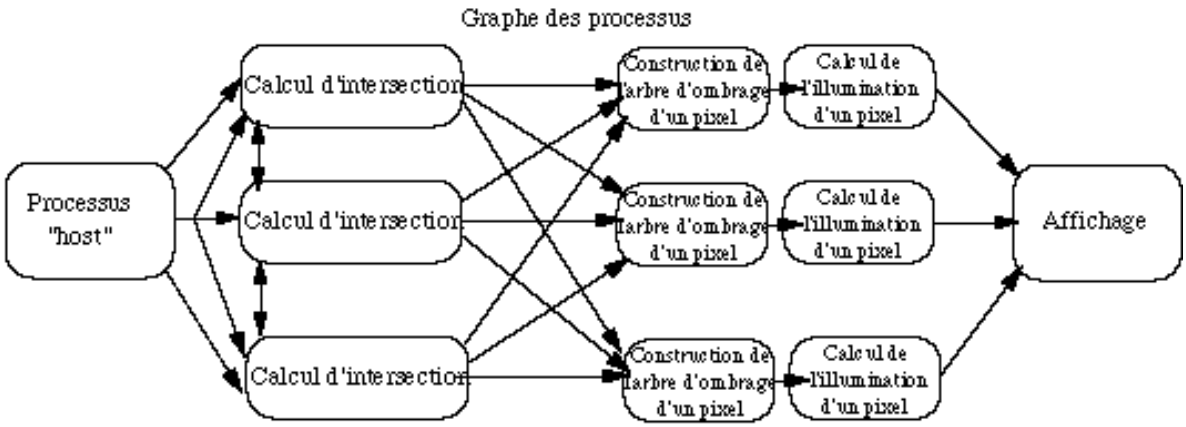


Figure III-31 : pipelines parallèles

Ici, faire le simple distinguo entre regroupement de rayons primaires et partition spatiale ne décrit pas complètement la richesse et la complexité de la méthode.

III.3) Conclusion sur cette nouvelle classification

La figure III-28 résumait dans quel ordre pouvaient être structurés les différents graphes et replaçait différentes méthodes dans cette classification statique. Cependant, ces graphes sont souvent améliorés au fur et à mesure du calcul de l'image. Pour être complet, il est donc souhaitable de considérer également l'aspect dynamique lié à ces graphes

Si l'on considère l'évolution des graphes entre le début des calculs de l'image et leur fin, l'algorithme "parfait" serait celui qui connaît avant de lancer le programme les graphes terminaux G_p, G_v, G_r (lorsque ces trois graphes existent pour une méthode donnée et qu'ils expriment effectivement une propriété d'optimilaté à la fin de l'algorithme) et qui n'a alors plus qu'à calculer inter(objet virtuel, objet réel) entre un seul rayon et un seul objet. Ce cas évident étant de fait difficilement réalisable, on dispose en général au mieux d'un graphe complètement connu (au moins G_p réduit à un sommet), d'une estimation des deux autres et cette connaissance va être complétée en cours de calculs en suivant, dans le meilleur des cas, une heuristique pour effectuer le moins de calculs possibles. Les méthodes d'optimisation se caractérisent donc aussi par la connaissance dont on dispose au départ, de la méthode et l'ordre employés pour affiner l'association de G_p, G_v, G_r.

Conclusion

Dans ce chapitre, le problème du lancer de rayon a été analysé et décomposé en terme de graphes.

La structure de ces graphes caractérise l'approche d'optimisation adoptée et ce formalisme permet de mieux classer les méthodes existantes mais aussi de montrer que certaines voies n'ont pas été totalement explorées (lancer de rayon formel, recalcul partiel de l'arbre des rayons, structuration complexe de tous les graphes…).

Ce formalisme est également important car il introduit la méthode d'optimisation exposée au chapitre suivant.

Bibliographie

[AK87] : ARVO (J.), KIRK (D.). - "Fast ray tracing by ray classification". - Computer Graphics, 21(4), juillet 1987, p. 55-64.

[AK89] : ARVO (J.), KIRK (D.). -"A Survey Of Ray Tracing Acceleration Techniques" p. 201-262 In An Introduction To Ray Tracing / Ed. A. S. Glassner. London : Academic Press Limited, 1989. - 327 p.

[Am84] : AMANATIDES (J.). - "Ray tracing with cones". - Computer Graphics, 18(3), 1984, p. 129-135.

[APB87] : ARNALDI (B.), PRIOL (T.), BOUATOUCH (K.). - "A new space subdivision method for ray tracing CSG modelled scenes." - The visual computer. - Springer-Verlag, 3, 1987, p. 98-108.

[BA94] : BERGAD (K.), ATAMENIA (A.). - "Approche discrète du lancer de rayon", Actes des journées AFIG-GROPLAN, Toulouse, 30 nov. - 2 dec. 1994, p.31-42.

[BBP89] : BADOUEL (D.), BODIN (F.), PRIOL (T.). "Lancer De Rayon : Approches Parallèles". Rennes IRISA, Publication Interne n°452, Jan. 1989, 15 p.

[BDMV88] : BOUVILLE (C.), DUBOIS (J.L.), MARSHALL (I.), VIAUD (M.L.). - "Monté-Carlo integration applied to illumination model". Actes d'Eurographics'88, 1988.

[BP90] : BADOUEL (D.), PRIOL (T.) - "Vm_pRay An Efficient Ray Tracing Algorithm On A Distributed Memory Parallel Computer". INRIA Rennes, Rapport De Recherche n°1198.

[BS93] : BARTH (W.), STURZLINGER (W.). - "Efficient ray tracing for Bezier and B-spline surfaces". Computers & Graphic, 17(4), juillet 93, p. 423-430

[BSS93] : BLASI (P.), Le SAEC (B.), SCHLICK (C.). - "A rendering algorithm for discrete volume density objets". Actes of Eurographics93, Sept. 1993, p. 201-210.

[BWJ84] : BRONSVOORT (W.F.), VAN WIJK (J.J.), JANSEN (F.W.). "Two methods for improving the efficiency of ray casting in solid modeling". - Computer Aided Design, 16, 1984, p. 51-55.

[CDHLS89] : CROW (F.C.), DEMOS (G.), HARDY (J.), McLAUGHLIN (J.), SIMS (K.). -"3D image synthesis on the connection machine" in "Parallel processing for computer vision and display" / sous la direction de DEW, EARNSHAW, HEYWODD : Addison Wesley, 1989.

[CDP95] : CAZALS (F.), DRETTAKIS (G.), PUECH (C.). "Filtering, clustering and hierarchy construction : a new solution for ray-tracing complex scenes", Actes d'Eurographics95, 14(3), 1995, p. 371-382.

[Co86] : COOK (R.L.). "Stochastic sampling in computer graphics". ACM Trans. Graph, 5(1), janvier 1986.

[CPC84] : COOK (R.L.), PORTER (T.), CARPENTER (L.). "Distributed ray tracing". - Computer Graphics (Siggraph'84 proceedings), 18(3), juin 1984, p. 137-145.

[CS93] : COHEN (D.), SHAKED (A.). - Photo-realistic imaging of digital terrains", Actes d'Eurographics93, Sept. 1993, p. 363-373

[CW88] : CLEARY (J.G.), WYVILL (M.). - "Analysis Of An Algorithm For Fast Ray Tracing Using Uniform Space Subdivision". The Visual Computer, 4(2), July 1988.

[CWBV86] : CLEARY (J.G.), WYVILL (B.), BIRTHWISTLE (G.M.), VATTI (R.) - "Multiprocessor ray tracing", Computer graphics forum, 5(1), march 1986, p. 3-12

[DK85] : DADOUN (N.), KIRKPATRICK (D.G.). "The geometry of beam tracing algorithm". - Proceedings Eurographics'85, juin 1985, p. 55-61.

[DS84] : DIPPE (M.), SWENSEN (J.). - "An adaptative subdivision algorithm and parallel architectrue for realistic image synthesis". Siggraph'84 conference proceedings, 18(3), 1984, p. 149-158.

[FDFH93] : FOLEY (J.), van DAM (A.), FEINER (S.), HUGHES (J.). - "Computer Graphics : principles and practice", Addison Wesley, 1993, 1175 p.

[FP81] : FUCHS (H.), POULTON (J.). "Pixel-planes : a VLSI oriented design for a raster graphics engine". - VLSI Design, 20(3), 1981.

[FTI86] : FUJIMOTO (A.), TANAKA (T.), IWATA (.), "ARTS : accelerated ray tracing system". - IEEE Computer Graphics Applications, 6(4), avril 1986, p. 16-26.

[GA93] : GARGANTINI (I.), ATKINSON (H.H.). - "Ray tracing an octree : numerical evaluation of the first intersection". Computer Graphics Forum, 12(4), Oct.n1993, p. 199-210.

[Gl84] : GLASSNER (A. S.). - "Space subdivision for fast ray tracing". - IEEE Computer Graphics Applications, 4(10), octobre 1984, p.15-22.

[Gl89] : An introduction to ray tracing / sous la direction de Andrew S. Glassner. - Londres : Academic press limited, 1989. 327 p.

[GP89] : GREEN (S.A), PADDON (D.J). - “Exploiting coherence for multiprocessor ray tracing”. IEEE computer graphics and application, 9(11), nov. 1989, p. 12-26.

[GS85] : GOLDSMITH (J.), SALMON (J.). - “A ray tracing system for the hypercube”. - Caltech Concurrent Computing Project Memorandum HM154, California Institude of Technology, 1985.

[He89] : HECKBERT (P. S.). - “Writing a ray tracer”. In An introduction to ray tracing, ed. A. S. Glassner. - Londres : Academic press limited, 1989, p. 263-293.

[HG86] : HAINES (E.A.), GREENBERG (D.P.) - “The light buffer : a shadow testing accelerator”. IEEE Comp. Graph. Appl., 7(5), Sept. 1986, p. 6-16.

[HG94] : HUMBERT (P.), GARDAN (Y.). - "Utilisation des arbres de rayons pour le rendu de séquence d'animation". Actes de Groplan 94, Toulouse, 1994, p. 261-272.

[HH84] : HECKBERT (P.S.), HANRAHAN (P.). - “Beam tracing polygonal objects”. - Computer Graphics, 18(3), juillet 1984, p. 119-127.

[Ka83] : KAJIYA (J.T.). - “New techniques for ray tracing procedurally defined objects” - Computer Graphics, 17(3), 1983, p. 91-102.

[Ka85] : KAPLAN (M.R.). - “Space tracing, a constant time ray tracer. State of the art in image synthesis”. Siggraph’85 course notes, 11, juillet 1985.

[Ka86] : KAJIYA (J.T.). - “The rendering equation”. - Computer Graphics, 20(4), Août 1986, p. 143-150.

[KE84] : KEDEM (G.), ELLIS (J.L.). "The ray casting machine". - Proceedings of ICCD'84, Ray NY, oct 1984, p.533-554.

[KH84] : KAJIYA (J.T.), VON HERZEN (B.P.). - “Ray tracing volume densities”. - Computer Graphics, 18(3), 1984, p. 129-134.

[KH95] : KEATES (M.), HUBBOLD (R.J.). - “Interactive ray tracing on a virtual shared-memory”. Computer graphics forum, 14(4), Oct. 1995, p. 189-202.

[KK86] : KAY (T.L.), KAJIYA (J.). - “Ray tracing complex scenes”. Computer Graphics, 20(4), Août 1986, p. 269-278.

[KNS87] : KABAYASHI (H.), NAKAMURA (T.), SHIGEI (Y.). - “Parallel processing of an object space image synthesis using ray-tracing”, The Visual Computer, 3(1), 1987, p. 12-22.

[LDC93] : MAUREL (H.), DUTHEN (Y.), CAUBET (R.). - “A 4D ray tracing”. Proceedings of Eurographics93, Sept. 1993, p. 285-294.

[Le92] : LEFER (W.). - "Parallélisation du lancer de rayon : une solution au problème de l'équilibre de la charge". Actes de GROPLAN 92, Nantes, 1992, p. 163-170.

[LOKSO83] : NISHIMURA (H.), OHNO (H.), KAWATA (T.), SHIRAKAWA (I.), OMURA (K.). - “LINKS-1 : a parallel pipelined multimicrocomputer system for image cration.” - Proceedings of th 10th symposium on computer architecture, SIGARCH, 1983, p. 387-394.

[LRU85] : LEE (M.), REDNER (R.A.), USELTON (S.P.). -”Statistically optimized sampling for distributed ray tracing.”. - Computer Graphics, 19(3), juillet 1985, p. 61-67.

[Lu92] : Lucas (M.). - “La Parallélisation De La Synthèse D'images Réalistes”. Revue Internationale De C.F.A.O. Et D'infographie, 7(1), 1992, p.41-50.

[Ma81] : MAX (N.L.). - “Vectorized procedural models for natural terrain : waves and islands in the sunset”. Computer graphics, 15(3), Août 1981,p. 317-324.

[MDC93] : MAUREL (H.), DUTHEN (Y.), CAUBET (R.) - “A 4D Ray Tracing”. Computer graphics forum, Actes d’Eurographics’93, Barcelone, septembre 1993, 12 (3), p. 285-294.

[OM87] : OHTA (M.), MAEKAWA (M.). - “Ray coherence theorem and constant time ray tracing algorithm”. Computer Grapics 1987 (Proc. Of CG International’87) , ed. T.L Kunni, 1987, p. 303-314.

[PB85] : PLUNKETT (D.J.), BAILEY (M.J.). - “The vectorization of a ray tracing algorithm for improved execution speed.”. - IEEE computer graphics application, 5(8), Août 1985, p. 52-60.

[Pi91] : PITOT (P.). - "Conception et réalisation d'une machine parallèle dédiée à la synthèse d'images réalistes : la machine VOXAR". Thèse de doctorat, Université Paul Sabatier, Toulouse, 1991, 195 p.

[Pr89] : PRIOL (T.). -"Lancer De Rayon Sur Des Architectures Parallèles : Étude Et Mise En Œuvre". Thèse Soutenue À L'université Rennes I, 1989, 136 p.

[PRIOL] :

* BADOUEL (D.), BOUATOUCH (K.), PRIOL (T.). -"Ray tracing on distributed memory parallel computers : stategies for distributing computations and data". - Rapport de recherche n°1163, INRIA RENNES, février 1990.

* [PB89] PRIOL (T.), BOUATOUCH (K.). -"Ray tracing on parallel computers : a performance study". - Publication interne n°451, IRISA RENNES, janvier 1989, 20 p.

[Pu86] : PURGATHOFER (W.). - "A statistical method for adaptive stochastic sampling", p. 145-152 in : Proceedings of Eurographic'86 / sous la direction de A.A.G. Requicha. - ELSEVIER (North Holland), 1986.

[Ro82] : ROTH (S.D.). - "Ray casting for modeling solids". - Computer graphics image process, 18, 1982, P. 109-144.

[RW80] : RUBIN (S.), WHITTED (T.). -"A three dimensional representation for fast rendering of complex scenes". - Computer graphics, 14(3), juillet 1980, p. 110-116.

[SC95] : STOLTE (N.), CAUBET (R.). - "Discrete Ray-tracing of huge voxel spaces". Actes d'Eurographics95, 14(3), 1995, p 383-394.

[SDB86] : SPEER (L.R.), DEROSE (T.D.), BARSKY (B.A.). - "A theoritical and empirical analysis of coherent ray tracing." . - Computer generated images (Proc. of graphics interface'85), 27-31 mai 1986, p. 11-25.

[Si89] : SILLION (François). - "Simulation de l'éclairage pour la synthèse d'images : réalisme et interactivité". - [1989]. - 172 f. dactyl. Thèse : spécialité informatique : PARIS-SUD centre d'ORSAY.

[SS90] : SEQUIN (C.H.), SMYRL (E.K.). - "Parametrized ray-tracing3. Computer graphics, vol 23, n°3, juillet 1990, p. 308-314.

[SSS74] : SUTHERLAND (I.E.), SPROULL (R.F.), SCHUMACKER (R.A.). -"A characterization of ten hidden surface algorithms". - Computer surv., 6(1), Mars 1974, p. 1-55.

[STN87] : SHINYA (M.), TAKAHASHI (T.), NAITO (S.). -"Principles and applications of pencil tracing". - Computer graphics, 21(4), juillet 1987, p. 45-54.

[VLG93] : ROBIN (V.), GARDAN (Y.), LANUEL (Y.). - "Optimisation du tracé de rayon : une approche originale et adaptative entre le ray-tracing et le beam-tracing". Actes de MICAD'93, Paris, 1993, p. 19-39.

[Wh80] : WHITTED (T.). -"An improved illumination model for shaded display." . - Communication of ACM, 23(6), juin 1980, p. 343-349.

[WKS86] : WYVILL (G.), KUNII (T.L.), SHIRAL (Y.). -"Space division for ray tracing in CSG". - IEEE Computer graphics applications, 6(4), avril 1986, p. 28-34.

[YMS83] : YAU (), MANN-MAY (), SRIHARI (S.N.). -"A hierarchical data structure for multi-dimensional digital images". - Communication of ACM, 26(7), juillet 1983, p. 504-515.