THESE

Présentée à

UNIVERSITE DE METZ

Pour l'obtention du grade de :
DOCTEUR de L'UNIVERSITE DE METZ

Spécialité : INFORMATIQUE

Yann LANUEL

LES ARBRES VSG: UNE NOUVELLE APPROCHE MULTIMODELES DE LA VISUALISATION

Soutenue à Metz le 10 Février 1994

Composition du jury :

Directeur de thèse :	Yvon GARDAN
Rapporteurs :	Marie-Christine HATON Bernard PEROCHE
Examinateurs :	Ileana COSTEA Claude LAURENT

LABORATOIRE DE RECHERCHE EN INFORMATIQUE DE METZ

THESE

Présentée à

UNIVERSITE DE METZ

Pour l'obtention du grade de :
DOCTEUR de L'UNIVERSITE DE METZ

Spécialité : INFORMATIQUE

Yann LANUEL

LES ARBRES VSG: UNE NOUVELLE APPROCHE MULTIMODELES DE LA VISUALISATION

Soutenue à Metz le 10 Février 1994

Composition du jury :

Directeur de thèse : Yvon GARDAN

Rapporteurs : Marie-Christine HATON
Bernard PEROCHE

Examinateurs : Ileana COSTEA
Claude LAURENT

LABORATOIRE DE RECHERCHE EN INFORMATIQUE DE METZ

A Cécile
A Isabelle

Remerciements

La page de remerciements qui débute tout mémoire est en fait celle que l'on écrit en dernier. En retraçant le parcours qui m'amène maintenant à la rédiger, je m'aperçois du rôle essentiel qu'ont joués les personnes de mon entourage familial et professionnel.

C'est donc le plus chaleureusement et le plus sincèrement du monde que je tiens à exprimer ma gratitude au personnes suivantes:

A Yvon GARDAN, merci pour tout.

A Marie-Christine HATON, professeur à l'université de NANCY I, et Bernard PEROCHE, professeur à l'Ecole des Mines de SAINT-ETIENNE, d'avoir consacrer de leur temps précieux pour lire, corriger, et évaluer cette thèse.

A Ileana COSTEA, professeur à l'université de LOS ANGELES, et Claude LAURENT, professeur à l'université de METZ, d'avoir aimablement accepter d'être examinateurs.

A Jean-Claude PAUL, directeur de Centre de Recherche en Architecture et Ingénierie de Nancy et Pascal DEVILLE, pour nous avoir gracieusement fourni les bases de données qui ont nous permis d'améliorer la qualité des images.

J'aimerai également remercier tous les membres du L.R.I.M. qui font régner une ambiance extrêmement sympatique. A Catherine, Dominique, Jocelyne et Martine d'être toujours souriantes et serviables. A Jean-Pierre, pour (entre autres) ses talents de photographe. Je remercie plus particulièrement Robin VIVIAN qui a pris sa revanche d'une bien belle manière et avec qui j'ai plaisir à travailler.

A Isabelle, merci pour tout. Il y a d'autres mots mais qui ne sont que pour toi seule.

Sommaire

Chapitre 2 : Les arbres VSG : un modèle de visualisation.

Chapitre 3 : Optimisation par classement a priori des objets.

Chapitre 4 : Etude des performances.

Introduction.

La visualisation occupe une place importante dans les systèmes de CFAO (Conception et Fabrication Assitée par Ordinateur). C'est souvent le meilleur moyen d'appréhender les résultats des traitements réalisés par le système. La visualisation est en particulier cruciale dans la phase de modélisation. Lorsqu'un opérateur décrit la forme d'un objet, il doit pouvoir "voir" si les ordres qu'il donne au système créent effectivement la forme souhaitée. La visualisation est alors intégrée au dialogue et doit donc répondre aux contraintes d'interactivité imposées par ce contexte. En théorie, pour qu'un système interactif puisse être considéré comme convivial, les temps de réponse doivent être inférieurs à une seconde, bien que deux ou trois secondes soient parfois admissibles. Le problème de la visualisation est donc de produire, en un temps très court, une image de l'objet qui doit être suffisamment proche de la réalité pour faciliter son interprétation par l'opérateur. Dans le cas de l'animation, le problème est encore plus important puisqu'il s'agit de produire 25 images par seconde. A l'inverse, pour le design par exemple, le réalisme est l'aspect prioritaire de la visualisation. Tous les efforts sont consacrés à la simulation des effets lumineux tels que les réflexions, les transparences, les ombres portées, etc. Le temps de calcul d'une image peut alors devenir très important, de l'ordre de plusieurs heures. Les deux caractéristiques de la visualisation sont donc performance et réalisme. Le choix est souvent arbitré par d'autres considérations (le modèle géométrique, le matériel, etc.).

Lorsque les objets sont modélisés par des facettes planes, la puissance de calcul des machines actuelles permet d'obtenir une visualisation peu réaliste, de type fil de fer, dont les temps de calcul répondent aux critères d'interactivité imposés par le dialogue ou l'animation. Mais de plus en plus, ces machines sont équipées d'unités graphiques spécialisées où un algorithme d'élimination des parties cachées et un algorithme de lissage des faces sont implantés de manière physique (câblés). Là encore, les temps de calculs sont compatibles avec l'interactivité, en ayant un réalisme certes de bas niveau (modèle d'éclairement simpliste, pas d'ombres portées, ni de réflexions, ni de transparences), mais souvent suffisant pour la conception des objets. Ce type de dispositif à l'inconvénient d'être très rigide c'est-à-dire qu'il limite son utilisation à une représentation donnée et un niveau de réalisme donné. Par exemple, les objets représentés par un arbre CSG (Constructive Solid Geometry ou arbre de Construction) ne peuvent profiter directement de tels processeurs spécialisés. Si l'utilisation exagérée des ombres portées ou des réflexions multiples a tendance à nuire à l'interprétation d'un objet, une transparence ou un reflet peut parfois aider à la compréhension en montrant des parties qui ne seraient pas visibles simplement. Or aujourd'hui, aucun algorithme réaliste n'est suffisamment rapide pour être utilisé de manière interactive. Même dans le cas où les performances ne sont pas prioritaires, il est naturel de chercher à réduire des temps de calcul

élevés. L'amélioration des performances des algorithmes de visualisation réaliste est toujours un problème ouvert.

Le travail présenté s'inscrit dans le projet SACADO, système de CFAO développé au Laboratoire de Recherche en Informatique de Metz. Notre souhait est de doter ce système d'un module de visualisation réaliste. Les objets de la scène sont représentés par des arbres CSG et nous voulons conserver les propriétés liées à ce modèle géométrique. En particulier, nous ne voulons pas passer par une facettisation. Contrairement à la démarche habituelle qui consiste à étudier un aspect précis d'un algorithme donné, nous abordons le problème de l'amélioration des algorithmes de visualisation sur un plan plus général. En nous plaçant dans le cadre d'une approche multimodèles des systèmes de CFAO, nous définissons un nouveau modèle de visualisation qui tient compte des contraintes fixées et dont les propriétés constituent à elles seules une amélioration. Dans la mesure où le choix du réalisme nous impose pratiquement l'algorithme du lancer de rayons, nous étudions en particulier les relations entre le modèle de visualisation et cet algorithme. Enfin, nous évoquons le problème du mixage des algorithmes et montrons notamment que le modèle de visualisation que nous avons défini permet d'appliquer le principe du mixage aux différentes représentations d'un objet.

Dans le premier chapitre, nous rappelons les notions de base du modèle CSG. Nous faisons également un bref tour d'horizon des principaux algorithmes de visualisation ainsi que des méthodes d'optimisation du lancer de rayons spécifiques à ce modèle. Dans le second chapitre, nous développons le modèle de visualisation que nous baptisons VSG (Visual Solid Geometry). Pour utiliser au mieux la décomposition induite par la construction des arbres VSG, nous décrivons dans le troisième chapitre une technique de classement a priori des objets pour améliorer les performances du lancer de rayons. Enfin, dans le quatrième chapitre, nous comparons, d'une part, les performances de la procédure d'intersection entre un rayon et un objet représenté dans les deux modèles CSG et VSG et, d'autre part, les performances des optimisations par classement a priori et par une méthode de subdivision spatiale classique. Dans l'annexe A sont décrits les algorithmes des principales procédures mises en oeuvre dans le lancer de rayons. L'annexe B récapitule l'ensemble des résultats relatifs aux comparaisons du chapitre 3. L'annexe C présente les images des scènes utilisées pour ces comparaisons.

Chapitre 1

De la modélisation à la visualisation réaliste

Recherche en Informatique de Metz dans le système SACADO est de type hybride incluant la représentation CSG.

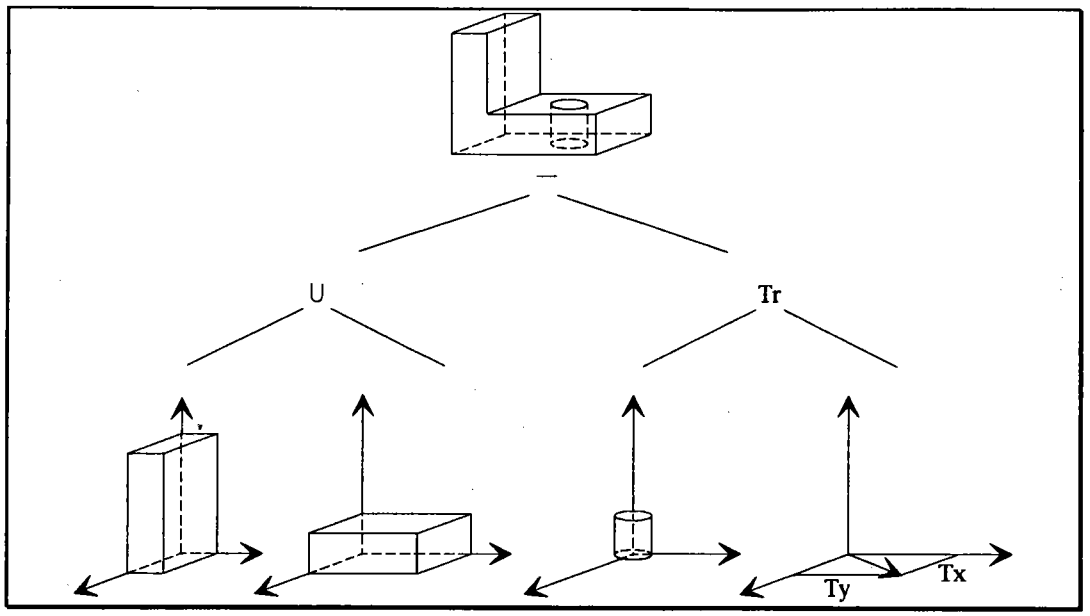


Figure 1 : Un arbre de construction.

Le modèle CSG (Constructive Solid Geometry) est l'un des modèles géométriques les plus utilisés de nos jours. Un solide peut être décrit comme un arbre binaire où les noeuds sont soit des opérations booléennes régularisées (union, intersection, différence) [REQ 80], soit des transformations géométriques (rotation, translation, homothétie ou certaines déformations libres [NIE 89]). Les feuilles contiennent soit des volumes primitifs, soit les paramètres des transformations géométriques (voir fig. 1). Il existe plusieurs variantes pour représenter des solides (arbre n-aire, graphe acyclique). Toutes ont le même domaine de représentation. Elles laissent plus ou moins de facilités à l'opérateur pour la description des objets. La définition du modèle CSG est récursive. Elle peut être représentée par la grammaire suivante:

<Arbre> ::= <Volume primitif> | <Arbre><Opérateur booléen><Arbre> |
<Arbre><Transformation géométrique><Paramètres>

Par conséquent, un noeud d'un arbre CSG est lui même un arbre CSG. C'est un objet à part entière qui correspond à une étape intermédiaire dans la construction de l'objet final. Le modèle CSG peut être vu comme un historique de construction.

L'ensemble des primitives est choisi de manière à s'adapter au mieux au domaine des objets à représenter. C'est lui en particulier, qui conditionne la puissance du modèle CSG.

Par exemple, si l'on ne considère que le cube, l'ensemble des objets représentables est l'ensemble des polyèdres. Dans un contexte mécanique, l'ensemble minimal des primitives est composé du cube, du cylindre, du cône, de la sphère et du tore. Il est souvent intéressant de compléter cet ensemble par d'autres primitives mêmes redondantes. Par exemple, il est préférable de considérer une pyramide à base carrée comme une primitive au lieu d'une différence entre un cube et quatre autres cubes. Le domaine de représentation n'est pas modifié mais la concision en est améliorée. En ce qui concerne le design ou les formes du vivant (visages, plantes, reliefs, ...), le modèle CSG n'est pas très bien adapté. L'intégration des surfaces gauches pose de nombreux problèmes qui ne sont pas encore bien résolus aujourd'hui.

Le modèle CSG est valide. A un objet représenté correspond un objet réel. Décrire un solide par un arbre est équivalent à l'écrire sous la forme d'une expression booléenne parenthésée. Il devient alors aisé de définir un langage de description permettant la création d'objets paramétrés de manière procédurale [BEI 88].

Le modèle CSG est relativement ouvert puisqu'il existe de nombreux algorithmes pour calculer diverses valeurs [LEE 82]. Cependant, l'un des inconvénients majeur est qu'il ne décrit pas de manière explicite les contours des objets. Chaque opération de calcul ou de visualisation nécessite l'évaluation souvent complète de l'arbre.

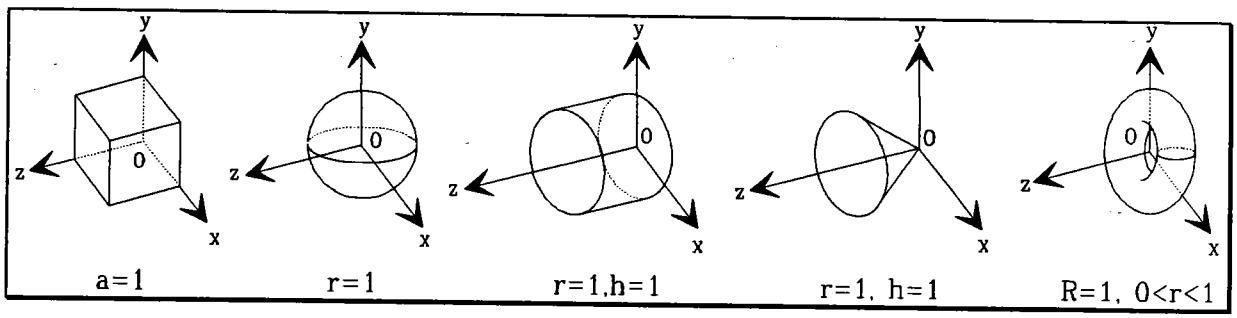


Figure 2 : Les primitives unitaires.

Dans le cadre de la visualisation, les structures décrites précédemment pour représenter un solide ne sont pas toujours satisfaisantes car elles entraînent souvent de nombreux calculs superflus alors que le gain en terme de performance est le souci principal. C'est pourquoi, on se ramène généralement à un arbre binaire. Lorsqu'un solide est représenté par un graphe, il est nécessaire de dupliquer les parties qui sont partagées par plusieurs arcs. De plus, pour simplifier les calculs et éviter de nombreux changements de repères, les transformations géométriques sont concaténées et associées aux primitives sous forme d'une matrice. Les primitives sont unitaires et placées dans un repère local (voir fig. 2).

3. Visualisation du modèle CSG.

Si l'on recense les algorithmes d'affichage existants pour le modèle B-Rep et ceux existants pour le modèle CSG, on constate que les premiers sont beaucoup plus nombreux que les seconds. Ce déséquilibre peut s'expliquer par le fait que certaines notions de cohérence [SUT 74] qui existent dans le modèle B-Rep (Boundary Representation ou représentation par les limites), n'existent pas dans le modèle CSG (cohérence d'arêtes et de faces notamment). Les raisons principales en sont que d'une part, les contours des objets ne sont pas décrits explicitement et que d'autre part, les primitives sont représentées de manière exacte (comme pour le B-Rep exact), par leur équation en général. C'est pourquoi de nombreux algorithmes exigent une approximation du modèle afin de retrouver ces notions de cohérence. Le schéma de la figure 3 illustre les différents moyens de résoudre le problème de l'affichage d'un modèle CSG.

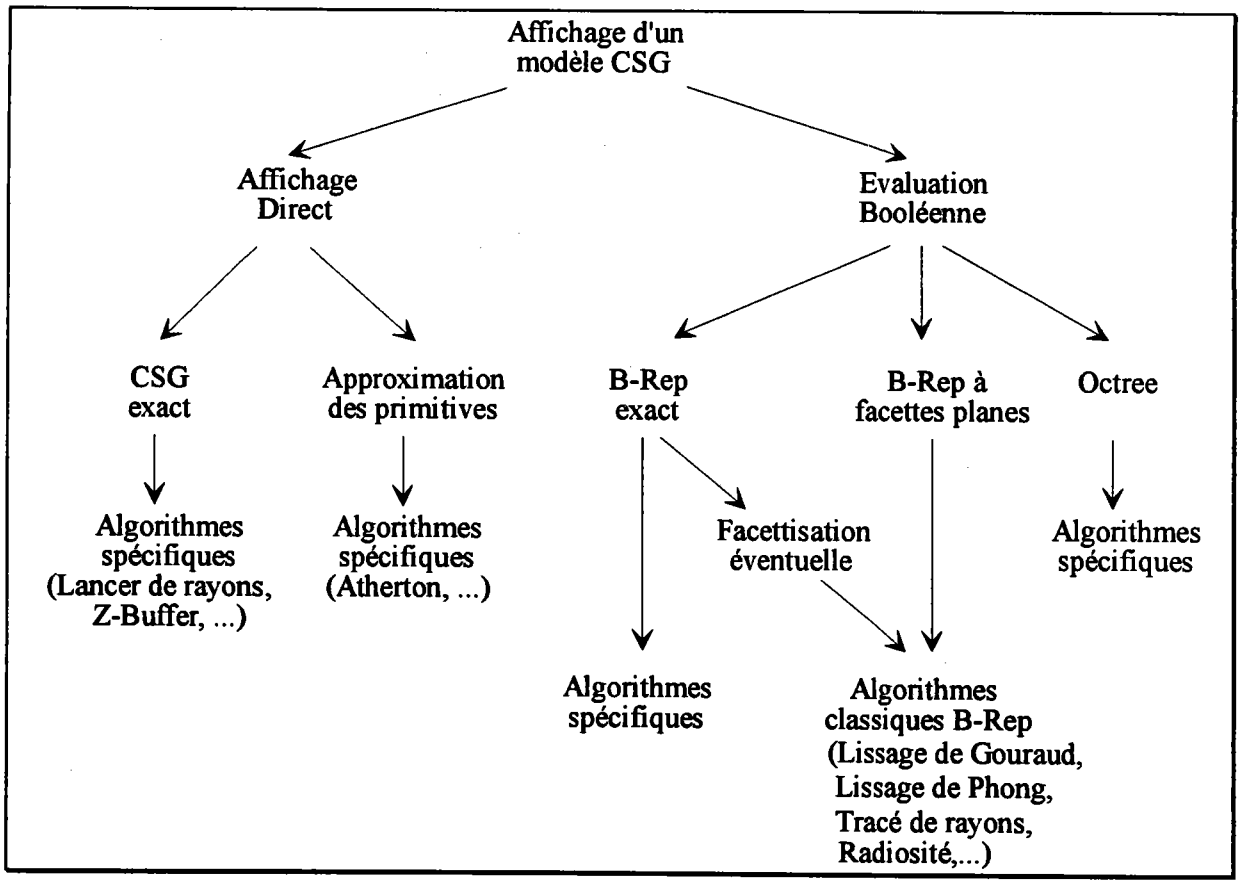


Figure 3 : Affichage d'un modèle CSG.

Dans la mesure où nous ne souhaitons pas passer par une évaluation booléenne (problème traité par ailleurs au LRIM [PER 94]), nous avons recentré notre étude sur les algorithmes

d'affichage directs. On distingue alors deux familles: les algorithmes d'affichage du modèle CSG exact et ceux où les primitives sont approchées par des facettes planes.

3.1. Affichage direct du modèle CSG exact.

3.1.1. Le tampon de profondeur (Z-Buffer).

L'algorithme du tampon de profondeur pour un arbre CSG, proposé par Rossignac et Requicha [ROS 86] est le suivant:

```
Pour chaque pixel (x,y) de l'écran faire
    z[x,y] ← infinité { initialisation de la profondeur }
    I[x,y] ← 0 { initialisation de l'intensité avec la couleur du fond }
Fpour

Pour chaque face de chaque primitive de l'arbre CSG faire
    Pour chaque point P du nuage représentant la face de la primitive faire
        (x,y) ← projection de P sur l'écran
        d ← profondeur de P par rapport au point de vue
        Si d < z[x,y] alors
            CP ← ClassePoint(P,Arbre CSG)
            Si CP = "sur la frontière du solide représenté par l'arbre CSG" alors
                z[x,y] ← d
                N ← normale de P
                I[x,y] ← intensité (P,N,sources lumineuses)
            Fsi
        Fsi
    Fpour
Fpour
```

Dans l'algorithme classique du tampon de profondeur appliqué à des facettes planes, le nuage de points représentant une face plane est obtenu par des algorithmes de discrétisation simples et bien connus [ROG 88]. A chaque pixel correspond un point de la face. Dans le modèle CSG (comme dans le B-Rep exact), il n'est pas toujours possible de discrétiser aussi précisément les primitives en particulier dans le cas de quadriques comme le tore. La solution proposée est d'utiliser l'équation paramétrique F(u,v) de la primitive ou d'une de ses faces et de choisir Δu et Δv, les pas de variation des paramètres, tels que le nuage produit soit suffisamment dense pour éviter des trous dans l'image finale.

La différence essentielle par rapport à l'algorithme classique est de déterminer si un point appartient bien à la surface du solide. Pour ce faire, les auteurs utilisent un algorithme de classification de point (ClassePoint). Son principe consiste à déterminer si un point est dans l'objet, hors de l'objet ou sur sa frontière. Pour ce faire, les auteurs classent le point par rapport aux primitives du solide, puis combinent ces résultats en fonction des opérations booléennes de l'arbre CSG. Par exemple, si un point P est dans A et est dans B alors P est dans $A \cap B$, si P est dans A et n'est pas dans B alors il est dans A-B, etc. Dans [JAN 91], Jansen décrit et compare les principales méthodes de classification de point existantes et leurs optimisations. Nous soulignons ici un problème commun à l'ensemble des algorithmes de visualisation: il existe des situations délicates (tangence entre des primitives notamment) qui nécessitent souvent un traitement particulier pour éviter l'affichage de volumes dégénérés (faces ou arêtes isolées) [TIL 80].

L'algorithme du tampon de profondeur, dans la version proposée par les auteurs, ne permet que l'élimination des parties cachées en prenant en compte l'illumination directe. Une implantation de cet algorithme sur une machine parallèle Pixel Plane [GOL 89] a permis d'obtenir des temps de calcul compatibles avec l'interactivité pour des objets de faible complexité.

3.1.2. La méthode des empans.

La méthode des empans sur un arbre CSG proposée par Pueyo et Mendoza [PUE 87], reprend le principe de l'algorithme de Watkins [WAT 70] pour les facettes planes. Elle consiste à calculer, pour chaque ligne de balayage, l'intersection entre le plan perpendiculaire à l'écran passant par cette ligne et les primitives des objets. L'ensemble des primitives est restreint aux blocs, cônes, cylindres et sphères. Par conséquent, les sections obtenues sont des coniques, des coniques tronquées ou des polygones (voir fig. 4). Elles sont caractérisées par deux types de points qui définissent des empans monotones : les points frontières (5, 8, 9, 10, 11, 12) et les extremums (1, 2, 3, 4, 6, 7). Un arc reliant deux points consécutifs est monotone. Une évaluation booléenne en deux dimensions sur ces sections permet de déterminer les empans visibles.

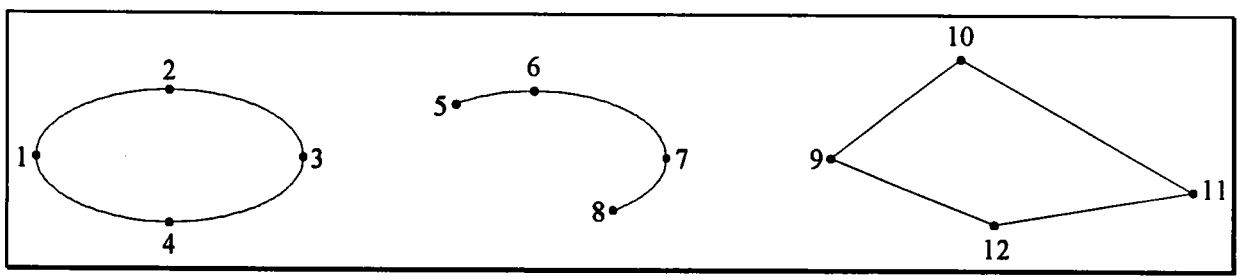


Figure 4 : Intersections entre une primitive et un plan.

Comme le tampon de profondeur, cet algorithme ne permet que l'élimination des parties cachées en tenant compte de l'illumination directe. L'inconvénient principal est qu'il est limité aux seules primitives citées précédemment. L'extension à d'autres primitives telles que le tore par exemple, semble difficile. L'intersection plan/quadrique pose des problèmes numériques, mais également des problèmes de performances qui ne peuvent être négligés dans un algorithme de visualisation.

3.1.3. Le lancer de rayons.

Le premier algorithme de lancer de rayons a été proposé par Appel en 1968 [APP 68] pour l'élimination des parties cachées. Trop coûteux en temps de calcul pour les machines de cette époque, il n'a pas connu de réel développement. Il n'est devenu véritablement populaire qu'à partir de 1980 [WHI 80] lorsqu'il permit de faire les premières images réalistes incluant ombres portées, réflexions et transparences. Il est basé sur la remarque suivante: l'oeil ne perçoit des objets que les rayons lumineux transmis par réflexion ou réfraction. L'idée est donc de retrouver la trajectoire de ces rayons pour déterminer l'éclairement des objets vus. Pour des raisons de complexité, le trajet des rayons est suivi à rebours, c'est-à-dire en partant de l'oeil vers les sources lumineuses. Calculer l'éclairement d'un point nécessite l'étude de plusieurs types de rayons : les rayons primaires déterminent les parties visibles de la scène, les rayons d'ombrage détectent les ombres portées et les rayons secondaires permettent d'obtenir l'éclairement indirect, c'est-à-dire par réflexion et transparence. Les trois types de rayons sont commentés dans les paragraphes qui suivent. Enfin, toutes les informations recueillies par le suivi de ces rayons sont utilisées dans un modèle d'éclairement pour déterminer réellement la couleur d'un pixel de l'écran. Un exemple est donné dans le paragraphe 3.1.3.4..

3.1.3.1. L'élimination des parties cachées.

Les rayons primaires sont lancés depuis l'oeil en passant par le centre de chaque pixel de l'écran. Si un rayon primaire rencontre plusieurs objets, le point d'intersection le plus proche de l'oeil est conservé (voir fig. 5). De ce point partent les rayons d'ombrage et secondaires.

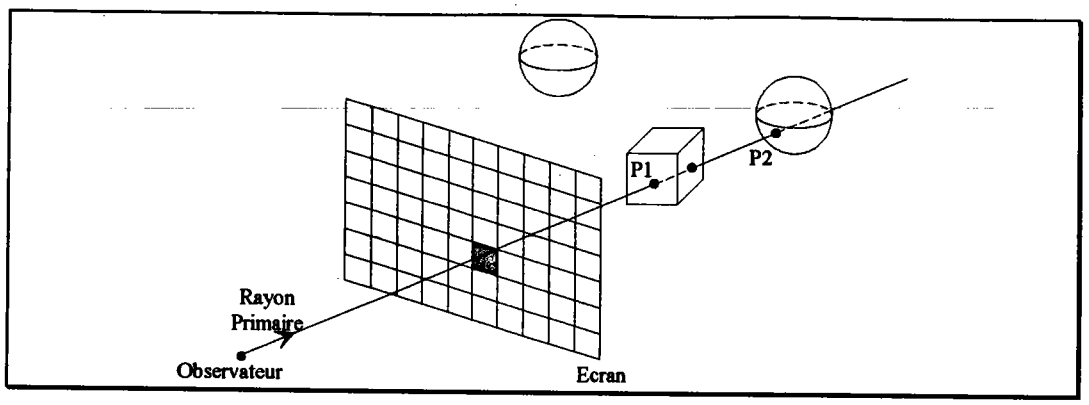


Figure 5 : Rayons primaires.

3.1.3.2. Détection des ombres portées.

La détection des ombres portées dans le lancer de rayons n'est possible que si les sources lumineuses sont ponctuelles. Une source non ponctuelle produit une zone de pénombre qui ne peut pas être traitée par l'approche classique que nous décrivons [WOO 90]. Les rayons d'ombrage sont lancés pour chaque rayon primaire ou secondaire.

Un rayon d'ombrage est un segment de droite ayant pour extrémités la source lumineuse et le point dont on veut calculer l'éclairement (voir fig. 6). Si un objet est rencontré entre ces deux extrémités alors le point se trouve dans l'ombre projetée par cet objet et la source lumineuse ne doit pas être prise en compte dans le calcul de l'éclairement.

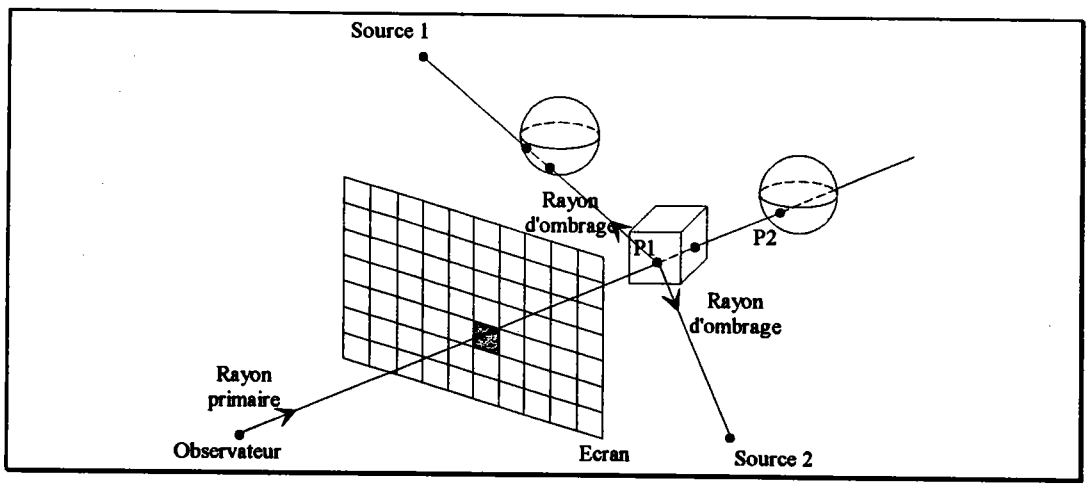


Figure 6 : Rayons d'ombrage.

3.1.3.3. Réflexion et réfraction.

Les rayons secondaires simulent les déviations subies par les rayons lumineux lorsque ceux-ci rencontrent des matériaux réfléchissants (miroir) ou transparents (vitres) (voir fig. 7). La mise en oeuvre des rayons secondaires est rendue possible grâce à deux restrictions importantes. D'une part les rayons lumineux sont considérés sans épaisseur et d'autre part l'étude des phénomènes optiques se limite à la seule composante spéculaire. La composante diffuse est généralement représentée par une constante.

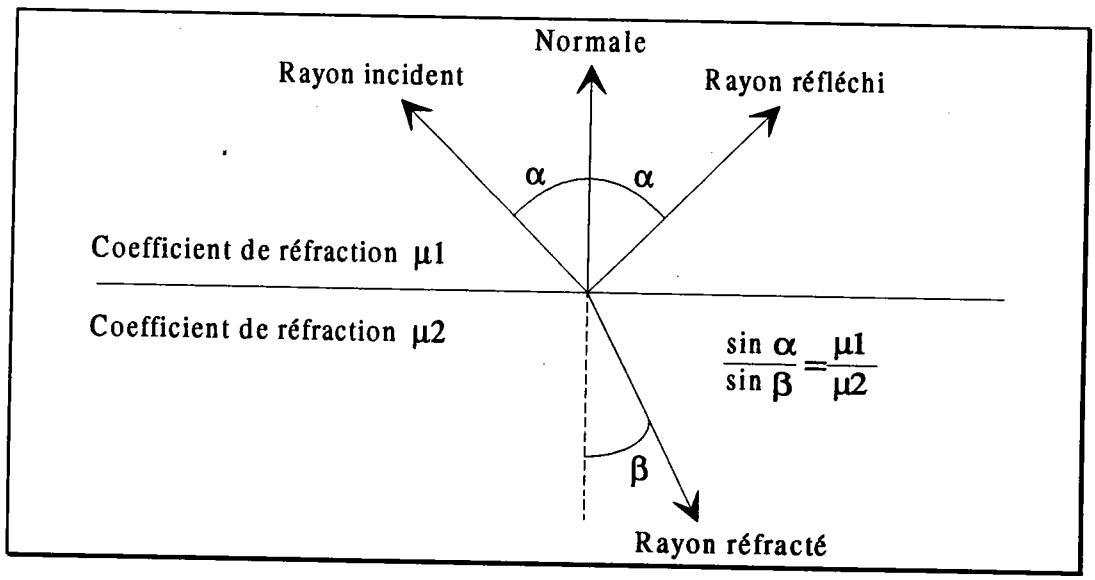


Figure 7 : Rayon réfléchi et rayon réfracté.

L'étude d'un rayon secondaire se déroule de la même manière que celle d'un rayon primaire. On recherche le point d'intersection le plus proche de l'origine du rayon. Son éclairement est déterminé en lançant de nouveaux rayons d'ombrages et éventuellement de nouveaux rayons secondaires. L'algorithme du tracé de rayons s'écrit simplement de manière récursive.

3.1.3.4. Le modèle d'éclairement.

L'ensemble des rayons lancés pour calculer l'éclairement d'un pixel constitue l'arbre de rayons (voir fig. 8). En général, la hauteur de cet arbre est fixée au préalable. Les informations recueillies par l'ensemble des rayons (couleurs, matériaux, normales, etc.) sont utilisées dans un modèle d'éclairement pour déterminer la couleur du pixel.

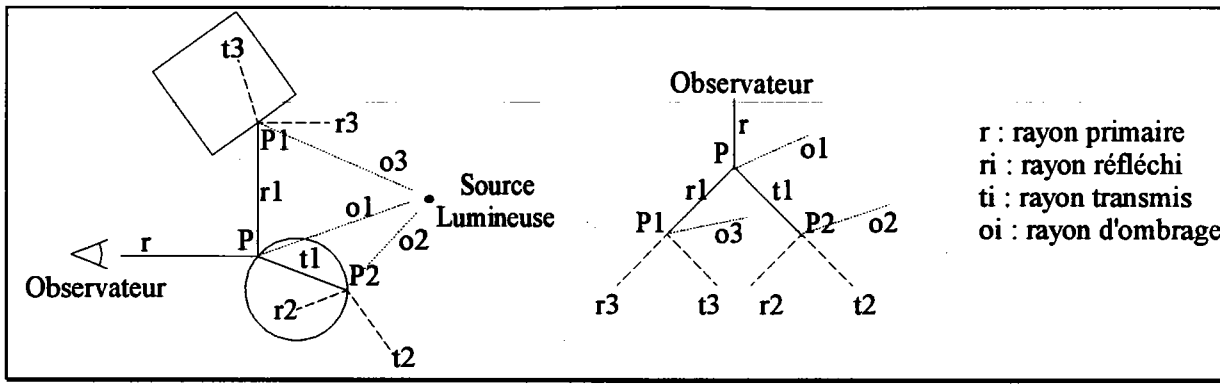


Figure 8 : Arbre de rayons.

Il existe de nombreux modèles d'éclairement allant du plus simple (modèle de Lambert) au plus réaliste (et beaucoup plus coûteux en temps de calcul) [COO 82] [SIL 91]. Nous ne donnons qu'un exemple basé sur le modèle de Whitted [WHI 80].

$$I_\lambda = I_{a\lambda}K_aO_{d\lambda} + \sum_{1\le i\le m} S_i f_{atti} I_{p\lambda i}\left[K_dO_{d\lambda}(\overline{N}\cdot\overline{L}_i)+K_s(\overline{N}\cdot\overline{H}_i)^n\right] + K_\lambda s I_\lambda s + K_\lambda t I_\lambda t$$

$I\lambda$: intensité lumineuse pour la longueur d'onde λ.
$I_{a\lambda}$: intensité ambiante.
K_a : coefficient ambiant.
$O_{d\lambda}$: composante diffuse de l'objet.
S_i : 1 si le point est éclairé par la $i^{ème}$ source, 0 sinon.
f_{atti} : facteur d'atténuation de la $i^{ème}$ source.
$I_{p\lambda i}$: intensité lumineuse de la $i^{ème}$ source.
$\overline{V}$: rayon.
$\overline{N}$: normale.
$\overline{L}i$: rayon d'ombrage vers la $i^{ème}$ source.
$\overline{H}i$: demi vecteur. $\overline{H}i=(\overline{V}+\overline{L}i)/2$.
$I_\lambda s$: intensité apportée par le rayon réfléchi.
$K_\lambda s$: coefficient spéculaire (0 = mat, 1 = miroir parfait).
$I_\lambda t$: intensité apportée par le rayon réfracté.
$K_\lambda t$: coefficient de transparence (0 = opaque, 1 = transparent).
n : valeur empirique simulant l'aspect brillant du matériau.

Pour un point donné, le modèle d'éclairement est appliqué pour l'ensemble des caractéristiques du matériau. Si le matériau est réfléchissant et/ou transparent, des rayons secondaires permettent de déterminer l'illumination indirecte du point. $I_\lambda s$ et $I_\lambda t$ sont les intensités lumineuses obtenues en appliquant le modèle d'éclairement aux points d'intersection déterminés par le rayon réfléchi et le rayon transmis. Le traitement récursif des rayons

lumineux permet donc d'évaluer l'arbre de rayons sans avoir à le représenter explicitement. L'ensemble des longueurs d'onde λ peut être réduit aux trois composantes du modèle RGB [PER 88][FOL 90] qui est en fait celui utilisé par les écrans graphiques. Cependant, cette simplification diminue le réalisme dans la mesure où ces trois longueurs d'onde ne sont pas toujours représentatives du spectre du matériau.

3.1.3.5. Intersection entre un rayon et un arbre CSG.

Nous supposons que l'arbre CSG est un arbre binaire sans transformation géométrique (celles-ci étant concaténées et associées aux primitives sous forme d'une matrice, ce qui est toujours possible). L'intersection entre un rayon et un arbre CSG ne consiste pas seulement à calculer les intersections avec les primitives mais encore et surtout à évaluer les opérations booléennes.

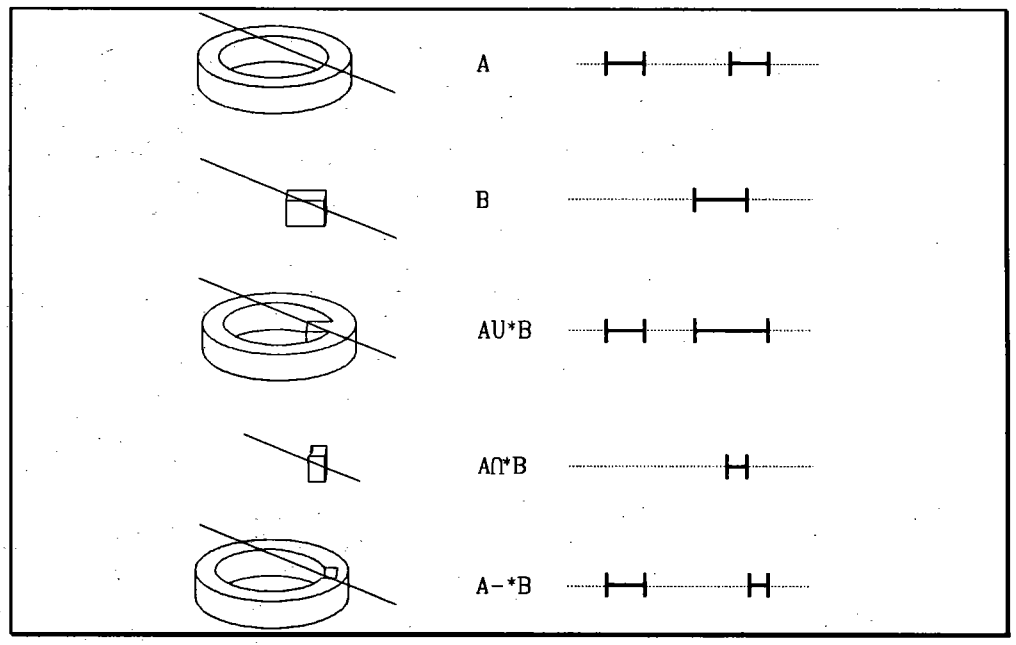


Figure 9 : Intersection rayon/arbre CSG.

Le principe est de ramener le problème en trois dimensions (opérations booléennes sur des volumes) en un problème à une dimension (opérations booléennes sur des segments) (voir fig. 9). L'algorithme de base proposé par Roth [ROT 82] est le suivant :

```
RayCasting (Rayon,Objet; VAR ListeSegments)
Début
    Si Objet=Primitive Alors
        CalculIntersection (Rayon,Objet,ListeSegments)
```

```
        Sinon
            RayCasting (Rayon,ArbreGauche(Objet),ListeGauche)
            RayCasting (Rayon,ArbreDroit(Objet), ListeDroite)
            ListeSegments = Combine(Operateur(Objet),ListeGauche,ListeDroite)
        Fsi
    Fin
```

La procédure *Combine* effectue l'opération booléenne sur les deux listes de segments *ListeGauche* et *ListeDroite*. A l'issue de l'algorithme, le point visible est la première extrémité du premier segment de *ListeSegments*. Afin de réduire le nombre de calculs, Roth propose d'associer à chaque noeud de l'arbre CSG un volume englobant. Si le rayon ne coupe pas l'englobant, il ne coupe pas le noeud correspondant. Dans le cas des opérateurs Intersection et Différence, si le rayon ne coupe pas le fils gauche, le résultat de l'opération booléenne est l'ensemble vide quel que soit le résultat de l'intersection avec le fils droit. En tenant compte de ces remarques, le nouvel algorithme est le suivant :

```
RayCasting (Rayon,Objet; VAR ListeSegments)
Début
  Si Intersection(Rayon,Englobant(Objet))=∅ Alors
        ListeSegments=∅
  Sinonsi Objet est une primitive Alors
        CalculIntersection (Rayon,Objet,ListeSegments)
  Sinon
        RayCasting (Rayon,ArbreGauche(Objet),ListeGauche)
        Si Operateur(Objet)=∪ ou ListeGauche≠∅ Alors
            RayCasting (Rayon,ArbreDroit(Objet), ListeDroite)
            ListeSegments = Combine(Operateur(Objet),ListeGauche,ListeDroite)
        Sinon
            ListeSegments=∅
        Fsi
  Fsi
Fin
```

3.2. Modèle CSG avec primitives polyédriques.

Le tampon de profondeur, la méthode des empans ou le lancer de rayons s'appliquent également lorsque les primitives sont facettisées. Nous rappelons simplement que la méthode des empans sur des primitives facettisées a été proposée par Atherton [ATH 83] et améliorée

par Bronstvoort [BRO 86]. Nous présentons deux autres méthodes originales permettant l'élimination des parties cachées.

3.2.1. Elimination des parties cachées par cohérence de fenêtre.

Le principe de la méthode proposée par Verroust [VER 87] consiste à subdiviser l'écran en fenêtres polygonales pour obtenir à l'intérieur de chacune d'entre elles, une liste de faces ordonnées selon leur profondeur. L'algorithme se décompose en trois étapes:

- A chaque primitive est associé un rectangle englobant dans le plan de l'écran. Lorsque plusieurs d'entre eux se recouvrent, ils sont subdivisés à leur tour. Finalement, à chaque rectangle est associée une liste de faces classées selon leur profondeur. Ils constituent les fenêtres de départ pour l'étape suivante.

- La deuxième étape subdivise récursivement les fenêtres pour obtenir un ordre constant des faces selon leur profondeur. Lorsque dans une fenêtre deux faces se coupent dans l'espace, cette fenêtre est découpée selon l'arête d'intersection. Ensuite, pour chaque fenêtre ayant un ordre constant, un rayon est lancé pour déterminer quelle est la face visible. Si aucune face n'est visible, la fenêtre est supprimée.

- La dernière étape consiste à fusionner les fenêtres contiguës qui ont la même face visible. L'image finale de type fil de fer est obtenue en affichant les faces visibles contenues dans les fenêtres.

3.2.2. L'algorithme de la liste de priorités.

L'algorithme de Jansen [JAN 85] consiste à appliquer le principe de la liste de priorités [NEW 72][FUC 80] sur les primitives facettisées de la scène. Le point fondamental de la méthode proposée (comme de tous les algorithmes de visualisation pour ce modèle) est de déterminer si un point appartient à la frontière de l'objet représenté par un arbre CSG. L'auteur décrit une structure permettant d'accélérer la classification des points. Il constate que dans la mesure où les faces sont classées, pour de nombreux pixels les primitives seront évaluées dans le même ordre. Ainsi, le résultat de l'évaluation d'un premier pixel peut être utilisé pour tous ceux qui partagent la même séquence. Il stocke alors ces dernières pour pouvoir les réutiliser au besoin (voir fig. 10).

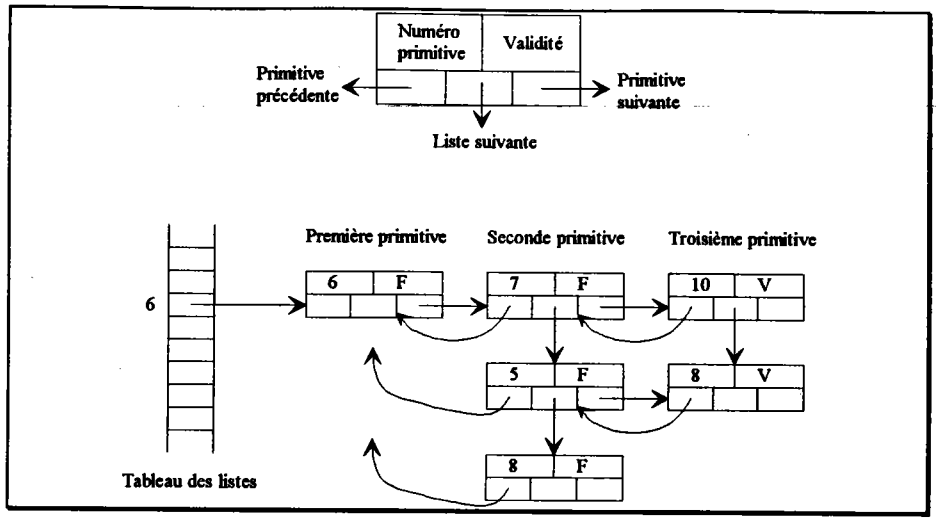


Figure 10 : Tableau des listes de séquences de primitives.

A chaque pixel de l'écran est associé un pointeur. Lorsqu'une face est discrétisée et qu'un pixel est étudié, son pointeur est positionné sur l'élément correspondant dans le tableau des listes. En fait, à chaque examen, le pointeur du pixel se déplace dans la structure en créant au besoin un nouvel élément. Lorsque l'élément pointé est valide, le point peut être affiché. Le pixel sera ignoré pour les faces suivantes. Cet algorithme permet un affichage par tache où les faces sont illuminées par le modèle de Lambert. L'auteur suggère d'augmenter le degré de polygonalisation des primitives pour obtenir un effet plus lisse.

La structure est indépendante de l'observateur. Chaque image l'enrichit de séquences nouvelles. De plus, les informations recueillies par l'élimination des parties cachées peuvent être utilisées ensuite pour améliorer les performances du lancer de rayons. L'inconvénient est qu'elle implique des coûts de gestion importants qui pénalisent les performances globales de l'algorithme.

3.3. Conclusion.

Hormis le lancer de rayons, toutes les méthodes décrites n'effectuent que l'élimination des parties cachées. Dans la mesure où le Z-Buffer existe pour le modèle CSG, il peut être utilisé dans les méthodes en deux passes pour obtenir des ombres portées [WIL 78] ou des réflexions [BLI 76]. Cependant, la méthode la plus simple et la plus complète (mais peut-être pas la plus rapide) reste le lancer de rayons. C'est encore aujourd'hui le seul algorithme, pour le modèle CSG exact, qui permet d'obtenir des images réalistes intégrant les ombres portées, les réflexions et les transparences. Ses performances étant relativement faibles, il est généralement accompagné d'une ou plusieurs optimisations.

4. Optimisations du lancer de rayons sur le modèle CSG.

L'inconvénient majeur du lancer de rayons est le temps de calcul d'une image. Cette caractéristique interdit son utilisation dans l'étape de conception où un maximum d'interactivité est recherché. Pourtant, la vocation de la visualisation, même réaliste, est bien de faire partie de cette étape. Un opérateur concevant une pièce doit pouvoir la voir à chaque instant, se déplacer autour ou à l'intérieur. Il doit pouvoir contrôler immédiatement les modifications qu'il y apporte. Et tout cela en temps réel ! Cela signifie que le système doit fournir une image tous les 25^e de seconde. Aujourd'hui, ces possibilités existent mais sur des représentations peu réalistes, généralement "fil de fer", ou sur des architectures spécialisées (implantation cablée de l'algorithme du Z-Buffer et du lissage des facettes planes par l'algorithme de Gouraud par exemple).

L'accroissement permanent des performances des nouveaux matériels réduit effectivement le temps de calcul. Mais la demande de réalisme est toujours plus grande ainsi que la complexité des scènes. L'objectif du temps réel ne pourra être atteint qu'en optimisant le lancer de rayons lui-même mais aussi en mettant en oeuvre des méthodes spécifiques à l'animation.

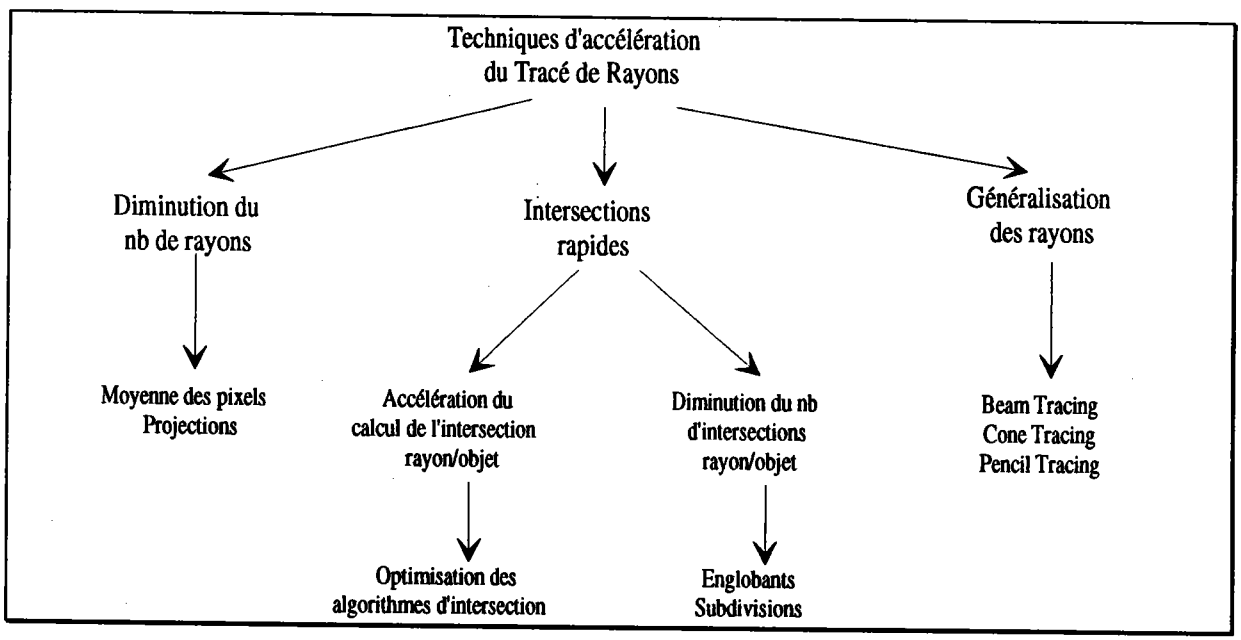


Figure 11 : Classification des optimisations.

4.1. Approche globale des optimisations.

De nombreuses optimisations ont déjà été proposées pour le lancer de rayons. Toutes n'agissent pas de la même manière pour réduire les temps de calculs. La classification de Vivian [VIV 93a] rappelle les différentes techniques (voir fig. 11). Ce schéma ne tient pas compte des

aspects matériels tel que le parallélisme qui peut également être considéré comme une optimisation.

Parmi toutes les solutions proposées, certaines ont été développées pour s'adapter au mieux au modèle CSG. Nous décrivons les principales d'entre elles dans le paragraphe suivant.

4.2. Les optimisations spécifiques au modèle CSG.

4.2.1. Les décompositions spatiales.

Le principe de la plupart des décompositions spatiales qu'elles soient régulières ou récursives, est de réduire l'espace contenant la scène, généralement représenté par un parallélépipède englobant, en éléments de taille réduite appelés voxels. Ainsi, lorsqu'un rayon est lancé, plutôt que de considérer toute la scène, seuls les objets contenus, totalement ou partiellement, dans les voxels traversés, sont pris en compte. La traversée s'arrête dès qu'une intersection est trouvée.

4.2.1.1. Décomposition spatiale binaire.

La subdivision spatiale proposée par Bouatouch, Madani, Priol et Arnaldi [BOU 87], est récursive. Mais à la différence des décompositions spatiales récursives classiques, elle n'est pas du type arbre octal. Initialement, la scène est représentée par un seul arbre CSG. Son englobant parallélépipédique est divisé successivement en deux. Le processus récursif de subdivision s'arrête lorsque le nombre de primitives contenues dans un voxel est inférieur à un nombre minimum ou lorsque le niveau de décomposition maximum est atteint. Ces deux critères sont fixés au départ par un opérateur. Cette méthode est similaire à celle de Kaplan [KAP 85], mais contrairement à cette dernière l'arbre binaire n'est pas conservé. Ses feuilles sont utilisées pour reconstruire une grille régulière, plus rapide à traverser (voir fig. 12).

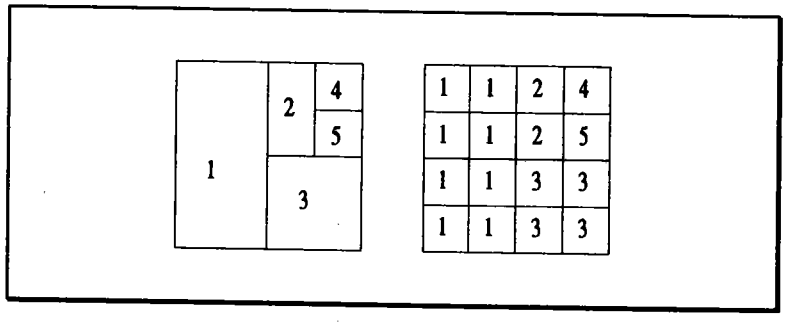


Figure 12 : Passage d'une décomposition récursive binaire à une grille régulière.

La spécificité de cette méthode apparaît lorsque la scène est modélisée par des arbres CSG. Les auteurs définissent d'abord la notion d'englobant minimum d'une primitive. C'est

l'englobant de la partie de la primitive effectivement utilisée dans l'objet. Il est obtenu en faisant l'intersection de l'englobant de la primitive avec celui de son noeud père. Ce principe peut être appliqué à chaque noeud en parcourant l'arbre des feuilles vers la racine. Nous verrons dans le paragraphe 4.2.2.1. que Cameron [CAM 90] va plus loin dans cette analyse.

Lors de la subdivision, les objets manipulés sont les englobants minimum des primitives. Les voxels contiennent donc une liste de primitives à partir desquelles est reconstitué un arbre CSG restreint. Les règles de reconstruction sont les suivantes :

On appelle R(Noeud) la restriction d'un noeud de l'arbre CSG.

1. Le noeud est une feuille : si la primitive P contenue dans le noeud est au moins partiellement intérieure (au sens de l'englobant) à la cellule alors R(Noeud)=P sinon R(Noeud)=∅.

2. Le noeud n'est pas une feuille (Noeud=A op B)

- si A≠∅ et B≠∅ alors R(Noeud)=A op B
- si A=∅ et B=∅ alors R(Noeud)=∅
- si A=∅ et B≠∅ alors
    - si op=∪ alors R(Noeud)=B
    - si op=∩ alors R(Noeud)=∅
    - si op=- alors R(Noeud)=∅
- si A≠∅ et B=∅ alors
    - si op=∪ alors R(Noeud)=A
    - si op=∩ alors R(Noeud)=∅
    - si op=- alors R(Noeud)=A

L'arbre CSG restreint est obtenu en appliquant ces règles dans un parcours de bas en haut de l'arbre CSG initial. Chaque voxel contient un seul arbre CSG restreint.

4.2.1.2. Décomposition de Wyvill, Kunii et Shirai.

La décomposition proposée par Wyvill, Kunii et Shirai [WYV 87] est basée sur un arbre octal. Mais en plus de réduire la complexité de la scène, les auteurs évaluent les opérations booléennes quand c'est possible. Lorsque cela ne l'est pas, la structure contient suffisamment d'informations pour qu'une intersection avec un rayon puisse être calculée sans revenir au modèle géométrique.

Un arbre octal peut contenir cinq types de feuilles différentes:

a) les feuilles vides.
b) les feuilles pleines.
c) les feuilles contenant une seule primitive et le reste de la cellule est vide.
d) les feuilles contenant une seule primitive et le reste de la cellule est plein.
e) les feuilles contenant plusieurs primitives et ne pouvant être subdivisées.

En ce qui concerne ce dernier type de feuille (e), c'est le seul cas où lors de la construction de l'arbre octal, les opérations booléennes ne peuvent être évaluées. Les auteurs proposent de leur associer l'équivalent d'un arbre CSG réduit, déduit des primitives contenues dans ces feuilles et du modèle CSG initial. Les feuilles de cet arbre sont du type a, b, c ou d. Les noeuds contiennent une opération booléenne (Intersection ou Différence). Un exemple est donné figure 13.

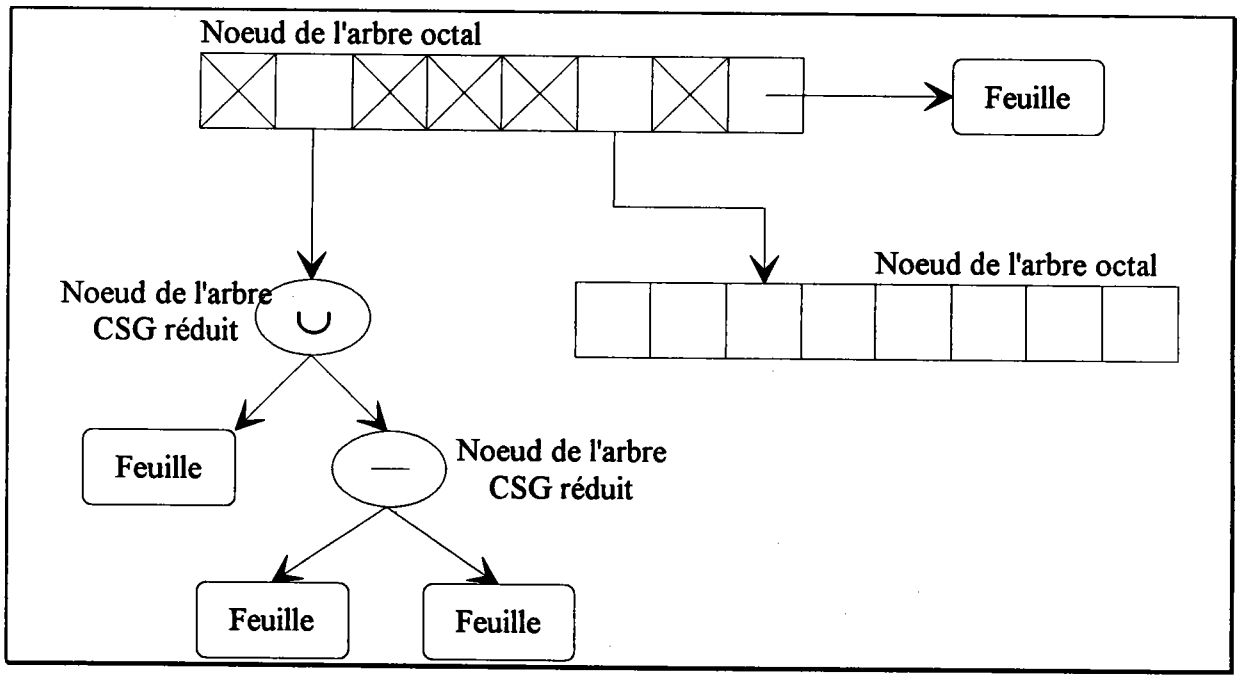


Figure 13 : Arbre octal de Wyvill, Kunii et Shirai.

L'arbre octal de la scène s'obtient en effectuant les opérations booléennes décrites dans le modèle CSG sur les arbres octaux associés à chacune des primitives. L'algorithme pour unir deux arbres octaux A et B est le suivant:

si A est vide **alors** le résultat est B
sinonsi B est vide **alors** le résultat est A
sinonsi A est plein **alors** le résultat est plein
sinonsi B est plein **alors** le résultat est plein

sinonsi le voxel a la taille mimimale **alors** créer un arbre réduit A∪B
sinon
Subdiviser A et B. Si A et B ne sont pas des feuilles cela consiste alors simplement à accéder aux fils. Si A ou B est une feuille, elle est alors décomposée.
Appliquer le processus récursivement à chacun des huit fils
fsi

L'algorithme pour soustraire deux arbres octaux est le suivant :

si A est vide **alors** le résultat est vide
sinonsi B est vide **alors** le résultat est A
sinonsi B est plein **alors** le résultat est vide
sinonsi A est plein et B est une feuille **alors** le résultat est le complémentaire de B dans le voxel
sinonsi le voxel a la taille minimale **alors** créer un arbre réduit A-B
sinon
Subdiviser A et B. Si A et B ne sont pas des feuilles cela consiste alors simplement à accéder aux fils. Si A ou B est une feuille, elle est alors décomposée.
Appliquer le processus récursivement à chacun des huit fils
fsi

Lorsqu'un noeud d'un arbre octal est subdivisé, pour chacun des huit coins d'un voxel créé, on détermine s'il appartient ou non à la primitive. Si tous les points sont dans la primitive, le voxel est plein. Si tous les points sont à l'extérieur, le voxel est vide.

4.2.2. Les volumes englobants.

Dans un modèle CSG, l'une des informations essentielles est l'englobant. On a vu qu'à chaque noeud d'un arbre CSG peut être associé un englobant. Il peut être parallélépipédique ou sphérique. L'initialisation des englobants parallélépipédiques des noeuds d'un arbre CSG est obtenue en effectuant un parcours de bas en haut en appliquant les règles suivantes [ROT 82] :

Soit un noeud T = A op B
Si op = ∪ alors Eng(T)←Eng(Eng(A)∪Eng(B))
Si op = ∩ alors Eng(T)←Eng(A)∩Eng(B)
Si op = - alors Eng(T)←Eng(A)

Les englobants sont très souvent utilisés pour accélérer les traitements sur les arbres CSG. Ils servent parfois même de représentation de base comme dans les subdivisions. Il est donc très important de gérer cette information le mieux possible.

4.2.2.1. Raffinement des englobants.

La méthode de raffinement des englobants (voir fig. 14) proposée par Cameron [CAM 90] repose sur deux théorèmes:

1) Théorème de montée: soit T=A op B un arbre CSG

a) op = $\cap$ alors Eng(T)$\leftarrow$Eng(T)$\cap$[Eng(A)$\cap$Eng(B)]
b) op = $\cup$ alors Eng(T)$\leftarrow$Eng(T)$\cap$[Eng(Eng(A)$\cup$Eng(B))]
c) op = - alors Eng(T)$\leftarrow$Eng(T)$\cap$Eng(A)

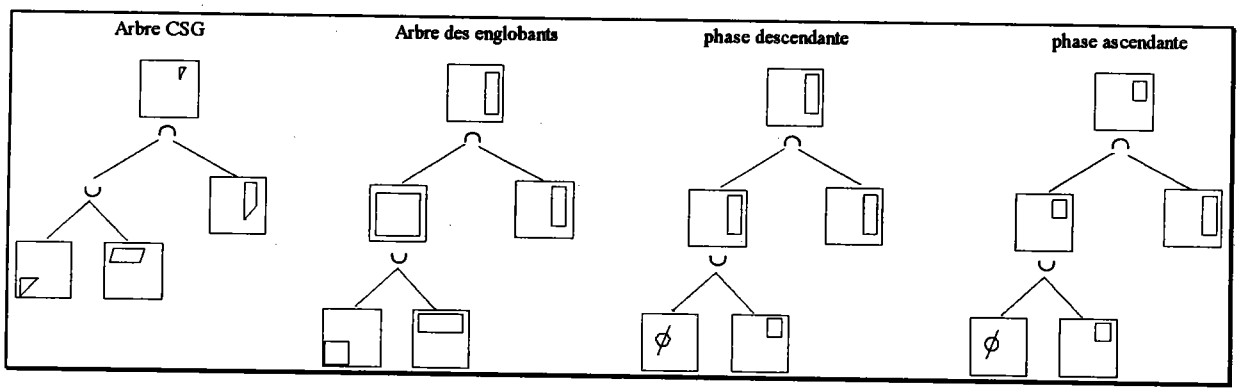


Figure 14 : Raffinement des englobants.

2) Théorème de descente : soit T= A op B un arbre CSG

Eng(A)$\leftarrow$Eng(A)$\cap$Eng(T) et Eng(B)$\leftarrow$Eng(B)$\cap$Eng(T)

Si l'on suppose les englobants des noeuds initialisés, l'application successive des deux théorème (descente puis montée) converge vers l'englobant minimal de l'arbre CSG. L'englobant minimal est défini ici comme le plus petit englobant que l'on puisse obtenir à partir des englobants des primitives. Il va de soi que le plus petit englobant réel est celui de l'objet même. Dans [CAM 92], les auteurs ont montré que dans le cas des englobants parallélépipédiques, il existe un point fixe unique obtenu en exécutant au plus n fois les deux théorèmes où n représente le nombre de primitives dans l'arbre CSG. Cette méthode permet d'obtenir également l'englobant minimal pour chaque noeud de l'arbre.

4.2.2.2. Intervalle englobant.

Dans la méthode proposée par Bronsvoort, Van Wyk et Jansen [BRO 84], les englobants parallélépipédiques sont projetés sur le plan de l'écran. De nouveaux englobants en deux dimensions, c'est-à-dire des rectangles, sont reconstruits. A chaque ligne de balayage de l'écran est alors associée une liste d'intervalles englobants des primitives. Le test d'intersection rayon/englobant est donc ramené à un test d'appartenance pixel/intervalle. A partir de la liste des primitives concernées par le pixel, ils déterminent un arbre CSG actif en restreignant l'arbre initial par un système de règles équivalent à celui de Bouatouch (cf. §4.2.1.1.). Cet arbre actif peut être réutilisé pour plusieurs pixels consécutifs.

D'après l'évaluation faite par les auteurs, cette méthode permet de réduire le nombre d'appels récursifs au lancer de rayons (entre 25% et 45%). Cependant les traitements liés à la recherche de l'arbre actif ainsi qu'à la gestion des intervalles englobants ne permettent pas d'avoir des gains en temps CPU très importants (entre 5% et 15%). Un autre inconvénient des intervalles englobants est qu'ils ne peuvent être appliqués qu'aux rayons primaires puisqu'ils sont liés au balayage de ligne.

4.2.2.3. Englobants relatifs.

Dans le modèle CSG, un objet est obtenu en combinant des primitives grâce à des opérations booléennes. Généralement, une partie seulement de chaque primitive intervient dans la forme finale de l'objet. Sandouk, Caubet, Natallah et Djedi [SAN 90] calculent cette partie "active" pour ensuite en déterminer l'englobant.

Dans la combinaison entre deux primitives A et B, on distingue deux parties. D'une part, la partie intérieure de A, notée PIn(A), est la partie de la primitive incluse dans la primitive B. D'autre part, la partie extérieure de A, notée PExt(A), est la partie qui n'est pas incluse dans B. On a $A=PIn(A)\cup PExt(A)$. On applique le même raisonnement pour obtenir PIn(B) et PExt(B). On peut remarquer que $PIn(A)=PIn(B)=A\cap B$. La partie active d'une primitive dépend de l'opération booléenne dans laquelle elle est impliquée:

1) $O = A \cap B$: $PAct(A) = PIn(A)$; $PAct(B) = PIn(B)$

2) $O = A - B$: $PAct(A) = PExt(A)$; $PAct(B) = PIn(B)$

3) $O = A \cup B$: $PAct(A) = PExt(A)$; $PAct(B) = Pext(B)$

L'englobant relatif d'une primitive est défini comme l'englobant de sa partie active. On a $Eng(A)=Eng(PAct(A))$. Pour calculer la partie active d'une primitive, deux solutions sont proposées. La première est analytique. Elle consiste à calculer l'intersection entre deux primitives à partir de leurs équations. La deuxième est numérique et nécessite la facettisation des primitives.

Cette définition des englobants relatifs appelle deux remarques. La première concerne la validité de la règle 3). La partie active d'une primitives intervenant dans une union est la primitive toute entière. On ne peut négliger la partie intérieure, qui fait aussi partie de l'objet. La règle 3), telle qu'elle est présentée conduit à des erreurs. La seconde remarque concerne l'intérêt de cette méthode étant donné qu'elle nécessite l'évaluation des opérations booléennes (même partielle) entre les primitives. Ces calculs paraissent relativement compliqués et coûteux pour une amélioration toute relative.

4.2.3. Accélération du calcul d'intersection.

4.2.3.1. Automate d'intersection.

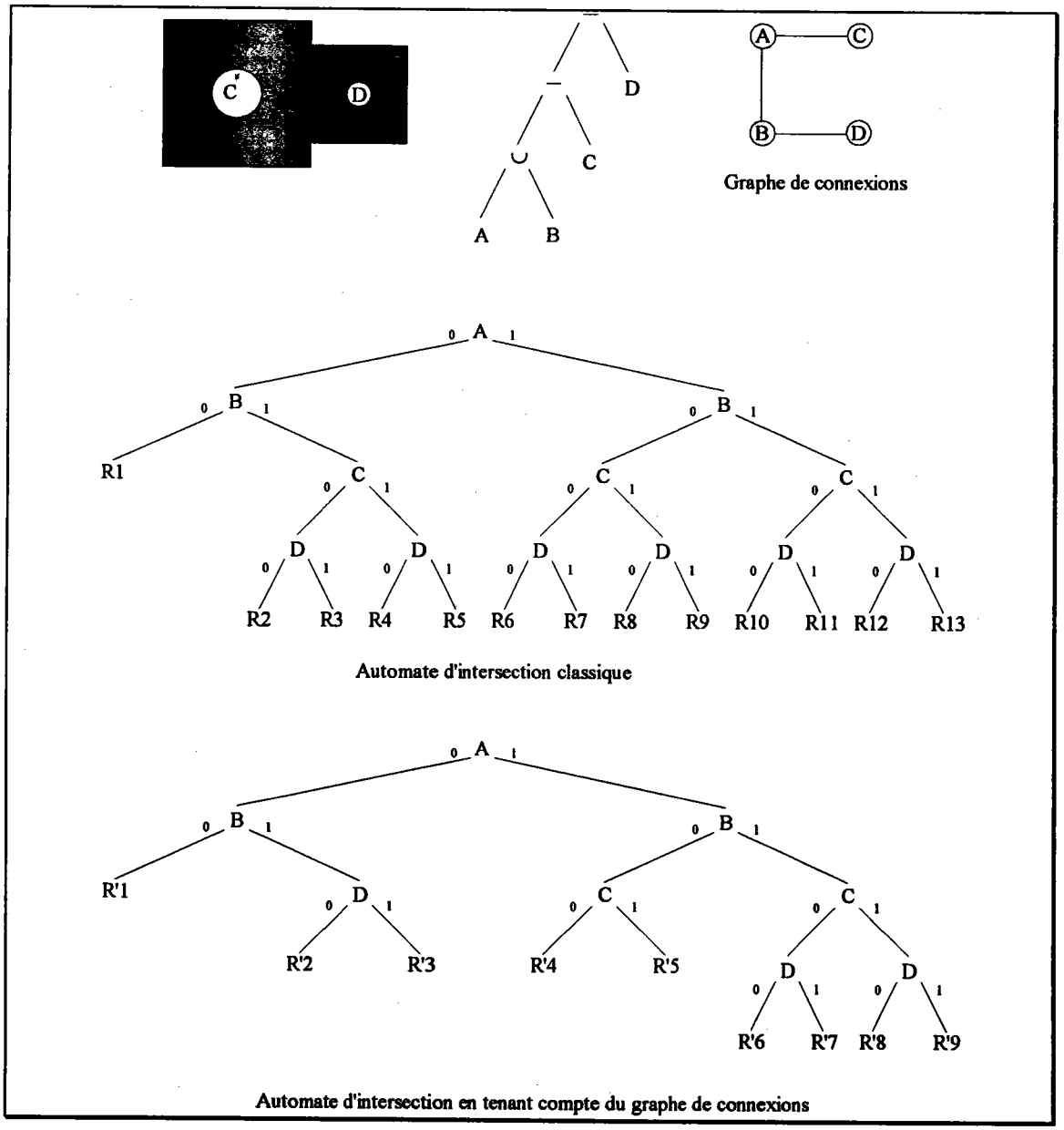


Figure 15 : Automate d'intersection. Les valeurs 1 et 0 indiquent que le rayon coupe ou non la primitive. Les Ri représentent les différents états possibles du rayon.

La procédure de lancer de rayons proposée par Roth (cf. §3.1.3.5.) ne tient compte que de l'ordre de construction de l'arbre CSG. La géométrie n'intervient que lors du calcul d'intersection entre le rayon et les primitives. Sandouk, Caubet, Matallah et Djedi [SAN 90] utilisent un automate d'intersection qui est en fait une synthèse entre la méthode de construction représentée par l'arbre CSG et les liaisons géométriques qui existent entre les primitives.

La construction de l'automate se décompose en deux étapes. La première consiste à créer un graphe de connexions. Les noeuds représentent les primitives. Un arc indique que les deux primitives se recoupent. Dans la seconde étape, l'automate est construit en évaluant l'arbre CSG et en tenant compte du graphe de connexions (voir fig. 15).

Cette structure est intéressante car elle élimine la récursivité et permet d'obtenir directement le chemin le plus court contenant les primitives devant être examinées par le rayon. De plus, le graphe de connexions permet éventuellement de simplifier l'automate d'intersection en diminuant le nombre des chemins potentiels et en réduisant la longueur de ces derniers. Il permet de prendre en compte des informations géométriques, les intersections entre primitives, pour réduire le nombre de primitives étudiées et donc accélérer le calcul d'intersection rayon/arbre CSG.

4.2.3.2. Les zones actives.

Les zones actives ont été proposées par Rossignac et Voelker [ROS 89] pour accélérer certains traitements (évaluation booléenne, élimination des noeuds redondants, détection des objets nuls et lancer de rayons entre autres). La zone active Z d'un noeud T d'un arbre CSG est la région dans laquelle toute modification de T affecte la forme du solide. A contrario, une modification hors de cette zone n'a pas de conséquence (voir fig. 16). Une zone active est définie algébriquement comme l'intersection de certains noeuds de l'arbre CSG. Elle est donc également représentée par un arbre CSG mais de taille plus réduite que celui du solide. Les auteurs montrent en particulier comment les zones actives permettent d'améliorer efficacement les algorithmes de classification.

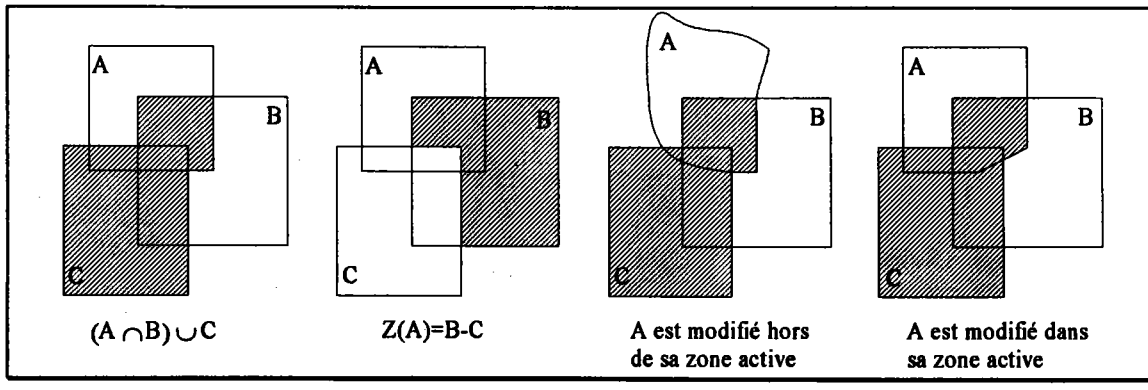


Figure 16 : Zone active.

La zone active Z(T) du noeud T dans l'arbre CSG est définie par la différence :

$$Z(T) = I(T) - U(T)$$

où I(T) et U(T) sont appelées respectivement I-zone et U-zone. Leurs règles de construction dépendent du signe du noeud T. Ce dernier est obtenu de la manière suivante: la racine de l'arbre est positive; si c'est une différence, les signes des noeuds du fils droit sont inversés. Dans tous les autres cas ils restent inchangés. Les zones actives des noeuds de l'arbre sont construites de manière récursive descendante. La I-zone de la racine est initialisée à Ω (l'espace entier) et la U-zone à $\varnothing$ (voir fig. 17).

Pour un noeud positif T les règles de construction sont les suivantes :

- Si $T = Tg \cup Td$ alors:

  $I(Tg) = I(T)$; $I(Td) = I(T)$

  $U(Tg) = U(T) \cup Td$; $U(Td) = U(T) \cup Tg$

- Si $T = Tg \cap Td$ alors:

  $I(Tg) = I(T) \cap Td$; $I(Td) = I(T) \cap Tg$

  $U(Tg) = U(T)$; $U(Td) = U(Td)$

- Si $T = Tg - Td$ alors:

  $I(Tg) = I(T) \cap \overline{Td}$; $I(Td) = I(T) \cap Tg$

  $U(Tg) = U(T)$; $U(Td) = U(T)$

Pour un noeud négatif T on a:

- Si $T = Tg \cup Td$ alors:

  $I(Tg) = I(T) \cap \overline{Td}$; $I(Td) = I(T) \cap \overline{Tg}$

  $U(Tg) = U(T)$; $U(Td) = U(T)$

- Si $T = Tg \cap Td$ alors:

  $I(Tg) = I(T)$; $I(Td) = I(T)$

  $U(Tg) = U(T) \cup \overline{Td}$; $U(Td) = U(Td) \cup \overline{Tg}$

- Si T = Tg - Td alors:

$I(Tg) = I(T)$; $I(Td) = I(T)$

$U(Tg) = U(T) \cup Td$; $U(Td) = U(T) \cup \overline{Tg}$

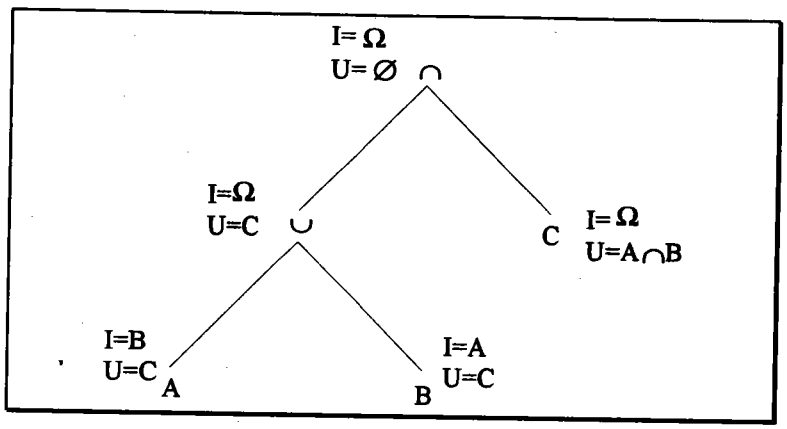


Figure 17 : Construction des I-zones et des U-zones.

La I-zone et la U-zone sont des arbres CSG dont la taille est généralement plus petite que celle de l'arbre CSG initial notamment lorsque ce dernier comporte des unions. Certaines implantations du lancer de rayons sont organisées de la manière suivante:

- pour chaque rayon, calculer toutes les intersections (d'entrée et de sortie) avec toutes les primitives.
- trier les points d'intersection selon leur profondeur depuis l'origine du rayon.
- rechercher les segments correspondant à la matière traversée par le rayon.

Les auteurs montrent que si un rayon coupant une primitive P engendre le segment [A,B], alors si A n'appartient pas à la I-zone de P, le segment n'appartient pas au solide et A n'est pas visible. La taille de la I-zone étant inférieure ou égale à celle de l'arbre CSG de l'objet, le traitement en est accéléré.

5. Conclusion.

L'objectif fixé est de produire des images réalistes en conservant les caractéristiques du modèle géométrique de SACADO. Après avoir rappelé les principes de la modélisation CSG, nous avons montré que le lancer de rayons est actuellement le principal algorithme permettant d'obtenir des images réalistes pour ce type de représentation. Les temps de calcul de ces

images étant très importants, il est nécessaire de les réduire par l'application d'une ou plusieurs méthodes d'optimisation. Nous avons rappelé les méthodes spécifiques au modèle CSG.

Si l'on observe les décompositions spatiales, l'automate d'intersection ou les zones actives, on s'aperçoit que ces méthodes ont un point commun. Toutes cherchent à transformer l'arbre CSG initial pour obtenir une représentation mieux adaptée à la visualisation. Bouatouch reconstruit un arbre CSG partiel en fonction des informations contenues dans les voxels. Après avoir effectué des opérations booléennes entre des arbres octaux, Wyvill obtient également des arbres CSG réduits dans les feuilles de l'arbre final. Sandouk n'utilise plus le modèle CSG générique mais un graphe permettant de déterminer la séquence de primitives à étudier. Enfin, Rossignac et Voelker associent à chaque noeud deux autres arbres CSG plus représentatifs de la partie active de ce noeud. Plusieurs de ces optimisations sont complémentaires. Par, exemple, on peut associer la décomposition spatiale de Bouatouch, l'automate d'intersection et les zones actives. Cé qui implique donc le mise en oeuvre de trois représentations différentes. Ces travaux montrent que le modèle CSG n'est pas bien adapté à la visualisation. Ils soulignent la nécessité de développer un véritable modèle applicatif dédié à cette opération. Des approches similaires dans des domaines différents (l'extraction des caractéristiques de formes en particulier) ont eu des résultats intéressants. Dans le deuxième chapitre, nous proposerons un nouveau modèle de visualisation dont les propriétés permettront d'éviter ces multiples représentations.

Chapitre 2

Les arbre VSG : un modèle de visualisation

1. Introduction.

On a constaté dans le chapitre 1, que l'un des objectifs recherchés par la plupart des optimisations est de simplifier ou de modifier la représentation générique de l'objet pour accélérer le lancer de rayons. Les solutions proposées ne répondent qu'à des problèmes locaux. Elles sont issues d'un raisonnement qui part des algorithmes et qui aboutit finalement à la transformation du modèle. L'approche proposée est plus générale et suit le raisonnement inverse. En nous appuyant sur des travaux existants, nous définissons les caractéristiques de base d'un modèle de visualisation dérivé du modèle CSG de manière à ce qu'il comprenne des propriétés recherchées pour le lancer de rayons.

L'une des principales propriétés du modèle CSG est la non-unicité. Un même objet peut être décrit de différentes façons. Cette particularité est basée sur deux caractéristiques fondamentales de cette représentation :

- Mathématiquement, pour une même expression booléenne, il peut exister une ou plusieurs expressions équivalentes (voir fig. 1b). Par exemple $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$, $(A-B)-C = A-(B \cup C)$, etc.

- Géométriquement, il est possible de créer le même objet à partir d'ensembles différents de primitives (voir fig. 1c).

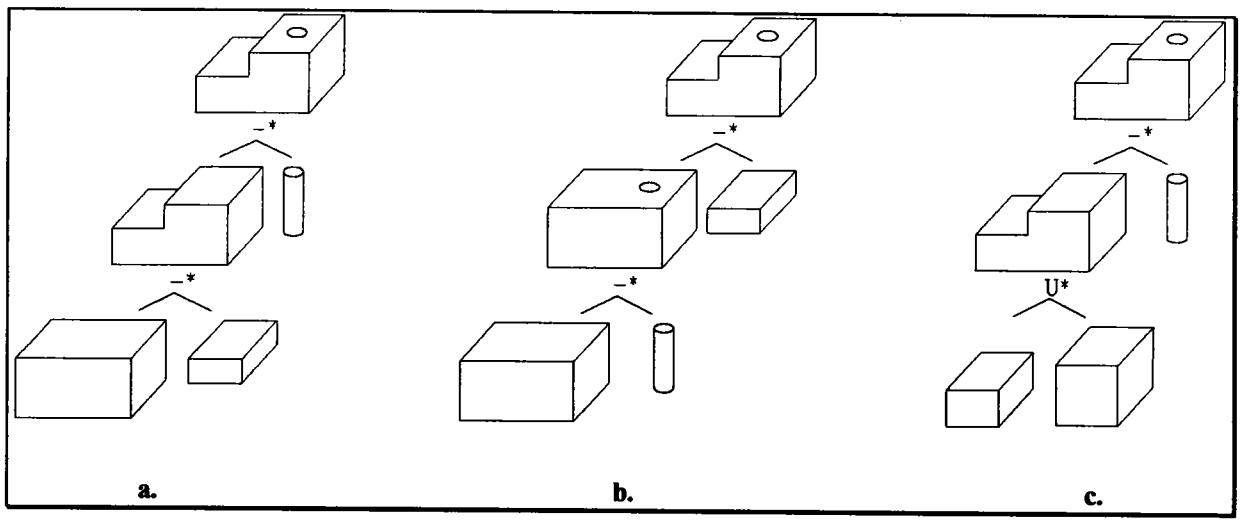


Figure 1 : Non-unicité du modèle CSG.

Malgré cette forte propriété, qui peut être considérée comme un inconvénient notamment pour montrer que deux objets représentés par un arbre CSG sont identiques, peu de travaux

se sont consacrés à la transformation des arbres CSG pour résoudre des problèmes plus simplement ou plus rapidement.

Cette étude d'un modèle de visualisation s'inscrit dans la réalisation du système de CFAO SACADO développé au L.R.I.M.. Nous commençons donc par rappeler quelques concepts sur l'approche multimodèles. Nous décrivons ensuite les principaux travaux sur la transformation des arbres CSG. Dans le paragraphe 3 nous développons la notion d'arbres VSG (Visual Solid Geometry), leur mode de construction et leurs propriétés.

2. L'approche multimodèles et la transformation des arbres CSG.

Un système de CFAO évoque souvent une image réductrice où son rôle est confondu avec dessin assisté par ordinateur et éventuellement pilotage de machine outil. En réalité, il intègre un ensemble de processus allant de la conception à la fabrication en prenant en compte, outre les aspects techniques classiques, des contraintes auxquelles doit répondre le produit ou l'objet conçu.

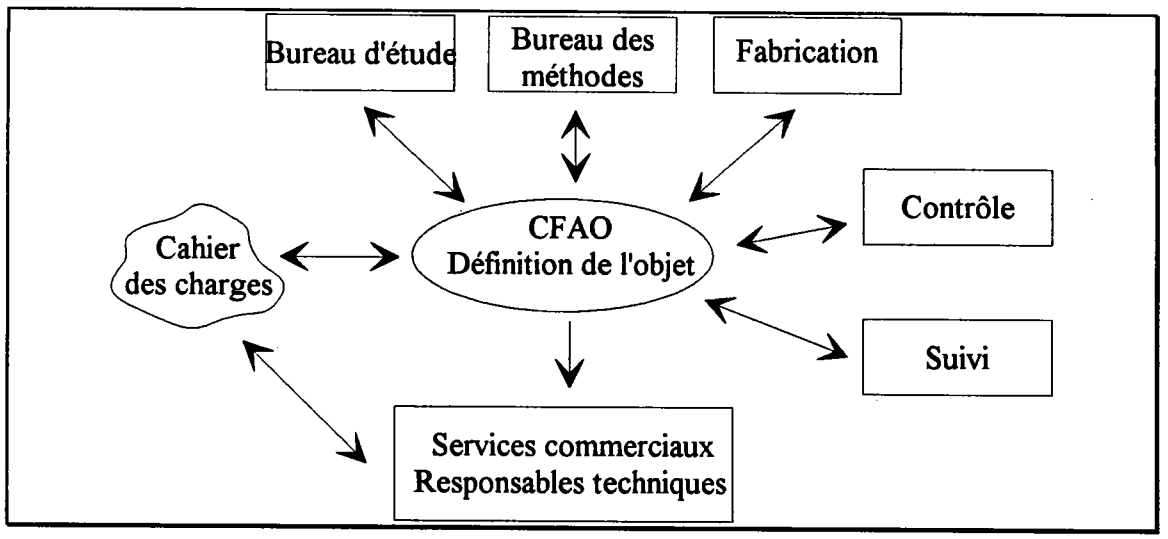


Figure 2 : un système de CFAO.

La figure 2 montre la place du système de CFAO dans l'entreprise et décrit ses liaisons avec les différents acteurs. Leur variété implique que la définition de l'objet, qui occupe le coeur du système, soit aussi complète que possible pour pouvoir répondre aux différents types de requêtes. Le modèle géométrique est un exemple de représentation que contient un système de CFAO, mais il n'est pas suffisant. C'est pourquoi il est accompagné d'autres représentations (géométriques ou non) convenant mieux aux opérations habituelles du système. On distingue deux types de modèles :

- Le modèle générique : il contient l'ensemble des informations géométriques, fonctionnelles et structurelles. C'est une véritable maquette virtuelle de l'objet. Il constitue le noyau du système de CFAO et est unique.

- Les modèles applicatifs : Ce sont des représentations souvent réduites de l'objet, construites à partir d'informations particulières provenant toutes du modèle générique. Modèle de fabrication, de simulation et de dialogue sont les exemples les plus courants.

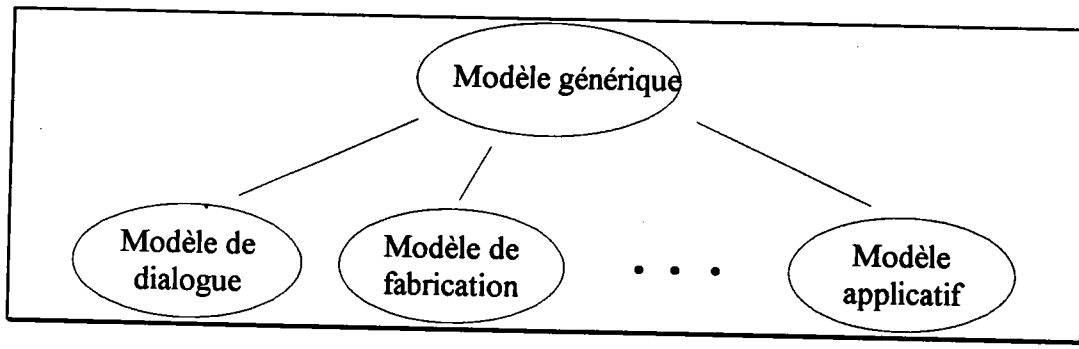


Figure 3 : Un multimodèle naïf.

La figure 3 décrit une organisation naïve de ces différents modèles. Le multimodèle MCAO1 introduit dans [GAR 82] prend en compte des notions plus générales telles que la gestion de projets. Son architecture (voir fig. 4) est composée, en plus du modèle générique et des modèles applicatifs, des éléments suivants :

- Les modèles dégradés : ce sont des modèles qui ont été construits à partir du modèle générique mais qui ne lui sont plus liés. Les modifications du modèle générique ne sont pas répercutées sur les modèles dégradés. Ils permettent en particulier de conserver un historique de l'objet.

- Les processus de localisation et de globalisation sont chargés respectivement du passage du modèle générique aux modèles applicatifs et inversement.

- La base d'algorithmes comprend un ensemble de programmes partageables par tous les processus applicatifs. Ils peuvent être dépendants ou indépendants du modèle générique.

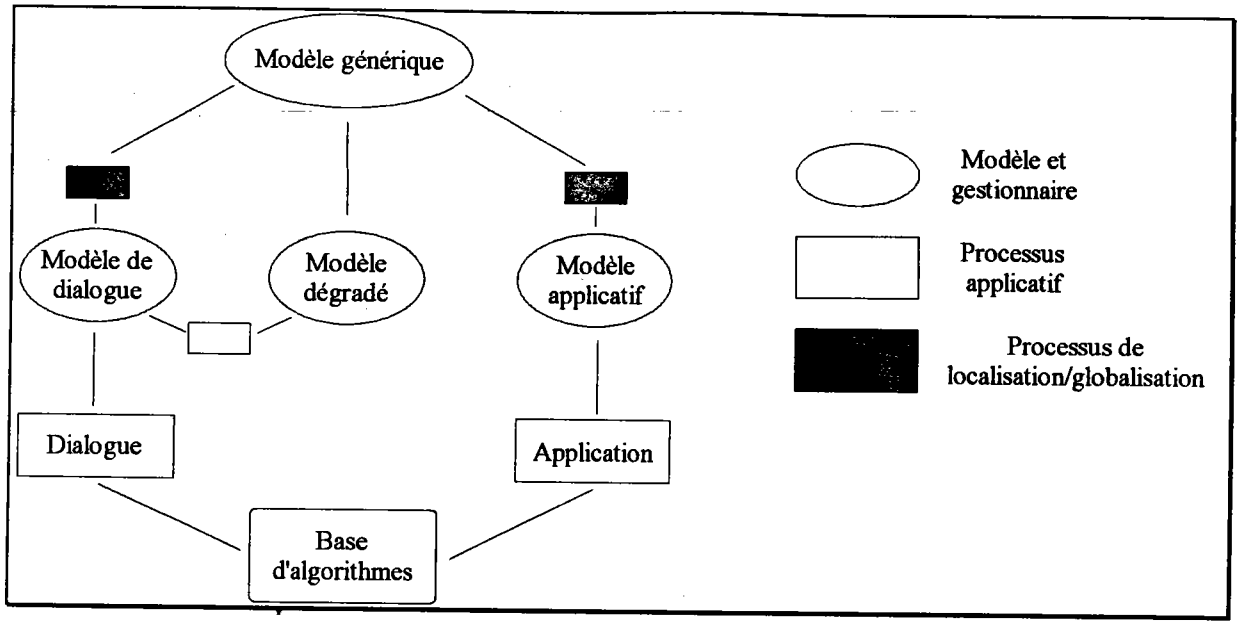


Figure 4 : Le multimodèle MCAO1.

Le modèle générique du système SACADO est basé sur une représentation géométrique du type CSG. Notre objectif est d'en déduire un modèle de visualisation avec comme principale contrainte de conserver les propriétés liées aux arbres CSG, et en particulier, ne pas passer par une approximation. L'utilisation de la non-unicité du modèle CSG pour obtenir des représentations équivalentes a fait l'objet d'un certain nombre de travaux. Nous décrivons dans les paragraphes suivants ceux qui nous ont parus les plus intéressants dans la mesure où leurs approches s'apparentent à la nôtre.

Le modèle DSG (Destructive Solid Geometry) proposé par Perng, Chen et Li [PER 90], est un modèle de fabrication déduit du modèle CSG. L'arbre CSG représentant un objet est transformé pour faire apparaître des caractéristiques de forme qui peuvent être associées à des séquences d'usinage.

Le domaine de représentation est restreint aux objets n'ayant que des faces planes et cylindriques. L'ensemble de volumes primitifs (voir fig. 5) contient de nombreuses redondances destinées à faciliter la représentation d'un objet dans le modèle DSG. On peut remarquer qu'à chaque primitive correspond son complémentaire (c et g, d et h, etc.). Les primitives doivent toutes être orientées parallèlement aux axes d'un même repère. Enfin, les intersections doivent être transformées en différences ($A \cap B = A - \overline{B}$).

Un usinage est une opération qui consiste à enlever de la matière. Elle correspond naturellement à une différence dans le modèle CSG. Le principe revient alors à représenter l'objet par une série de différences. Un arbre DSG est de la forme $P=S-E_1-...-E_m$ où S est la pièce brute et E_i un excès de matière appartenant à l'ensemble des primitives et tels que $\forall i \neq j$, $E_i \cap E_j = \varnothing$.

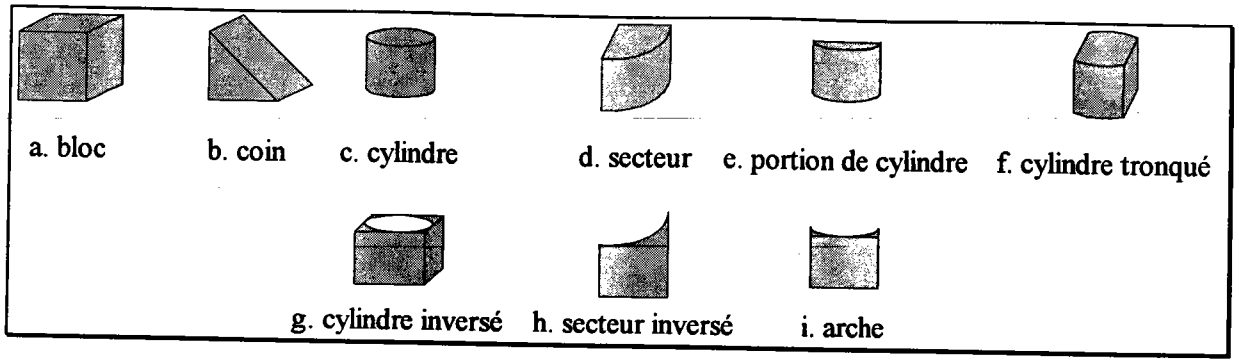


Figure 5 : primitives du modèle DSG.

Nous ne résumons que les principales étapes de la transformation et n'abordons pas les aspects liés à la détection des caractéristiques de forme. Si l'on représente un arbre CSG par une expression parenthésée, alors la première étape consiste à supprimer les parenthèses. Le résultat est une nouvelle expression de la forme : $P=P_0 \pm P_1 \pm ... \pm P_n$ où ± signifie union ou différence. Ensuite, chaque fois qu'une primitive qui n'est pas du type bloc intervient dans une union, elle est remplacée par une différence entre son bloc englobant et sa primitive inverse. Enfin, une union est remplacée par une suite équivalente de différences. Par exemple, $P_i+P_j=B_{i,j}-E_1-...E_k$ où $B_{i,j}$ est le bloc englobant de P_i+P_j et E_k est un excès de matière. La figure 6 illustre ces transformations sur un exemple très simple.

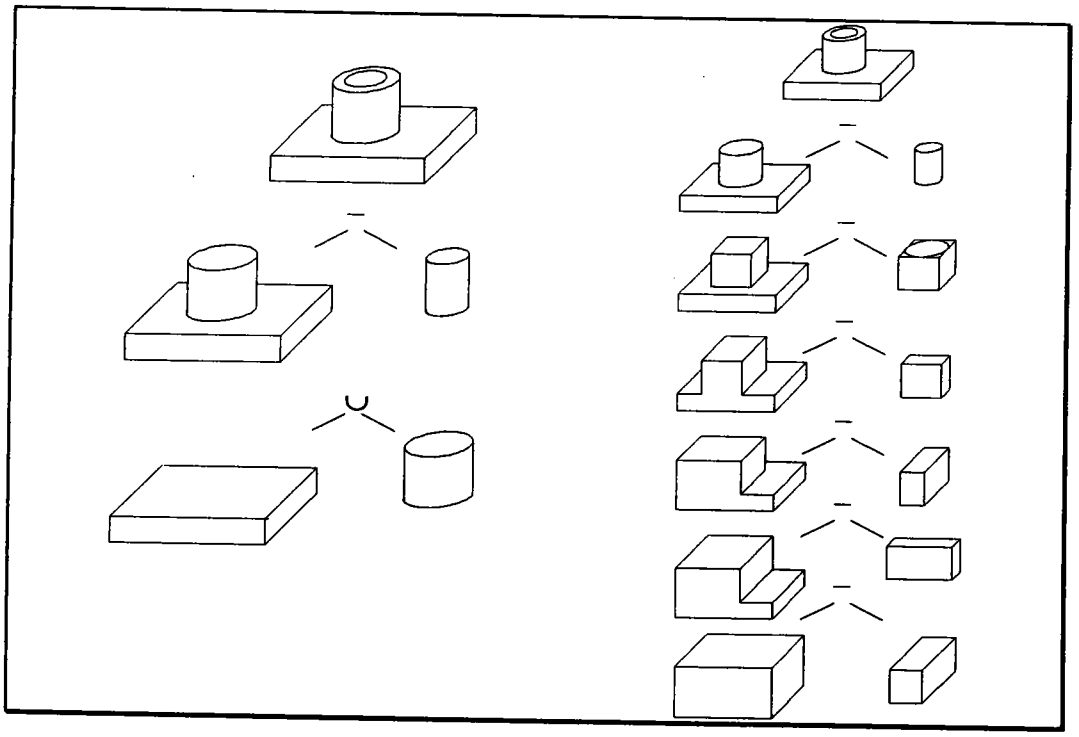


Figure 6 : Enchaînement des opérations dans un arbre CSG et un arbre DSG.

Les arbres DSG illustrent bien l'intérêt de l'approche multimodèle dans un système de CFAO. Lorsque l'opérateur décrit un objet par un arbre CSG, il ne se soucie généralement

que de sa forme et de ses dimensions. Il revient alors à chaque application d'en déduire les informations dont elle a besoin et les exprimer d'une manière plus adéquate. Le modèle DSG est équivalent au modèle CSG, mais les objets sont décrits sous une forme proche des opérations d'usinage, complètement différente de celle de l'opérateur. Un objet n'est plus représenté par les mêmes opérations ni les mêmes primitives.

La construction des arbres DSG repose essentiellement sur la non-unicité géométrique des arbres CSG. Deux ensembles différents de primitives permettent de représenter le même objet. La méthode proposée par Lee et Fu [LEE 87], pour extraire les caractéristiques d'usinage, repose sur la non-unicité mathématique. Les auteurs cherchent à réorganiser l'arbre CSG pour faire apparaître des combinaisons de primitives caractérisant des opérations d'usinage.

Lorsque des primitives interviennent dans une même caractéristique de forme, les axes principaux qui caractérisent ces primitives respectent une certaine cohérence dans leurs orientations. La méthode proposée consiste à les regrouper dans le même sous-arbre. Si ce dernier correspond à une caractéristique de forme type, il est remplacé par un sous-arbre équivalent mais reflétant mieux les opérations d'usinage.

$(A \cup B) \cup C$	$= (C \cup B) \cup A$
$(B \cup A) \cup C$	$= (B \cup C) \cup A$
$C \cup (A \cup B)$	$= A \cup (C \cup B)$
$C \cup (B \cup A)$	$= A \cup (B \cup C)$
$(A-B)-C$	$= A-(B \cup C)$
$(B-A)-C$	$= (B-C)-A$

Figure 7 : exemples d'équivalences permettant de faire remonter le noeud A d'un niveau.

Le point qui nous semble le plus intéressant est l'utilisation d'équivalences pour déplacer un noeud d'un arbre CSG. La figure 7 donne des équivalences qui permettent de faire monter le noeud A d'un niveau dans l'arbre CSG. Plus généralement, un arbre CSG peut être exprimé différemment sans modifier les caractéristiques des primitives. Des règles de transformation permettent de réorganiser l'arbre pour faire émerger les informations nécessaires au traitement envisagé. Dans la démarche qui consiste à rechercher un modèle de visualisation dérivé du modèle CSG, ces deux exemples, les arbres DSG et la méthode de Lee et Fu, donnent des solutions. Ils montrent deux façons différentes de transformer un arbre en gardant l'équivalence avec la représentation initiale. Dans un modèle de fabrication, les caractéristiques recherchées sont connues. Il faut pouvoir associer à la représentation d'un objet les opérations d'usinage qui permettent d'automatiser sa fabrication. Dans le cadre

1. Introduction.

La visualisation est un processus qui comprend deux étapes fondamentales. La première consiste à modéliser la scène. Il s'agit de fournir les informations suffisantes pour que le système chargé de l'affichage puisse produire le résultat voulu. Ces informations comprennent des données géométriques sur les objets (positions, dimensions, etc.) et éventuellement des données concernant leur nature (couleurs, matériaux, etc.). A partir de ce modèle et de paramètres liés à l'image elle-même (position de l'observateur, sources lumineuses, etc.), un algorithme de visualisation détermine, dans la deuxième étape, ce qui doit apparaître à l'écran.

Nous rappelons dans ce premier chapitre, quelques notions essentielles pour mieux connaître les différentes composantes de ce processus de visualisation. Dans le deuxième paragraphe, nous résumons les concepts de base du modèle CSG qui est la représentation de base des scènes que nous souhaitons afficher. Dans le troisième paragraphe, nous passons en revue les principaux algorithmes d'affichage pour ce modèle. Enfin, celui que nous avons retenu pour produire des images réalistes est le lancer de rayons. Nous décrivons dans le quatrième paragraphe ses principales méthodes d'optimisation.

2. Le modèle CSG.

Dans un système de CFAO, le modèle géométrique est le point de départ de nombreux traitements, et en particulier celui de l'affichage. Il contient toutes les informations nécessaires à la description de la forme des objets à représenter. La qualité d'un modèle géométrique peut se mesurer par son aptitude à remplir les conditions suivantes [REQ 80][GAR 91] :

- La validité : toute représentation dans ce modèle correspond-elle à un objet réel et un seul ?
- La puissance : tout objet réel peut-il être représenté dans ce modèle ?
- L'ouverture : peut-on déduire d'autres informations géométriques (volume, enveloppe convexe, ...) ?

Aujourd'hui aucun modèle proposé ne répond correctement à ces trois questions. C'est pourquoi, de plus en plus, les recherches dans ce domaine se tournent vers des multimodèles (l'objet est représenté dans plusieurs modèles) ou des modèles hybrides (la représentation d'un objet comporte des informations partielles de plusieurs modèles). On cumule ainsi les avantages des différents modèles mais les problèmes de cohérence entre les multiples représentations sont difficiles à gérer. Le modèle géométrique choisi par le Laboratoire de

de la visualisation, ces caractéristiques ne sont pas clairement définies. La seule qui soit évidente est que le modèle de visualisation doit permettre de calculer plus vite une image ou tout au moins faciliter la mise en oeuvre des algorithmes. Mais cela n'indique pas quel type d'information rechercher ni quelles transformations faire.

La transformation des arbres CSG a été également étudiée dans le cadre de la visualisation. L'objectif des travaux de Goldfeather, Molnar, Turk et Fuchs [GOL 89] est de diminuer le coût de stockage de l'algorithme du Z-Buffer pour la machine Pixel-Planes. Les auteurs utilisent la notion de normalisation des arbres CSG. En s'appuyant sur les propriétés de la logique booléenne, on peut dire qu'à un arbre CSG correspond une forme normale "disjonctive" équivalente. Normaliser un arbre CSG revient à l'exprimer sous forme d'une réunion de termes ne contenant que des opérateurs Intersection et Différence. Dans le cadre de la mise en oeuvre du Z-Buffer, le fait de ne plus avoir d'opérateur Union simplifie la classification des points dans un terme de la normalisation. Le stockage de l'algorithme ($\log_2 N$ bits par pixel où N est le nombre de primitives [JAN 86]) est ramené à un nombre constant de bits (≤ 128). Les arbres CSG considérés sont binaires. Les transformations géométriques sont concaténées et ramenées au niveau des primitives.

1. $X-(Y \cup Z) = (X-Y)-Z$
2. $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$
3. $X-(Y \cap Z) = (X-Y) \cup (X-Z)$
4. $X \cap (Y \cap Z) = (X \cap Y) \cap Z$
5. $X-(Y-Z) = (X-Y) \cup (X \cap Z)$
6. $X \cap (Y-Z) = (X \cap Y)-Z$
7. $(X \cup Y)-Z = (X-Z) \cup (Y-Z)$
8. $(X \cup Y) \cap Z = (X \cap Z) \cup (Y \cap Z)$

Figure 8 : règles de transformation.

L'algorithme figure 9 normalise un arbre CSG à partir de huit règles de transformation (voir fig. 8). Ces règles ont été choisies car elles représentent toutes les expressions booléennes non normalisées. Le principe de l'algorithme est qu'un arbre est normalisé si aucune des huit règles ne peut s'appliquer. L'arbre transformé est un arbre binaire où les opérateurs Union sont regroupés au sommet. Dans un parcours en profondeur de cet arbre, lorsqu'un noeud contient une Intersection ou une Différence alors c'est la racine d'un terme ne contenant plus d'opérateurs Union (voir fig. 10).

```
PROCEDURE Normalise( VAR T : ArbreCSG )
Début
        Si T≠Primitive Alors
          Repéter
            TantQue T correspond au membre de gauche d'une des 8 règles Faire
              Remplacer par le membre de droite
            FTant

            Normalise(FilsGauche(T))
          Jusqu'à ce que (T=∪) ou (FilsDroit(T)=Primitive et FilsGauche(T)≠∪)

            Normalise(FilsDroit(T))
        Fsi
Fin
```

Figure 9 : algorithme de normalisation.

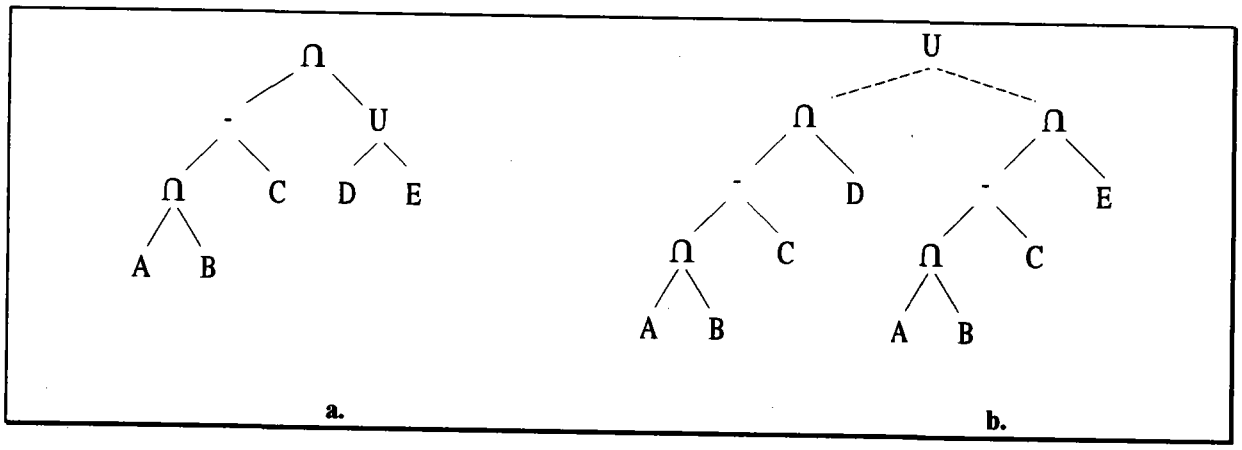


Figure 10 : a) arbre CSG classique b) arbre CSG normalisé contenant deux termes.

Si l'on observe ces règles, cinq d'entre elles (2,3,5,7 et 8) dupliquent un membre de l'expression booléenne. Certains cas défavorables conduisent à un accroissement exponentiel du nombre de noeuds. En se basant sur les résultats de la logique booléenne, la normalisation d'un arbre CSG produit au plus 2^n termes, où n est le nombre de primitives. Bien que respectant la logique booléenne, ces transformations peuvent engendrer des expressions géométriquement vides donc sans intérêt. Par exemple, bien que l'arbre X∩(Y∪Z) ne soit pas vide, le nouveau sous arbre X∩Z produit par la règle 2 peut très bien l'être. La normalisation s'accompagne donc d'une phase de simplification. La méthode utilisée par les auteurs consiste à évaluer les englobants parallélépipédiques et à élaguer au fur et à mesure les branches vides

(voir fig. 11). Cela peut conduire à la suppression d'un ou plusieurs termes de la forme normale.

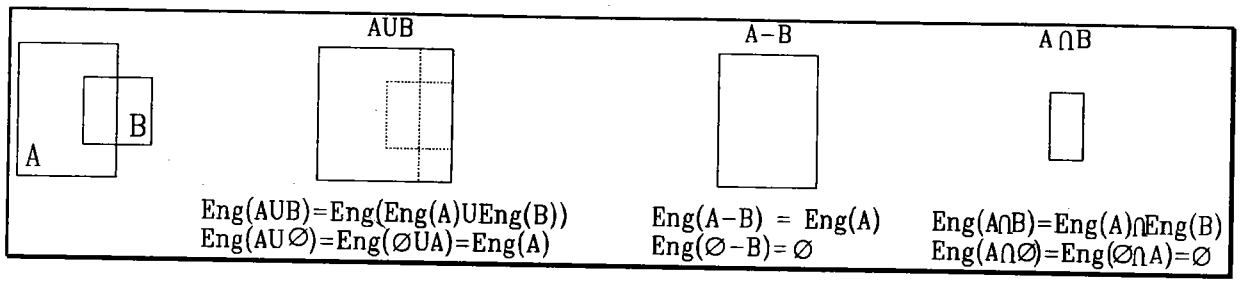


Figure 11 : simplification d'un arbre CSG.

Les termes de la normalisation sont combinés par des unions pour constituer l'objet final. Cet opérateur étant commutatif, l'ordre dans lequel sont considérés les termes n'influence pas la forme de l'objet. Il peut donc être modifié pour répondre au mieux aux besoins des traitements.

Dans le cadre de la visualisation, les termes de la normalisation peuvent être dissociés. En effet, l'image d'une union d'arbres CSG est équivalente à l'union (ou la composition) des images des arbres CSG en respectant un ordre décroissant dans la profondeur en chaque pixel. La normalisation des objets permet d'obtenir un autre niveau de représentation de la scène. Un terme correspond à une partie de l'objet et est généralement de taille plus réduite. Puisque c'est aussi un arbre CSG, les algorithmes classiques de visualisation (cf. chap.1 §3) s'appliquent de la même façon. La normalisation constitue la base du modèle de visualisation que nous proposons.

3. Le modèle VSG (Visual Solid Geometry).

3.1. Construction des arbres VSG.

La première étape dans la construction du modèle de visualisation est de normaliser l'arbre CSG en utilisant l'algorithme de la figure 9 et les règles de la figure 8. Si l'on observe la forme des termes de la normalisation (voir fig. 10), on remarque que ce sont des arbres dégénérés. Tous les fils gauches, sauf le dernier qui est une primitive, sont soit des intersections soit des différences, et tous les fils droits sont des primitives. Avant de démontrer cette propriété remarquable, on se fixe quelques notations :

- on appelle $\mathcal{P}$ l'ensemble des primitives {Cube,Sphère,Cylindre,Cône, Tore, etc.}.
- on appelle $\mathcal{O}$ l'ensemble des opérations booléennes $\{\cup,\cap,-\}$.
- on appelle $\mathcal{A}$ l'ensemble des arbres CSG

$\mathcal{A} = \mathcal{P} \cup \{(Fg\ op\ Fd)$ où op$\in \mathcal{O}$ et Fg,Fd$\in \mathcal{A}\}$.

- on appelle $\mathcal{J}$ l'ensemble des arbres construits à partir des seuls opérateurs $\cap$ et -

$\mathcal{J} = \mathcal{P} \cup \{(Fg\ op\ Fd)$ où op$\in \mathcal{O}-\{\cup\}$ et Fg,Fd$\in \mathcal{J}\}$.

Propriété 1 : Un terme d'un arbre CSG normalisé par l'algorithme de la figure 9 est dégénéré, c'est-à-dire que tous les fils droits sont des primitives. On démontre cette propriété par induction.

Hypothèse : soit T$\in \mathcal{J}$ et par construction aucune des huit règles ne peut s'appliquer à T.

1. T$\in \mathcal{P}$ alors la propriété est vérifiée

2. $\exists$T1,T2$\in \mathcal{J}$, $\exists$op$\in \mathcal{O}-\{\cup\}$ tels que T=T1 op T2

   2.1. T2$\in \mathcal{P}$ alors la propriété est vérifiée

   2.2. $\exists$X,Y$\in \mathcal{J}$, $\exists$op'$\in \mathcal{O}-\{\cup\}$ tels que T2=X op Y
   donc T=T1 op (X op' Y)
   dans ce cas l'une des quatre règles 3,4,5 ou 6 peut s'appliquer. Ce qui est contradictoire avec l'hypothèse

Conclusion : si T$\in \mathcal{J}$ alors soit T$\in \mathcal{P}$, soit T=T1 op T2 où T1$\in \mathcal{J}$, T2$\in \mathcal{P}$, op$\in \mathcal{O}-\{\cup\}$. Les fils droits sont des primitives donc les termes sont dégénérés.

Puisque les termes ont cette structure particulière, ils ne peuvent être construits qu'à partir de quatre expressions booléennes :

A. (X-Y)-Z
B. (X-Y)$\cap$Z
C. (X$\cap$Y)-Z
D. (X$\cap$Y)$\cap$Z

Or l'expression (X-Y)$\cap$Z (forme B) est équivalente à (X$\cap$Z)-Y (forme C). Deux noeuds peuvent être permutés dans un sens ou dans l'autre. D'où l'idée de regrouper d'un côté les opérateurs d'intersection et de l'autre les opérateurs de différence (voir fig 12 b.). On en déduit les quatre nouvelles règles suivantes :

A'. $(X-Y)-Z = X-(Y \cup Z)$

B'. $(X-Y) \cap Z = (X \cap Z)-Y$

C'. $(X \cap Y) \cap Z = \cap(X,Y,Z)$

D'. $(X \cup Y) \cup Z = \cup(X,Y,Z)$

Les règles C' et D' permettent de faire apparaître l'union et l'intersection n-aire, ce qui réduit la hauteur de l'arbre et permet d'avoir une écriture plus concise (voir fig. 12 c.).

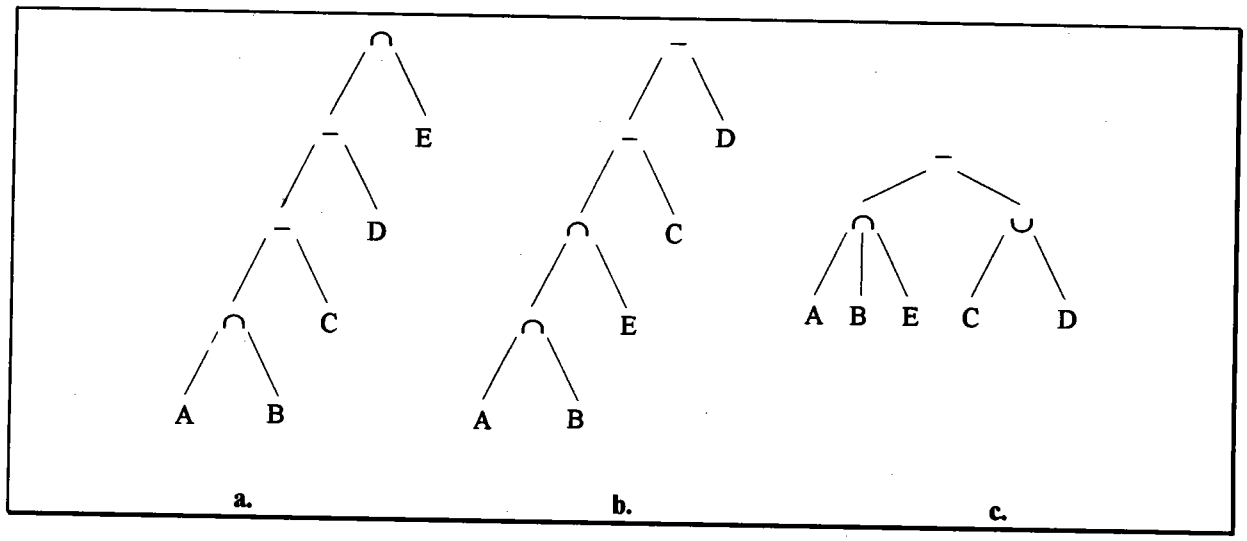


Figure 12 : a) terme de la forme normale
b) permutation des opérateurs d'intersection et de différence
c) arbres VSG.

Grâce à ces quatre règles, on peut démontrer que les termes d'une normalisation peuvent toujours s'écrire sous la forme de ce que nous baptisons arbres VSG.

Propriété 2 : un terme dégénéré T_n de hauteur n, donc composé de n+1 primitives, peut être transformé grâce aux quatre règles ci-dessus sous la forme :

$$T_n = \cap(P_1,\ldots,P_i)-\cup(P_{i+1},\ldots,P_{n+1}) \text{ où } Pj \in \mathcal{P}, 1 \leq j \leq n+1$$

Rq : dans cette expression, l'ordre croissant des indices n'a aucun rapport avec leur situation dans l'arbre CSG.

Nous démontrons cette propriété par récurrence.

Au rang 0 : $T_0 \in \mathcal{P}$ donc la propriété est vérifiée.

Au rang n : on suppose la propriété vérifiée donc $T_n = \cap(P_1,...,P_i)-\cup(P_{i+1},...,P_{n+1})$

Au rang n+1 : soit $P_{n+2} \in \mathcal{P}$

1. $T_{n+1} = T_n - P_{n+2}$
   $T_{n+1} = [\cap(P_1,...,P_i)-\cup(P_{i+1},...,P_{n+1})] - P_{n+2}$
   $T_{n+1} = \cap(P_1,...,P_i)-[\cup(P_{i+1},...,P_{n+1}) \cup P_{n+2}]$ Règle A'
   $T_{n+1} = \cap(P_1,...,P_i)-\cup(P_{i+1},...,P_{n+1},P_{n+2})$ Règle D'

   Donc la propriété est vraie

2. $T_{n+1} = T_n \cap P_{n+2}$
   $T_{n+1} = [\cap(P_1,...,P_i)-\cup(P_{i+1},...,P_{n+1})] \cap P_{n+2}$
   $T_{n+1} = [\cap(P_1,...,P_i) \cap P_{n+2}]-\cup(P_{i+1},...,P_{n+1})$ Règle B'
   $T_{n+1} = \cap(P_1,...,P_i,P_{n+2})-\cup(P_{i+1},...,P_{n+1})$ Règle C'

   Donc la propriété est également vérifiée

La propriété est vraie au rang n+1 donc elle est vraie quel que soit n

Cet arbre particulier est facile à représenter. Il est entièrement défini par deux listes de primitives. Si la liste des primitives de l'union est vide, l'arbre est de la forme $\cap(P_1,...,P_{n+1})$ sinon il est de la forme $\cap(P_1,...,P_i)-\cup(P_{i+1},...,P_{n+1})$. Il n'est donc pas utile de représenter explicitement les opérateurs booléens. Ils se déduisent automatiquement du contenu de la liste de l'opérateur union.

Les deux listes de primitives sont en plus très faciles à construire. Dans le terme initial, lorsqu'une primitive est le fils droit d'une intersection, elle appartient forcement à la liste de l'intersection n-aire (cas 2. de la démonstration). Si elle est le fils droit d'une différence alors elle appartient à la liste de l'union n-aire (cas 1. de la démonstration). La seule primitive qui soit le fils gauche d'une opération booléenne appartient à la liste de l'intersection n-aire (rang 0 de la démonstration). On en déduit l'algorithme de construction de l'arbre VSG d'un terme de la forme normale :

```
Procédure ConstructionVSG( T; VAR L_inter,L_union )
Début
   Si T=∅ Alors
         L_inter = ∅
         L_union = ∅
```

```
    Sinonsi T∈ ℙ Alors
        L_inter = T
        L_union = ∅
    Sinon
        ConstructionVSG( FilsGauche(T), L_inter, L_union )
        Cas T de
            Intersection    : L_inter = L_inter+FilsDroit(T)
            Différence      : L_union = L_union + FilsDroit(T)
        FCas
    FSi
Fin
```

La nature linéaire de la représentation des arbres VSG nécessite la remise en cause de la plupart des algorithmes récursifs conçus pour les arbres CSG. La procédure d'intersection entre un rayon et un arbre VSG en particulier peut être réécrite de manière complètement itérative. L'algorithme est le suivant :

```
RayCasting(Rayon, ArbreVSG; VAR ListeSegments)
Début
    ListeSegments=∅
    PasInter=Faux {PasInter est vrai lorsqu'il n'y a pas d'intersection avec l'arbre VSG }

    LI=ListeIntersection(ArbreVSG) {Liste des primitives de l'intersection n-aire}
    Tantque LI≠∅ et Non PasInter Faire
        CalculIntersectionPrimitive(Rayon,Premier(LI),LS)
        ListeSegments=Combine(Intersection,LS,ListeSegments)
        Si ListeSegments=∅ Alors
            PasInter=Vrai
        FSi
        LI=Suivant(LI)
    FTant

    LU = ListeUnion(ArbreVSG) {Liste des primitives de l'union n-aire}
    Tantque LU≠∅ et Non PasInter Faire
        CalculIntersectionPrimitive(Rayon,Premier(LU),LS)
        ListeSegments=Combine(Différence,LS,ListeSegments)
        Si ListeSegments=∅ ALors
            Pas Inter=Vrai
```

FSi
LU=Suivant(LU)
FTant
Fin

La construction de l'englobant d'un arbre VSG est facilitée par sa forme particulière. D'après les règles de la figure 11, l'englobant est égal à celui fils gauche, c'est-à-dire de l'intersection n-aire : $Eng(\cap(P_1,...,P_i)-\cup(P_{i+1},...,P_{n+1}))=Eng(\cap(P_1,...,P_i))$. Il se calcule de manière itérative puisque $Eng(\cap(P_1,...,P_i))=Eng(P_1)\cap...\cap Eng(P_i)$. La liste de l'union peut être facilement simplifiée. Si l'englobant d'une primitive ne coupe pas l'englobant de l'arbre VSG, alors elle n'intervient pas dans la forme du sous-objet correspondant et peut être supprimée de la liste.

3.2. Le test d'appartenance.

Nous venons de montrer qu'un arbre VSG a une forme constante entièrement définie par deux listes de primitives. La première, correspondant à l'intersection n-aire, décrit la forme "brute" du sous-objet représenté. Selon la nature géométrique des primitives mises en oeuvre (concaves ou convexes), le brut peut être constitué d'une ou plusieurs composantes connexes. La deuxième liste de primitives, correspondant à l'union n-aire, représente les "perçages" à effectuer dans le brut. Le mot "brut" n'est pas employé ici dans le sens habituel de la fabrication. Il désigne simplement la forme décrite par l'arbre VSG mais sans les perçages.

La représentation d'un objet telle qu'elle est décrite par l'opérateur ne tient pas forcément compte de la topologie, mais d'autres contraintes comme la géométrie de l'objet, la concision de la représentation ou encore l'expérience de l'opérateur. Ces informations topologiques existent mais sont dispersées dans l'arbre CSG et ne sont plus exploitables. Ce sont pourtant des indications essentielles dans la résolution de certains problèmes. Les travaux menés sur la transformation des arbres CSG pour la détection et l'extraction des caractéristiques de forme en sont l'illustration. Le fait qu'un arbre VSG représente un brut percé d'un certain nombre de trous est une information topologique importante. De plus elle fait partie des caractéristiques intrinsèques de cette représentation.

Dans le cadre de la visualisation, le lancer de rayons est utilisé pour déterminer le point visible le plus proche de l'origine du rayon. Si l'on recherche l'intersection entre un rayon et un objet VSG, l'algorithme itératif décrit dans le paragraphe précédent convient évidemment mais ne tient pas compte du tout de cette information topologique. Or si le rayon coupe le brut en un point P, alors P est le point visible s'il n'appartient à aucun des trous. S'il appartient à un trou, il faut calculer l'intersection du rayon avec ce dernier et évaluer l'opération booléenne pour déterminer un nouveau point du brut P'. En réitérant ce processus on finira soit par traverser totalement le brut, soit par trouver un point qui n'appartient à aucun autre trou. Nous venons de faire appel à nouvel outil de décision : le test d'appartenance. Le test d'appartenance

d'un point à une primitive est la propriété qui est vraie lorsque le point est dans le volume décrit par la primitive ou se situe sur sa frontière. Par exemple, un point appartient à une sphère si la distance de ce point au centre de la sphère est inférieure ou égale au rayon de la sphère. Cette technique n'est intéressante évidemment que si elle est moins coûteuse qu'un calcul d'intersection.

Afin de vérifier cette hypothèse, nous avons comparé les procédures d'intersection et de test d'appartenance. L'origine et la direction des rayons sont calculées de manière aléatoire. L'origine du rayon est également utilisée pour tester l'appartenance d'un point à une primitive. Les tests ont été réalisés de manière à ce que le temps lié à l'initialisation des données soit le même quel que soit le type de procédure. Soit $\Delta t = n\Delta init + n\Delta proc$ où $\Delta init$ représente le temps consacré à l'initialisation d'un rayon et $\Delta proc$ représente le coût d'un appel à la procédure testée. Donc $n\Delta proc = \Delta t - n\Delta init$. Le tableau 2 donne les temps CPU en seconde de $n\Delta proc$ obtenus sur une station de travail SUN Sparc 2 pour n=100 000.

	Cube	Sphère	Cylindre	Cône	Tore
Intersection	1.07	0.89	1.29	1.26	1.56
Appartenance	0.25	0.25	0.49	0.42	0.25
	≈×4	≈×4	≈×3	≈×3	≈×6

Tableau 1 : comparaison des procédures d'intersection rayon/primitives et de test d'appartenance point/primitive.

Certes, la fiabilité de ces tests est discutable et pour plusieurs raisons. Tout d'abord, les procédures d'intersection testées (cf. Annexes A) ne sont peut-être pas les plus performantes. La façon dont elles sont programmées, l'ordre des instructions ou le compilateur ont une influence qui n'est pas négligeable. Enfin et surtout, les temps obtenus sont étroitement liés aux caractéristiques du matériel. La présence d'unités mathématiques spécialisées ou certaines architectures peuvent modifier ces temps de calcul et perturber le rapport exact entre les deux types de procédure. Par exemple, le fait que lors des tests la même opération soit répétée successivement un grand nombre de fois peut tirer profit d'opérateurs "pipeline" alors qu'en conditions réelles ceux-ci sont inopérants ou beaucoup moins. Néanmoins, les chiffres indiqués expriment un ordre de grandeur et prouvent que le test d'appartenance est dans tous les cas plus rapide qu'un calcul d'intersection. Nous recherchons donc à utiliser ce test plutôt que de calculer des intersections avec les primitives.

Nous décrivons plus en détails un algorithme de lancer de rayons utilisant le test d'appartenance pour diminuer le nombre de calculs d'intersection avec les trous. Ensuite, nous montrons qu'il peut, au moins en théorie, être utilisé pour diminuer le nombre de calculs d'intersections dans la recherche du point visible sur le brut.

3.2.1. Traitement des trous.

Lorsqu'un arbre est de la forme de la figure 13, et que B,C,D et E sont distincts, alors un point quelconque de A ne peut se trouver au plus que dans un seul de ces trous. La procédure classique ne peut déterminer le point d'intersection visible qu'après avoir calculé toutes les intersections et les avoir combinées selon les opérations booléennes. Or puisqu'un point ne peut appartenir qu'à un seul trou , un test d'appartenance éliminera naturellement au moins 3 des 4 possibilités et réduit donc le nombre de calculs d'intersections à 2 au lieu de 5.

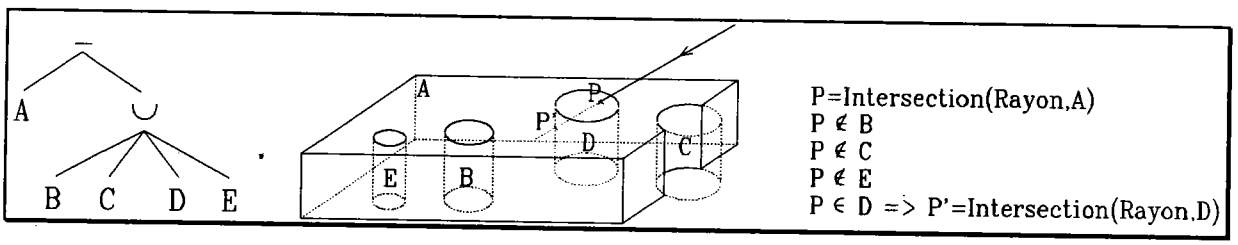


Figure 13: appartenance d'un point à un trou.

Si ces trous ne sont pas distincts, le principe est le même mais il peut y avoir plusieurs calculs d'intersection. Après avoir calculé l'intersection du rayon avec un trou, il est nécessaire de reconsidérer l'ensemble des autres trous qui peuvent le recouper (voir fig. 14).

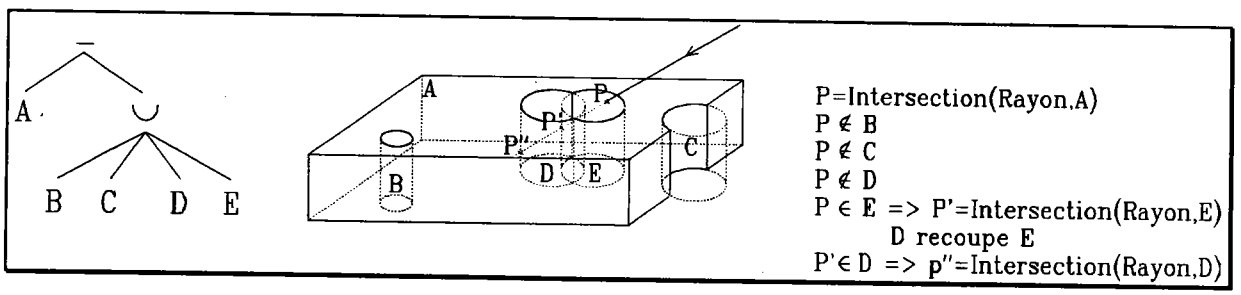


Figure 14 : trous non distincts.

On en déduit l'algorithme suivant :

Procédure IntersectionTrous(Rayon, O1..On; **VAR** ListeSegments)
Début

{O1,..,On représente la liste des trous c'est-à-dire la liste des primitives de l'union n-aire}
{ ListeSegments contient initialement le résultat de l'intersection entre le rayon et le brut}
{ A l'issue de cette procédure, la première extremité de ListeSegments est le point visible}
P = PointPlusProche(ListeSegments)
i=1

```
Trouve=Faux
Tant que ListeSegments≠∅ et non Trouvé et i≤n Faire
        Si Appartient(P,Oi) Alors
                {Le point appartient à un trou}
                CalculIntersectionPrimitive(Rayon,Oi,LSi)
                ListeSegments=Combine(ListeSegments,LSi)
                P = PointPlusProche(ListeSegments)
                Trou = Oi
                {Il faut vérifier que le nouveau point n'appartient pas à un autre trou}
                Tant qu'il existe Oj recoupant Trou et Appartient(P,Oj) et
                ListeSegments≠∅ Faire
                        CalculIntersectionPrimitive(Rayon,Oj,LSj)
                        ListeSegments=Combine(ListeSegments,LSj)
                        P = PointPlusProche(ListeSegments)
                        Trou=Oj
                Ftant
                Trouvé=Vrai
        Sinon
                i=i+1
        Fsi
    Ftant
Fin
```

3.2.2. Traitement du brut.

Considérons la liste des primitives de l'intersection n-aire. Un point quelconque d'une primitive appartient au brut s'il appartient à toutes les autres primitives. C'est la définition même de l'opérateur d'intersection.

Lorsqu'un rayon est lancé, les points recherchés sont ceux de la surface des primitives qui respectent cette propriété. Après avoir calculé l'intersection entre le rayon et la première primitive, on regarde si le point trouvé appartient aux autres primitives. S'il n'appartient pas à l'une d'entre elles, l'intersection du rayon avec cette primitive est calculée et combinée avec l'intersection précédente. Le processus se poursuit avec le nouveau point et s'arrête lorsque toutes les primitives ont été étudiées ou lorsque l'intersection est vide.

Cet algorithme permet de déterminer le point d'intersection visible du brut pour un rayon donné. L'évaluation booléenne nécessaire dans le traitement des trous exige la connaissance de tous les points délimitant les segments d'intersection entre le rayon et le brut. On ne peut pas s'arrêter à la seule recherche du point visible.

Lorsqu'un rayon traverse une primitive convexe, il ne produit qu'un seul segment. Donc, lorsqu'il traverse une intersection de primitives convexes, il ne produit également qu'un seul segment. Le processus décrit précédemment détermine l'extrémité de ce segment la plus proche de l'origine du rayon. En l'appliquant de manière inverse on obtient l'extrémité la plus éloignée. Lorsque les primitives sont concaves, leur intersection peut être composée de plusieurs composantes connexes. Appliquer le même principe de recherche n'est plus possible puisqu'il est nécessaire de connaître également les points intermédiaires en plus des deux extrémités.

L'algorithme détaillé pour déterminer le segment d'intersection entre un rayon et un brut formé de primitives convexes en utilisant le test d'appartenance est le suivant :

```
Procédure IntersectionBrut( Rayon; O1...On; VAR ListeSegments )
Début
    {O1,..,On représente la liste des primitives de l'intersection n-aire}
    { A l'issue de cette procédure, ListeSegments contient le segment d'intersection entre le rayon
    et le brut}

    { Recherche de la première extrémité du segment}
    CalculIntersectionPrimitive( Rayon, O1, ListeSegments )
    P = PointPlusProche( ListeSegments )
    ListeEtudiée = ∅
    i = 2
    Tantque ListeSegments≠∅ Et i≤n-1 Faire
        Si Non Appartient(P,Oi) Alors
            CalculIntersectionPrimitive( Rayon, Oi, LSi )
            ListeSegments = Combine( ListeSegments, LSi )
            P = PointPlusProche( ListeSegments )
        Sinon
            ListeEtudiée = ListeEtudiée + Oi
        Fsi
    FTant

    { Recherche de la deuxième extrémité du segment}
    {ListeEtudiée = O'1...O'm}
    CalculIntersectionPrimitive( Rayon, On, LSj )
    ListeSegments = Combine( ListeSegments, LSj )
    Q = PointPlusEloigne( ListeSegments )
    j = m
```

```
Tantque ListeSegments≠∅ Et j≥1 Faire
    Si Non Appartient(Q,O'j) Alors
        CalculIntersectionPrimitive( Rayon, Oj, LSj )
        ListeSegments = Combine( ListeSegments, LSj )
        Q = PointPlusEloigné( ListeSegments )
    FSi
FTant
Fin
```

Contrairement au traitement des trous, on n'est pas sûr de diminuer le nombre de calculs d'intersection par rapport à la méthode classique qui consiste à les calculer systématiquement toutes. Si l'on suppose que pour P primitives, N est le nombre d'intersections calculées dans la boucle de la première étape et M celui de la deuxième étape, on cherche à maximiser N+M pour déterminer à partir de quand cet algorithme n'est plus rentable. On considère également qu'une intersection vaut en moyenne 4 tests d'appartenance.

Méthode classique :

P intersections soit 4P tests

Algorithme proposé :

Etape 1 : (N+1) intersections et (P-2) tests soit 4(N+1)+(P-2) tests

Etape 2 : (M+1) intersections et (P-2-N) tests soit 4(M+1)+(P-2-N) tests

On recherche donc :

$4(N+1)+(P-2)+4(M+1)+(P-2-N) \leq 4P$

$3N+4M+2P+4 \leq 4P$

Soit : $3N+4M \leq 2P-4$

On majore $3N+4M$ par $4(N+M)$

Donc: $4(N+M) \leq 2(P-2)$

$$N+M \leq \frac{1}{2}(P-2)$$

Cette limite est relativement faible pour un petit nombre de primitives. Si P=3 alors N+M doit être égal à 0 pour que la méthode soit rentable. Ce qui signifie que puisque pour deux des trois primitives on recherche d'office une intersection, la troisième ne doit pas être étudiée. Or, l'expérience montre que dans la construction des arbres VSG, il est très rare d'obtenir des bruts composés de plus de trois primitives. Cette proposition n'est donc pratiquement pas rentable.

La prise en compte des primitives concaves est relativement simple. Il suffit de traiter d'abord toutes les primitives convexes avec cet algorithme puis de revenir à la méthode classique pour traiter les primitives concaves. L'opérateur d'intersection est commutatif. L'ordre de traitement des primitives n'a pas d'influence sur la géométrie du brut. La liste des primitives peut donc être réarrangée en fonction des convenances.

On peut remarquer également que l'on a intérêt à commencer par étudier la primitive la plus éloignée de l'origine du rayon (voir fig. 15). Le premier point d'intersection a alors plus de chance d'appartenir aux autres primitives. La recherche d'un classement des primitives favorables pour chaque rayon n'est pas réaliste. Le coût d'une réorganisation de la liste serait trop élevé pour les gains de performance espérés. Une solution consisterait à trier cette liste lorsque de nombreux rayons partent du même point dans une direction cohérente (rayons primaires, d'ombrage et certains rayons secondaires). Ce classement semble néanmoins très difficile car les primitives sont du fait de l'intersection, enchevêtrées les unes dans les autres.

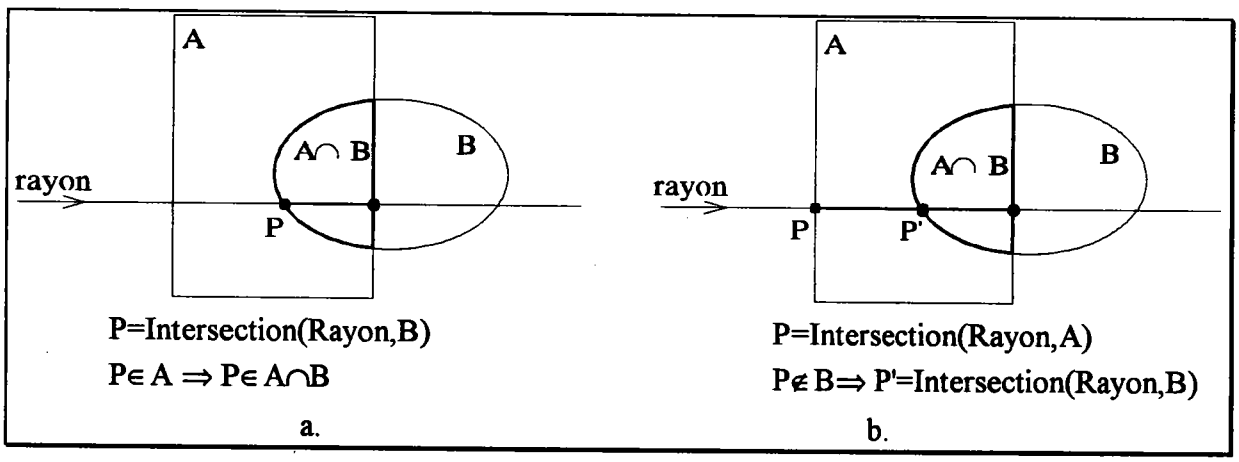


Figure 15 : influence de l'ordre dans le traitement du brut.

L'application du test d'appartenance est en théorie possible pour le traitement du brut mais les limites sont telles qu'il ne présente aucun intérêt en pratique. Nous ne l'avons donc pas utilisé dans ce cadre.

3.3. Arbres VSG et optimisations du lancer de rayons.

Il s'agit dans ce paragraphe de montrer que les arbres VSG possèdent en plus de celles montrées précédemment, un certain nombre de propriétés qui, soit incluent certaines optimisations spécifiques aux arbres CSG, soit en simplifient la mise en oeuvre.

3.3.1. Les englobants.

On a montré que l'englobant d'un arbre VSG est égal à l'intersection des englobants des primitives qui composent le brut. Il en découle une construction itérative simple et rapide.

Dans un arbre CSG, à chaque noeud est associé un englobant. Pour un arbre VSG un seul englobant suffit. Il n'est même pas nécessaire de conserver les englobants des primitives. L'englobant de l'arbre est par construction inclus dans l'englobant de chacune des primitives du brut. Le test d'appartenance quant à lui, détermine plus efficacement le trou à étudier qu'un calcul d'intersection rayon/englobant et est moins coûteux.

Nous avons décrit dans le chapitre 1 (cf. §4.2.2.1.) la méthode de raffinement des englobants proposée par Cameron. Le principe repose sur deux théorèmes appelés théorème de descente et théorème de montée. En appliquant ces théorèmes successivement un certain nombre de fois (au plus n où n est le nombre de primitives de l'arbre CSG), les englobants des noeuds convergent vers les englobants minimaux.

Si l'on applique ce raffinement à l'englobant d'arbre VSG, on obtient :

Soit $T=\cap P_i-\cup P'_j$ un arbre VSG. D'après le théorème de montée (cas c) on a :

$Eng(T)=Eng(T)\cap Eng(\cap P_i)$ or par construction $Eng(T)=Eng(\cap P_i)$
donc Eng(T) est invariant

Si l'on applique le théorème de descente on a :

$Eng(\cap P_i)=Eng(\cap P_i)\cap Eng(T)$
donc $Eng(\cap P_i)$ est également invariant.

L'englobant d'un arbre VSG est minimal puisqu'il est invariant pour les deux théorèmes. Les englobants des primitives pourraient être optimisés par cette méthode, mais on vient de montrer qu'il n'était pas nécessaire de les conserver. Cette propriété est très importante. La qualité des englobants conditionne les performances d'un grand nombre d'opérations notamment en synthèse d'images. Le fait que l'englobant d'un arbre VSG soit minimal, est donc un avantage considérable par rapport aux arbres CSG. On peut noter que cette propriété qui se démontre facilement avec les arbre VSG provient de la normalisation. En effet, les termes étant, d'une part, constitués uniquement d'opérateurs Intersection et Différence et étant dégénérés d'autre part, l'englobant d'un terme est construit à partir des mêmes primitives.

3.3.2. L'automate d'intersection.

Considérons un arbre VSG $T=\cap P_i-\cup P'_j$. Par construction, toutes les primitives P_i sont connectées entre elles puisqu'elles interviennent dans l'intersection n-aire. De même, les primitives P'_j sont toutes connectées aux primitives P_i. La figure 16 illustre le graphe de connexions d'un arbre VSG.

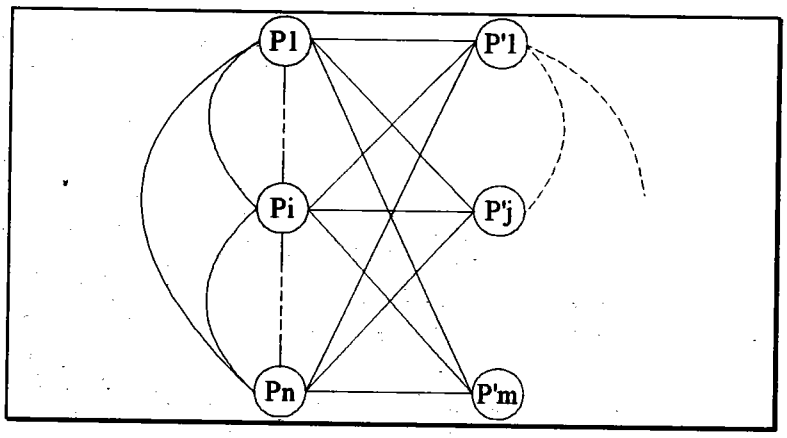


Figure 16 : Graphe de connexions d'un arbre VSG.

Si l'on considère la procédure d'intersection itérative (cf. §3.1.) l'automate d'intersection (voir fig. 17 a.) est linéaire. Les primitives sont étudiées les unes après les autres. Du fait de l'organisation de l'arbre VSG, le graphe de connexions ne permet pas de réduire le nombre de primitives étudiées. En revanche, si l'on considère le test d'appartenance, les seuls trous considérés sont ceux traversés par le rayon. On peut noter également que lorsqu'un trou est étudié, il est forcément coupé par le rayon (voir fig. 17 b.).

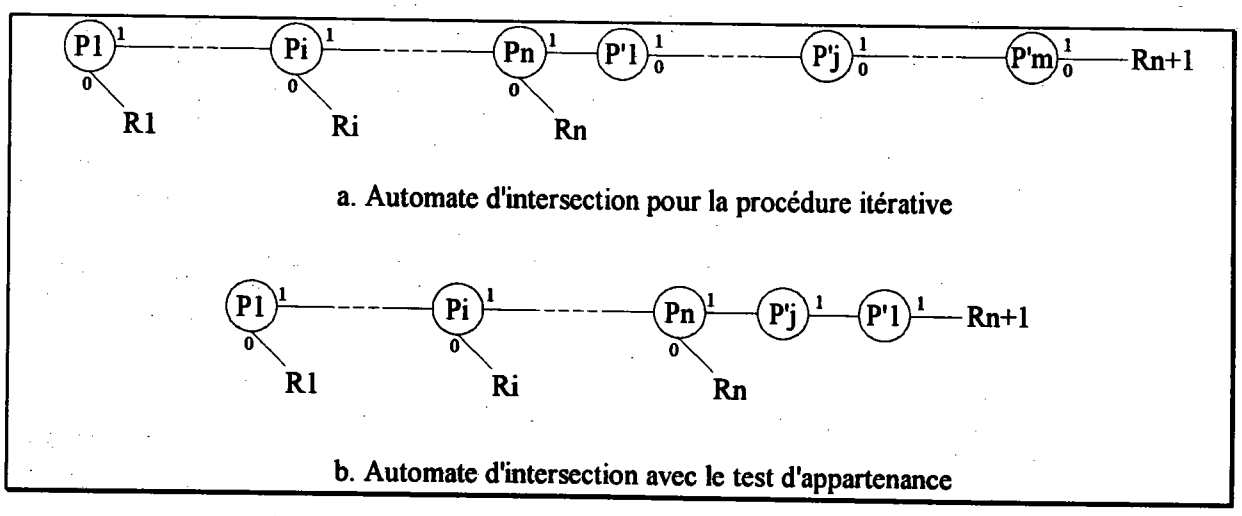


Figure 17 : Automates d'intersection pour un arbre VSG.

Les arbres VSG incluent naturellement l'optimisation par automate d'intersection puisque les primitives sont étudiées séquentiellement et que le graphe de connexions n'a pas d'utilité. De plus, l'utilisation du test d'appartenance est nettement plus efficace car il permet de déterminer exactement les primitives devant être coupées.

3.3.3. Les zones actives.

La zone active (cf. chap. 1 §3.2.3.2.) d'un noeud N d'un arbre CSG A est donnée par la différence : $Z_A(N)=I_A(N)-U_A(N)$. Si l'on considère un arbre VSG $T=\bigcap_{i=1}^{n} Pi - \bigcup_{j=1}^{m} P'j$, on peut appliquer les règles de construction des I-zones et des U-zones pour chacun des noeuds. La figure 18 montre leur construction. On remarque que pour tous les noeuds, les U-zones sont vides. Par conséquent la zone active d'un noeud est égale à la I-zone : $Z_A(N)=I_A(N)$.

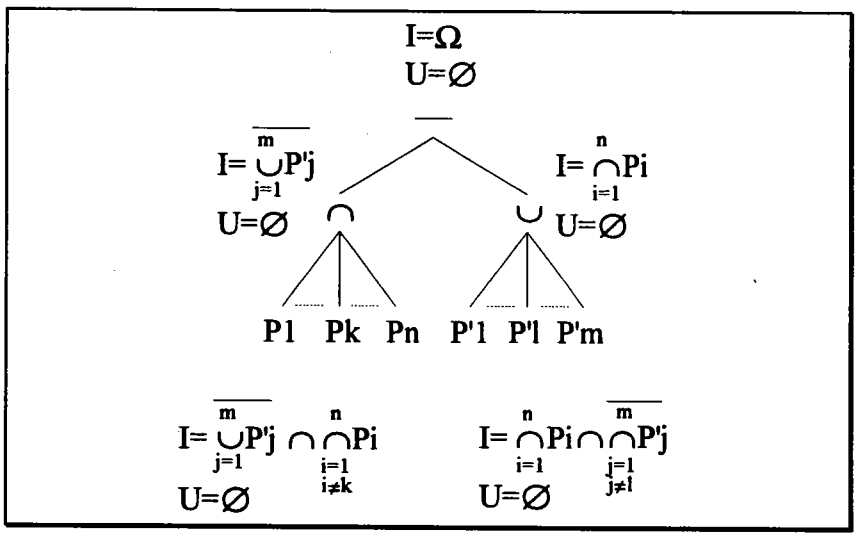


Figure 18 : Construction des I-zones et des U-zones pour un arbre VSG.

La zone active d'une primitive quelconque Pk de l'intersection n-aire est la suivante :

$$Z_T(Pk)=\overline{\bigcap_{j=1}^{m} P'j} \cap \bigcap_{\substack{i=1 \\ i\neq k}}^{n} Pi$$

Or, on a les équivalences suivantes : $\overline{A} \cap B = B \cap \overline{A} = B-A$. Donc $Z_T(Pk)$ peut s'écrire :

$$Z_T(Pk)=\bigcap_{\substack{i=1\\ i\neq k}}^{n} Pi - \bigcup_{j=1}^{m} P'j$$

La zone active d'une primitive de l'intersection n-aire est un arbre VSG. De plus il correspond à l'arbre VSG initial dans lequel on a enlevé la primitive en question.

La zone active d'une primitive quelconque Pl de l'union n-aire est la suivante :

$$Z_T(Pl)=\bigcap_{i=1}^{n} Pi \cap \overline{\bigcap_{\substack{j=1\\ j\neq l}}^{m} P'j}$$

En utilisant les mêmes équivalences que précédemment, la zone active peut s'écrire :

$$Z_T(Pl)=\bigcap_{i=1}^{n} Pi - \bigcap_{\substack{j=1\\ j\neq l}}^{m} P'j$$

Là encore, la zone active est l'arbre VSG initial moins la primitive considérée.

La recherche des zones actives pour un noeud quelconque d'un arbre VSG est directe. Elle ne nécessite aucune évaluation récursive. Il n'est pas nécessaire non plus de la stocker puisqu'elle se déduit immédiatement de l'arbre VSG. La zone active d'un noeud quelconque est égale à l'arbre VSG moins ce noeud. Compte tenu des remarques faites dans le chapitre 1, la zone active d'une primitive d'un arbre VSG ne présente aucun intérêt dans le cadre du lancer de rayons.

3.3.4. Les partitions spatiales.

Un objet décrit par un arbre CSG peut être décomposé en un ou plusieurs sous-objets, chacun représenté par un arbre VSG. Le volume de chaque sous-objet est inférieur ou égal au volume de l'objet initial. Par conséquent, la description d'une scène par des arbres VSG est plus précise, plus fine que par des arbres CSG (voir fig. 19). Elle favorise naturellement les partitions spatiales récursives dans le sens où elles peuvent mieux épouser la forme de la scène. De cette manière, l'accroissement du nombre d'entités manipulées est compensé par une meilleure décomposition et une meilleure répartition des sous-objets dans les cellules.

Lorsqu'un arbre CSG est associé à une cellule, certaines parties de cet arbre qui se situent à l'extérieur de la cellule peuvent ne pas influencer les calculs d'intersection entre le rayon et la

partie intérieure. C'est pourquoi les partitions spatiales s'accompagnent généralement d'une phase de restriction des arbres CSG par rapport aux cellules. L'algorithme classique de restriction proposé par Tilove[TIL 81] et repris par Bouatouch, Madani, Priol et Arnaldi [BOU 87] a été décrit dans le chapitre 1 (cf. §4.2.1.1.). Si un arbre VSG est associé à une cellule, cela signifie que son englobant se trouve au moins partiellement à l'intérieur de la cellule. Or cet englobant est construit à partir de la liste des primitives du brut. Elles se trouvent donc toutes au moins partiellement à l'intérieur de la cellule. Le brut ne peut donc pas être restreint. Par contre, un trou peut être complètement à l'extérieur de la cellule et être supprimé. La liste des trous peut être réduite à ceux dont l'englobant coupe la cellule. Cet algorithme est très voisin de l'algorithme de simplification des arbres VSG.

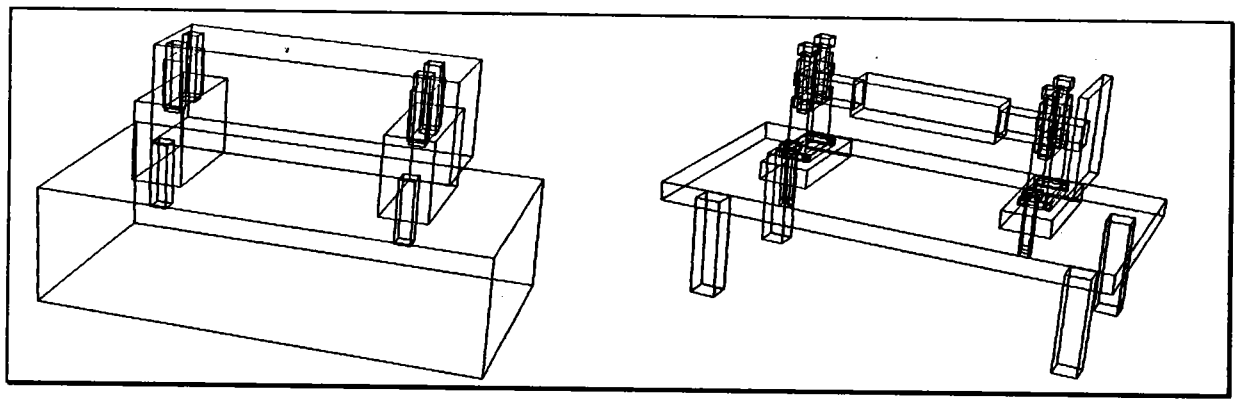


Figure 19 : Représentation d'une scène par les englobants des arbres CSG et des arbres VSG.

4. Conclusion.

Le modèle CSG est l'un des modèles géométriques les plus répandus. Sa simplicité de mise en oeuvre, ses facilités pour la description des solides et sa concision sont quelques uns de ses avantages. En revanche, il est assez mal adapté aux opérations de calcul et de visualisation, car les frontières des objets ne sont pas décrites explicitement, et nécessitent l'évaluation complète du modèle. De plus, la représentation des objets donnée par l'opérateur n'est pas toujours bien adaptée aux traitements envisagés.

Dans le cadre de l'approche multimodèles du système de CFAO SACADO nous avons proposé un modèle de visualisation appelé VSG (Visual Solid Geometry). La définition de ce modèle s'appuie essentiellement sur la notion de normalisation des arbres CSG. Cette transformation permet de décomposer un objet en une liste de termes qui sont encore des arbres CSG et qui sont indépendants du point de vue de la visualisation. Ils sont réorganisés pour obtenir une forme très particulière et constante quel que soit l'arbre CSG initial. Un arbre VSG est une différence entre une intersection n-aire de primitives et une union n-aire de primitives.

Les arbres VSG facilitent la mise en oeuvre du lancer de rayons. En particulier, la recherche itérative du point d'intersection entre un rayon et un arbre VSG permet d'utiliser le test d'appartenance qui est moins coûteux que le calcul d'intersection classique. Nous avons également montré que le modèle VSG inclut naturellement certaines optimisations :

- Grâce à la méthode de raffinement des englobants, nous avons montré que l'englobant d'un arbre VSG est minimal (au sens défini dans le paragraphe 4.2.2. du chapitre 1).

- L'automate d'intersection pour un arbre VSG est optimal. Le test d'appartenance permet en plus de ne calculer les intersections qu'avec les primitives effectivement traversées par un rayon.

Il facilite ou favorise également la mise en oeuvre d'autres optimisations :

- La zone active d'une primitive est l'arbre VSG moins cette primitive. Sa construction est inutile puisqu'elle est implicitement contenue dans cette représentation.

- Les subdivisions spatiales récursives épousent mieux la scène lorsqu'elle est représentée par des arbres VSG.

Ces propriétés sont importantes pour la visualisation. Elles montrent que sur ces différents aspects, le modèle VSG est satisfaisant au moins en théorie. Le seul point délicat est que la normalisation entraîne éventuellement un accroissement important de la complexité. Cette décomposition peut être avantageuse si l'on peut traiter les arbres VSG d'une manière cohérente. Dans le chapitre 3 nous proposons une méthode basée sur un classement afin d'utiliser au mieux les différentes propriétés de ce modèle de visualisation. Dans le chapitre 4, nous analysons les résultats de tests concrets pour mesurer l'intérêt réel du modèle. Nous essayons également de déterminer l'influence de la complexité sur les performances du lancer de rayons sur des scènes représentées par un grand nombre d'arbres VSG.

Certaines de ces propriétés (les englobants minimum et les zones actives notamment) sont également recherchées dans des contextes différents de la visualisation, ce qui permet d'envisager une utilisation plus générale de ce type de représentation. Par exemple, un problème classique est la détection d'une collision entre deux objets. Le modèle VSG est un moyen de décomposer ce problème. En effet, deux objets se recoupent si au moins deux de leurs arbres VSG se recoupent. La forme caractéristique des arbres VSG permet alors de simplifier cette opération. Deux arbres VSG se recoupent si leurs bruts se recoupent et si le brut de l'un n'est pas inclus dans un trou de l'autre.

Chapitre 3

Optimisation par classement a priori des objets

1. Introduction.

Dans le premier chapitre, nous avons rappelé les principales méthodes d'optimisation du lancer de rayons sur un arbre CSG. Cette étude de l'existant a permis de mettre en évidence un point fondamental qui est la recherche d'une meilleure représentation pour la visualisation. Nous avons alors proposé, dans le deuxième chapitre, le modèle VSG dont les propriétés sont très intéressantes en particulier pour la mise en oeuvre du lancer de rayons. Dans ce troisième chapitre, nous proposons une méthode d'optimisation qui s'efforce de tenir compte des caractéristiques de ce nouveau modèle.

Lorsqu'un arbre CSG est visualisé grâce à l'algorithme du lancer de rayons, les primitives sont étudiées dans l'ordre des opérations booléennes imposé lors de la construction de l'objet. Sa transformation dans le modèle VSG produit une ou plusieurs parties qui peuvent être affichées séparément et dans un ordre quelconque. L'idée consiste à rechercher un classement adéquat évitant au maximum les calculs d'intersection, en ordonnant par exemple les différents arbres VSG en fonction de leur profondeur par rapport à l'observateur.

Les subdivisions spatiales sont les techniques les plus utilisées pour optimiser le lancer de rayons. Leur principe est relativement simple. L'espace est décomposé en cellules (voxels) régulièrement ou récursivement. Chaque cellule contient une liste d'arbres ou de sous-arbres CSG. Lorsqu'un rayon est lancé, il traverse successivement une série de cellules, de la plus proche vers la plus éloignée. Seuls les arbres (ou sous-arbres) contenus dans ces cellules sont pris en compte dans la recherche d'un point d'intersection. Le trajet du rayon s'arrête lorsqu'une intersection est trouvée. Donc, une subdivision constitue un moyen d'obtenir le classement souhaité. Mais on peut noter que pour chaque rayon, il est nécessaire de rechercher de nouveau les cellules traversées et ce au prix de calculs non négligeables. De plus la taille des structures mises en oeuvre et les conséquences sur le coût de leurs traversées limite souvent la taille des cellules et donc la précision avec laquelle est approchée la scène. Il serait peut être plus intéressant, au moins pour certains types de rayon, de calculer directement une liste ordonnée des objets.

Classer des objets est un problème qui a déjà été étudié dans le cadre de l'élimination des parties cachées pour des modèles par les limites. Nous rappelons dans un premier temps quelles sont les principales solutions existantes et détaillons en particulier la méthode de l'arbre BSP. Dans un deuxième temps, nous montrons comment utiliser le principe de ce dernier pour optimiser le lancer de rayons.

2. Problème de la liste de priorités.

Parmi tous les algorithmes d'élimination des parties cachées, l'un des plus simples est sans doute celui du peintre. En effet, il suffit d'afficher les objets constituant la scène, du plus éloigné au plus proche de manière à ce qu'à la fin il ne reste que les parties visibles. Le principe est le même que celui des peintres qui superposent les différents plans qui composent leurs tableaux. Aucun calcul n'est par conséquent nécessaire pour exprimer de manière explicite les parties cachées (ou les parties visibles). La principale difficulté de cette méthode est de déterminer l'ordre dans lequel doivent être affichés les objets afin de garantir la correction du résultat. Le problème à résoudre revient à trier des objets en trois dimensions.

Deux méthodes se sont développées simultanément pour résoudre ce problème dans une scène modélisée par des facettes planes. Elles ont chacune une approche différente. D'un côté, l'algorithme de Newell, Newell et Sancha [NEW 72] trie les faces potentiellement visibles par un certain nombre de tests. Il nécessite peu de place mémoire mais étant dépendant de la position de l'observateur, le tri doit être recommencé à chaque déplacement. De l'autre côté, l'algorithme de Schumacker, Brand, Gilliland et Sharp [SCH 69], propose une structure indépendante de l'observateur mais nécessite en revanche suffisamment de mémoire pour être stockée.

2.1. L'algorithme de Newell, Newell et Sancha [NEW 72].

Dans la description de l'algorithme, on considère le repère tel que l'oeil soit son origine et la direction de visée soit son axe des z.

La première étape de l'algorithme classe l'ensemble des faces de la scène par ordre de profondeur croissante du sommet le plus éloigné de chaque face (voir fig. 1).

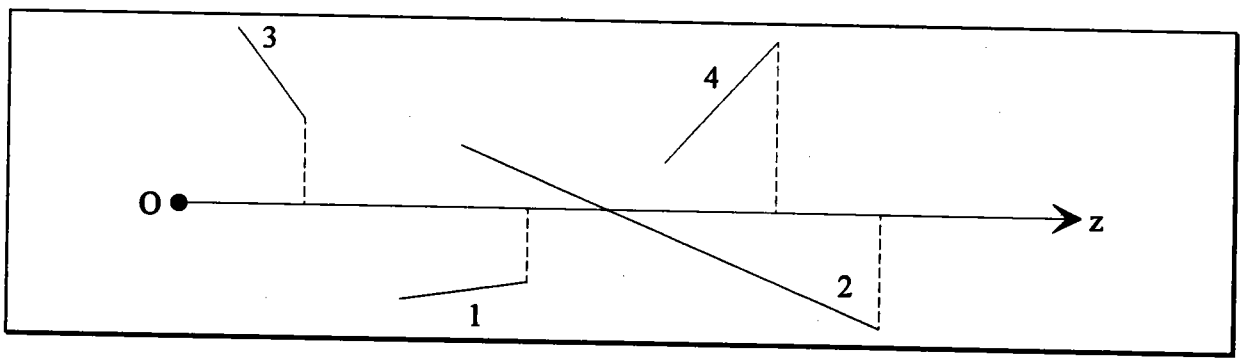


Figure 1 : Classement des faces par rapport à la profondeur.
Les faces sont classées dans l'ordre : 3 1 4 2.

La deuxième étape vérifie si la liste ainsi obtenue est ou n'est pas dans un ordre de priorité décroissante. Pour ce faire, on commence par étudier la dernière face P de cette liste. Si l'avant dernière face Q ne recouvre pas en profondeur la face P alors aucune face précédant Q ne le peut. P a la priorité la plus faible et peut être affichée. Si ce n'est pas le cas, il faut utiliser d'autres tests pour séparer non seulement P et Q mais P et toutes les autres faces la recouvrant en profondeur. On peut noter cependant que ces dernières ne représentent pas (sauf cas particulier) la totalité des faces. Elles constituent un ensemble contigu dans la liste jusqu'à la première ne recouvrant pas P. Dans l'exemple de la figure 1, la vérification de 2 nécessite l'étude de 4 et de 1.

Le test "la face P peut-elle cacher la face Q ?" doit être effectué un très grand nombre de fois. Toute l'efficacité de l'algorithme réside dans son aptitude à répondre "non" pour afficher la face P le plus tôt possible. La comparaison sur la profondeur n'est pas suffisante dans tous les cas. C'est pourquoi, les auteurs proposent d'exécuter en séquence six tests (voir fig. 2). Dès que l'un d'entre eux est vrai, la réponse à la question est "non" et P peut être affichée avant Q. L'ordre des tests est déterminé par le nombre de calculs qu'ils engendrent. Si cette séquence ne permet toujours pas de classer P par rapport à Q, alors les deux faces sont permutées et le processus recommence avec Q dans le rôle de la dernière face de la liste. Pour ne pas avoir de bouclage infini en cas de nouvel échec, une marque est associée à P.

1. P ne recouvre pas Q en Z
2. P ne recouvre pas Q en X
3. P ne recouvre pas Q en Y
4. Tous les sommets de P se situent derrière le plan de Q
5. Tous les sommets de Q se situent devant le plan de P
6. Les projections sur l'écran de P et Q ne se recouvrent pas

Figure 2 : Les six tests d'arrêt.

Si malgré tout, deux faces P et Q n'ont pu être classées, la situation est celle d'un recouvrement cyclique (voir fig. 3) et nécessite le coupage d'une face par rapport au plan de l'autre. Les deux nouvelles faces sont insérées à la place de l'ancienne et la procédure de test est relancée.

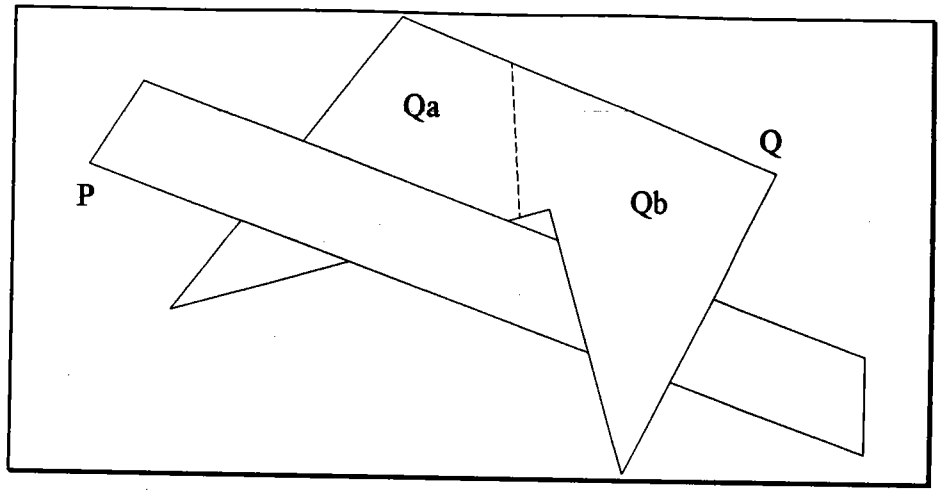


Figure 3 : Recouvrement cyclique. L'algorithme ne peut classer P par rapport à Q mais après coupage affichera Qa, P puis Qb.

L'algorithme de Newell, Newell et Sancha dépend à la fois de la position de l'observateur et de la géométrie de la scène. La modification de l'un ou de l'autre nécessite de recalculer complètement la liste de priorités. De plus, les tests complexes et les subdivisions de polygones sont coûteux en calculs et pénalisent la méthode d'autant qu'ils sont nombreux.

2.2. L'algorithme de Schumacker, Brand, Gilliland et Sharp [SCH 69].

La méthode proposée par les auteurs s'appuie sur le constat que dans certains contextes, comme les simulateurs de vol, la position de l'observateur change beaucoup plus fréquemment que la géométrie de la scène. La liste de priorités n'est pas considérée comme une opération liée à la position de l'observateur mais plutôt comme une caractéristique géométrique de la scène. Ils posent comme hypothèses que toutes les faces sont convexes et que la scène est constituée de groupes de faces pouvant être séparés géométriquement par un plan.

Dans la méthode de Newell, la priorité d'une face est déterminée de manière globale en étudiant toutes les autres faces. En revanche, Schumacker distingue deux niveaux de priorité. Le niveau le plus élevé caractérise l'ordre d'affichage des groupes de faces. Ce type de priorité est global puisqu'il dépend de la géométrie de l'ensemble de la scène. Le deuxième niveau est local à un groupe. La priorité d'une face est déterminée afin d'être indépendante de l'observateur et en ne prenant en compte que les faces du groupe. La priorité globale d'une face peut être vue comme un nombre où la partie entière représente la priorité de groupe et la partie décimale représente celle de la face (Pr=groupe,face).

Lorsqu'une scène est composée de groupes de faces distincts les uns des autres, l'espace peut être décomposé en régions telles que tout déplacement de l'observateur à l'intérieur d'une région engendre le même ordre d'affichage de ces groupes. Schumacker obtient une telle

décomposition en utilisant des plans séparant les différents groupes. Il construit un arbre binaire où les noeuds contiennent les plans, et les feuilles, qui sont en fait les régions, contiennent les listes ordonnées des groupes (voir fig. 4). La liste de priorités des groupes de faces est obtenue en déterminant dans quelle région l'observateur se situe par un parcours de l'arbre. Cet arbre est par conséquent indépendant de l'observateur puisqu'il contient toutes les situations possibles. Le fait que les plans choisis ne sont pas directement liés aux faces de la scènes autorisent le déplacement ou la modification d'un groupe à l'intérieur d'une région sans qu'il soit nécessaire de reconstruire l'arbre.

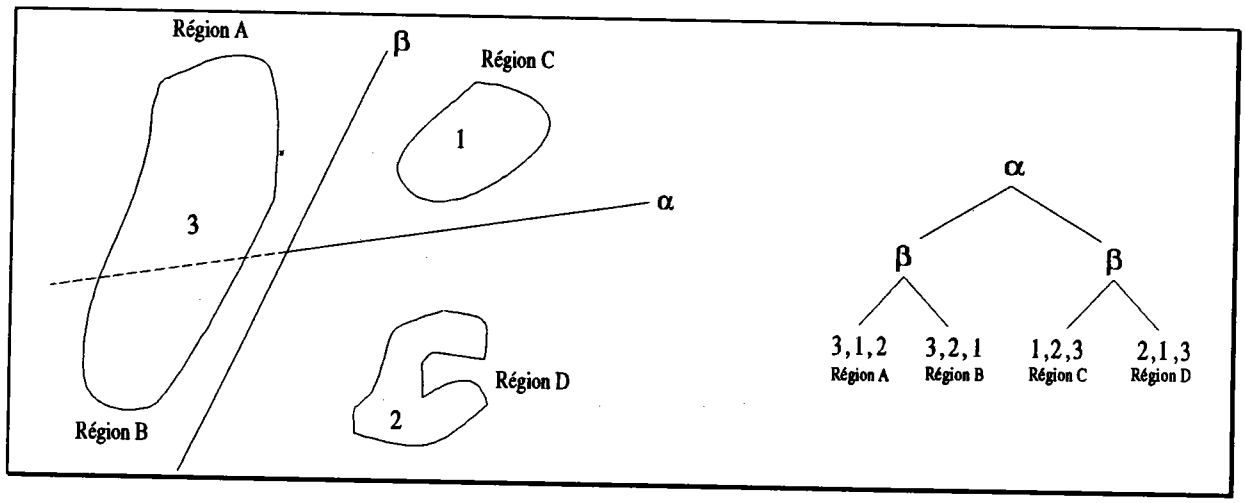


Figure 4 : Arbre binaire pour déterminer la priorité de groupe.

A l'intérieur d'un groupe, la priorité d'une face est déterminée de la manière suivante : si quel que soit le point de vue, la face A cache la face B alors la priorité de A est supérieure à celle de B. En répétant cette condition pour tous les couples de faces du groupe, on obtient un graphe. Si ce graphe ne contient aucun cycle, alors une priorité peut être affectée à chaque face (voir fig. 5). Sinon, le groupe doit être divisé manuellement en deux nouveaux groupes de manière à supprimer ces cycles.

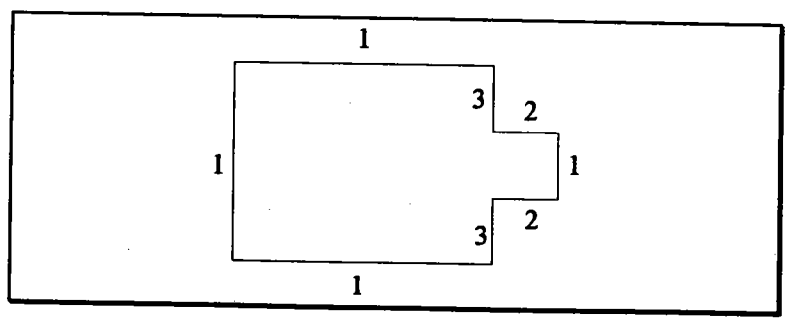


Figure 5 : Priorité de face. Quelle que soit la position de l'observateur, les faces sont affichées dans l'ordre de priorité croissante (3,2,1).

L'intérêt essentiel des travaux de Schumacker, est d'avoir montré le potentiel d'un classement a priori par une partition statique de l'espace. L'indépendance de la structure par rapport à l'observateur est une propriété fondamentale en synthèse d'images. Cependant, la méthode en elle même souffre d'un certain nombre de défauts. En particulier, l'intervention d'un opérateur pour la subdivision d'un groupe est une contrainte importante. Le calcul des plans de subdivision est également délicat et très coûteux.

2.3. Les arbres BSP.

Les arbres BSP (Binary Space Partitioning) reprennent directement la méthode de Schumacker. Mais contrairement à cette dernière, on ne distingue plus de groupes d'objets et les plans de subdivision sont simplement les faces de la scène. Les arbres BSP sont également indépendants de l'observateur. La simplicité et l'efficacité des algorithmes mis en oeuvre ont fait des arbres BSP un outil de base de la synthèse d'images. Après en avoir rappelé le principe général, et une optimisation, nous montrons deux exemples d'utilisation dans des contextes très différents.

2.3.1. Principe général.

Le principe général des arbres BSP a été décrit par Fuchs, Kedem et Naylor[FUC 80]. Les auteurs ont repris le principe de partition statique de l'espace de Schumacker mais ont supprimé l'étape de calcul des plans de subdivision en prenant directement les faces de la scène. Un arbre BSP est un arbre binaire où les noeuds et les feuilles sont des faces. Sa construction est récursive. Une face est choisie dans la liste des faces de la scène et constitue la racine de l'arbre. Le plan contenant cette face divise l'espace en deux demi-espaces, l'un devant et l'autre derrière en fonction de la normale. Chacune des faces restantes se situe donc soit dans un demi-espace soit dans l'autre. Celles situées de part et d'autre sont divisées par rapport au plan en deux nouvelles faces dont l'une est devant et l'autre derrière. On obtient finalement deux listes de faces, l'une devant et l'autre derrière. En appliquant le principe récursivement sur la liste devant on obtient le fils gauche de l'arbre, et sur la liste derrière on obtient le fils droit (voir fig. 6).

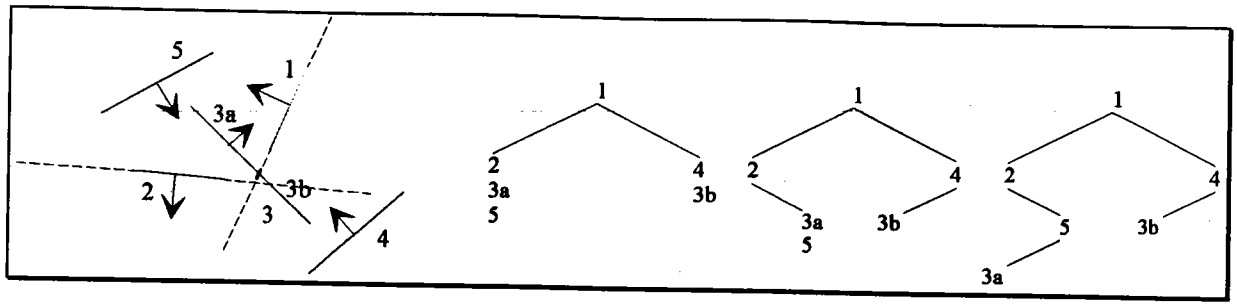


Figure 6 : Construction d'un arbre BSP.

L'algorithme détaillé pour la construction d'un arbre BSP est le suivant :

```
Fonction ArbreBSP(ListeFaces) : Arbre
Début
    Si ListeFaces=∅ Alors
        ArbreBSP=∅
    Sinon
        Racine=PremièreFace(ListeFaces)
        ListeDevant=ListeDerrière=∅

        Pour toutes les faces F de ListeFaces Faire
            Si F devant Racine Alors
                ListeDevant=ListeDevant+F
            Sinonsi F derrière Racine Alors
                ListeDerrière=ListeDerrière+F
            Sinon
                Subdivise(F,Racine,FDevant,FDerrière)
                ListeDevant=ListeDevant+FDevant
                ListeDerrière=ListeDerrière+FDerrière
            FSi
        FPour

        ArbreBSP=ArbreBSP(ListeDevant)+Racine+ArbreBSP(ListeDerrière)
    FSi
Fin
```

Grâce à un arbre BSP, il est très facile et très rapide de déterminer une liste de priorités des faces pour un point donné. Il suffit de parcourir l'arbre en profondeur et de classer le point par rapport à la face contenue dans le noeud. Si le point est devant le plan, les faces les moins

prioritaires (ou les plus éloignées) se trouvent dans le demi-espace derrière. Les faces les plus prioritaires (ou les plus proches) sont dans le demi-espace devant. Si le point est derrière le plan, le classement est inversé. Le partitionnement de l'espace représenté par l'arbre BSP n'est pas unique. Il dépend essentiellement de l'ordre dans lequel sont examinées les faces. Par conséquent, la liste de priorités pour un point donné n'est pas unique non plus. L'algorithme détaillé pour obtenir la liste de priorités croissantes des faces est le suivant :

```
Fonction ListePriorités(Arbre,Point):ListeFaces
Début
    Si Arbre=∅ Alors
        ListePriorités=∅
    Sinon
        LPDevant=ListePriorités(FilsGauche(Arbre))
        LPDerrière=ListePriorités(FilsDroit(Arbre))
        Si Point devant Racine(Arbre) Alors
            ListePriorités=LPDerrière+Racine(Arbre)+LPDevant
        Sinon
            ListePriorités=LPDevant+Racine(Arbre)+LPDerrière
        FSi
    FSi
Fin
```

Comparé à la méthode de Newell ou de Schumacker, l'arbre BSP présente de nombreux avantages. Tout d'abord, il ne nécessite aucun choix particulièrement délicat. La sélection de la face placée à la racine est triviale puisque n'importe laquelle convient. Dans l'algorithme précédent, on prend simplement la première de la liste. Ensuite les calculs mis en oeuvre sont simples et peu coûteux. La subdivision d'une face par rapport à un plan est le point le plus compliqué de l'algorithme. Enfin cette méthode ne demande aucune restriction quant à la nature de la scène (convexité, répartition). On peut remarquer que les cas pathologiques de recouvrement cyclique sont traités naturellement. En revanche, contrairement à la méthode de Schumacker, l'arbre BSP n'autorise aucun déplacement. La modification d'une face remet en cause la totalité de l'arbre. Enfin, si toutes les faces peuvent être choisies pour être placées à la racine, toutes n'ont pas les mêmes conséquences. Certaines faces entraîneront plus de subdivisions que d'autres. Or tout accroissement du nombre de faces dans l'arbre BSP augmente non seulement la complexité de stockage mais également la complexité de l'algorithme de parcours, d'affichage etc. Un choix moins "trivial" s'impose pour limiter ce phénomène.

2.3.2. Optimisation.

Le point le plus important lors de la construction de l'arbre BSP, est de choisir le plus judicieusement possible la racine pour limiter le nombre de subdivisions. Fuchs, Abram et Grant [FUC 83] proposent de choisir aléatoirement un certain nombre de faces et de conserver celle entraînant le moins de subdivisions. Les exemples illustrant l'article montrent que l'arbre BSP obtenu par la méthode classique contient en moyenne 3.1 fois le nombre de faces de la scène alors qu'en étudiant un échantillon de 5 candidats, l'arbre ne contient plus que 1.7 fois le nombre de faces. Le coût de l'étude de cet échantillon n'est sans doute pas négligeable. Il est cependant regrettable que les auteurs n'en donnent aucune estimation. De même qu'ils ne donnent aucune information concernant les gains (éventuels) obtenus lors de l'affichage et des traitements liés à l'arbre BSP.

2.3.3. Applications.

2.3.3.1. Représentation d'un polyèdre.

Thibault et Naylor [THI 87] ont proposé les arbres BSP pour la représentation des polyèdres. Nous ne rappelons ici que les principes généraux dans un espace de dimension trois.

Pour représenter un polyèdre, un arbre BSP ne doit pas seulement représenter la répartition géométrique des faces dans l'espace mais doit contenir d'autres informations essentielles à la validité de l'objet. On doit y retrouver la notion d'intérieur (ou de matière), d'extérieur et de frontière.

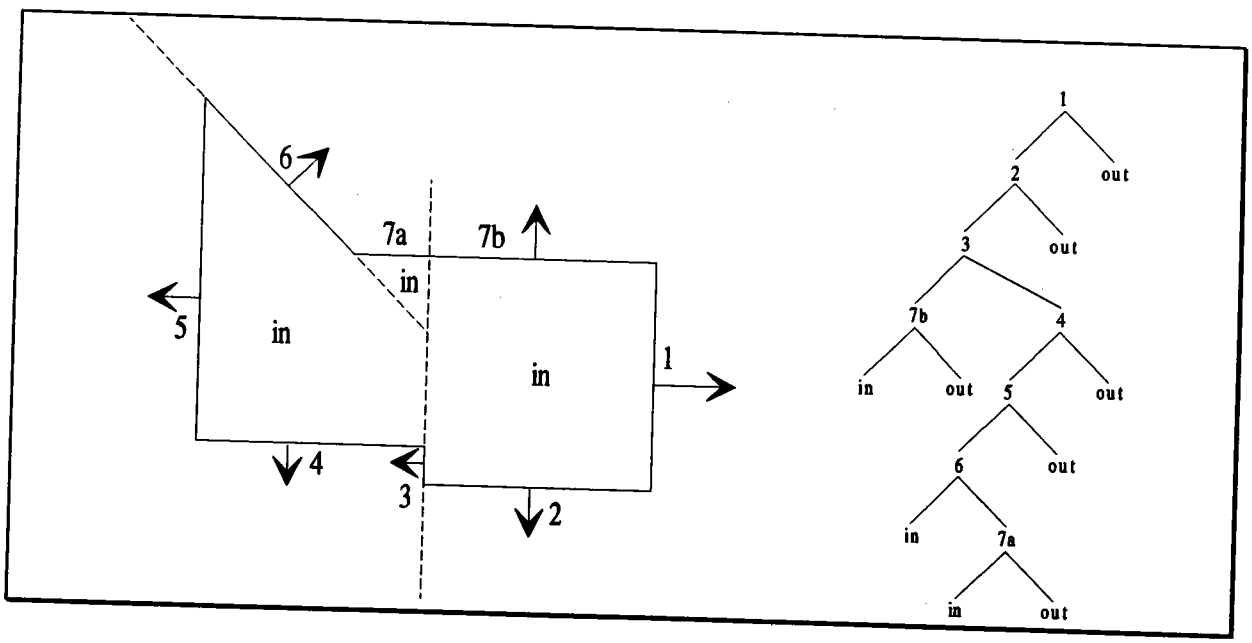


Figure 7 : Représentation d'un polyèdre par un arbre BSP.

L'exemple de la figure 7 montre un objet avec une des représentations possibles par un arbre BSP. Chaque noeud contient une liste de faces appartenant au même plan et telles que leurs normales soient orientées vers l'extérieur. Ce plan définit deux régions qui sont représentées par le fils droit pour le côté négatif et le fils gauche pour le côté positif. Les feuilles, appelées cellules, sont des régions homogènes, c'est à dire qu'elles sont soient complètement intérieures (in), soient complètement extérieures (out) au polyèdre. Une cellule intérieure est forcément fermée et convexe. Une cellule extérieure peut être ouverte ou fermée. Si le solide S est représenté par un arbre BSP alors on a les propriétés suivantes :

$S = \bigcup Ci$ tel que Ci=in

$FrS = \bigcup (Ci \cap Cj)$ tel que Ci=in et Cj=out

Les auteurs donnent un algorithme pour construire un arbre BSP à partir d'un polyèdre modélisé par les limites (B-Rep). Ils montrent également comment évaluer simplement les opérations booléennes entre l'arbre BSP d'un objet et un autre objet B-Rep ou entre deux objets B-Rep pour donner un nouvel arbre BSP.

2.3.3.2. Calcul des ombres portées.

Pour résoudre le problème des ombres portées, Chin et Feiner [CHI 89] proposent de construire les volumes d'ombres produits par une source. A chaque source lumineuse ponctuelle est associé un arbre SVBSP (Shadow Volume BSP). Ces arbres SVBSP sont en fait dérivés des arbres BSP utilisés pour représenter des polyèdres (voir § 2.3.3.1.). Les feuilles sont également des régions intérieures ou extérieures mais indiquent surtout si une région est dans l'ombre ou est éclairée. Les noeuds contiennent les plans passant par la source lumineuse et les deux extrémités des arêtes des faces de la scène.

Après avoir calculé un arbre BSP classique pour toute la scène, la liste de priorités décroissantes est construite depuis la position de la source lumineuse. Ensuite l'arbre SVBSP est construit de manière incrémentale en étudiant les faces dans l'ordre de cette liste. Pour chaque face (sauf les faces arrières pour la source) est construit un arbre SVBSP local qui est ajouté à l'arbre global de la source. Un exemple en deux dimensions est donné figure 8.

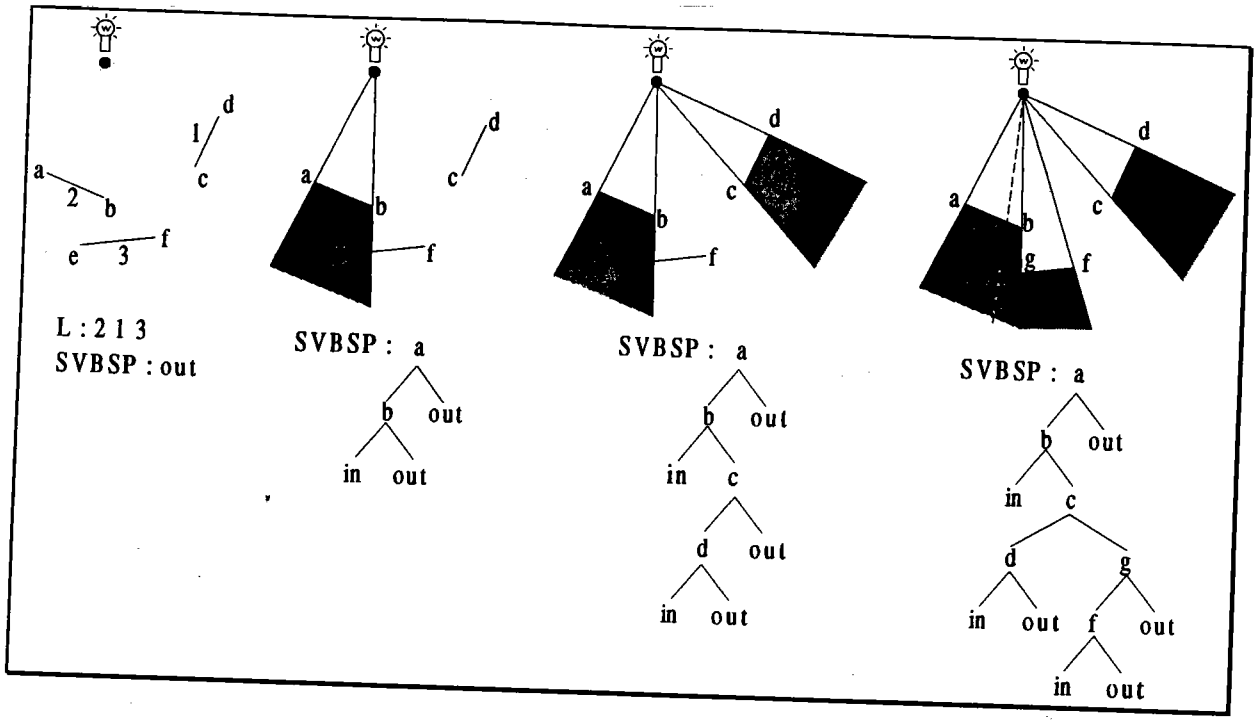


Figure 8 : Construction d'un arbre SVBSP.

Lorsqu'une face est affichée, elle est classée récursivement dans l'arbre SVBSP, en étant au besoin subdivisée par les plans délimitant les régions. Lorsqu'une face se situe complètement dans une région "out", elle est affichée en ajoutant la quantité de lumière provenant de la source. Si elle est dans une région "in", la source lumineuse n'est pas prise en compte dans l'éclairement de la face (elle est à l'ombre).

Une amélioration importante pour la construction des arbres SVBSP est de ne considérer que les arêtes de la silhouette des objets.

2.4. Conclusion.

Ordonner des faces ou des objets dans une scène est un problème très important en synthèse d'images. Nous avons rappelé les principaux algorithmes permettant de classer des faces. Nous avons en particulier, détaillé la technique des arbres BSP qui présente de nombreux avantages.

Lorsqu'on désire classer non pas des faces mais des volumes, seul l'algorithme de Schumacker répond au problème. Cependant, l'étape délicate du choix des plans de subdivision limite ses possibilités.

3. Proposition d'une optimisation du lancer de rayons.

En utilisant le même principe on va montrer qu'il est possible de construire très simplement un arbre BSP non pas de faces planes, mais de volumes englobants parallélépipédiques à faces parallèles aux plans des axes.

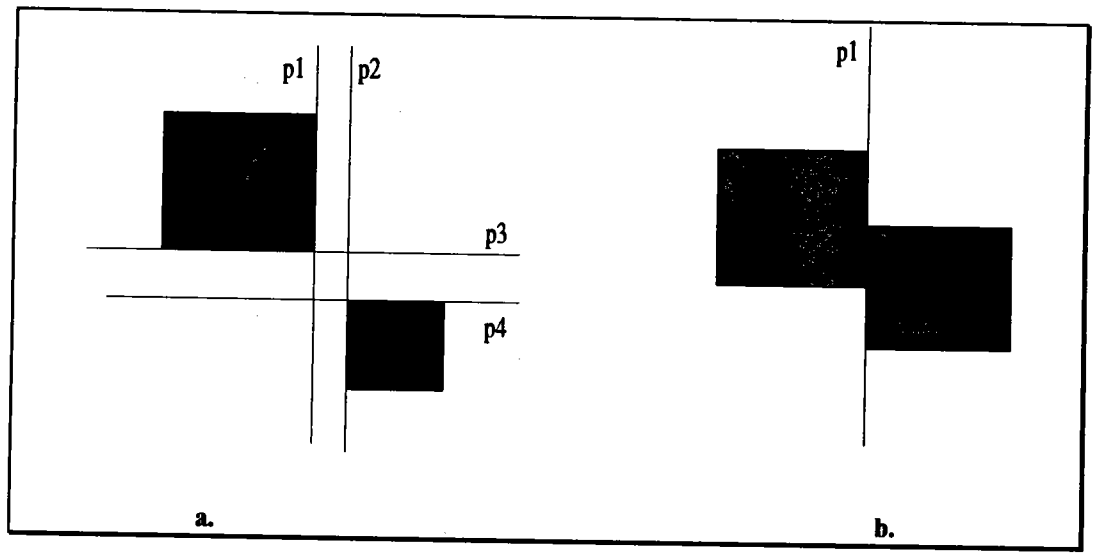


Figure 9 : Choix d'un plan de subdivision (exemple en 2 dimensions)

a. Cas général b. Cas de tangence.

Considérons tout d'abord deux englobants. S'ils ne se coupent pas, alors il existe au moins un plan passant par l'une de leurs faces tel que l'un des englobants soit devant et l'autre derrière. Ce plan convient toujours même en cas de tangence (voir fig. 9). S'ils se recoupent alors ce plan n'existe pas et il est impossible de les classer l'un par rapport à l'autre. Nous verrons ultérieurement que dans ce cas, le traitement d'un de ces englobants nécessite de toute manière le traitement des autres englobants qu'il coupe. Par conséquent, l'ordre de traitement d'une liste d'englobants qui s'enchevêtrent n'a pas d'importance.

Grâce à cette propriété, on peut construire un arbre BSP de manière assez voisine de la procédure classique. A partir de la liste des englobants de la scène, on recherche un plan respectant la condition énoncée précédemment. S'il existe, il est placé à la racine de l'arbre. Chaque englobant restant est classé soit devant soit derrière ce plan ou alors est subdivisé en deux nouvelles parties. Bien qu'elles ne le soient pas au sens habituel du terme, on continuera à appeler ces deux entités des volumes englobants. Le processus est répété récursivement sur les deux listes d'englobants ainsi construites. Si aucun plan n'est trouvé, l'arbre est une feuille contenant la liste des englobants. On obtient finalement un arbre binaire où les noeuds

contiennent des plans de subdivision et les feuilles contiennent des listes de volumes englobants s'enchevêtrant (voir fig. 10). L'algorithme de construction d'un arbre BSP de volumes englobants est le suivant :

```
Fonction ConstruitArbre(ListeEnglobants) : ArbreBSP
Début
    Si listeEnglobants=∅ Alors
        ConstruitArbre=∅
    Sinon
        Rechercher une face F dans ListeEnglobants séparant au moins deux englobants
        Si F existe Alors
            Répartir ListeEnglobants dans deux nouvelles listes : ListeDevant et ListeDerrière
            ConstruitArbre=ConstruitArbre(ListeDevant)+F+ConstruitArbre(ListeDerrière)
        Sinon
            ConstruitArbre=ListeEnglobants
        Fsi
    Fsi
Fin
```

La liste de priorités décroissantes des volumes englobants est obtenue grâce à l'algorithme suivant :

```
Fonction ListePriorités(Arbre,Point) : ListeClassée
Début
    Si Arbre=∅ Alors
        ListePriorités=∅
    Sinonsi Arbre=Feuille Alors
        ListePriorités=ListeObjets(Arbre)
    Sinon
        Si Point devant Racine(Arbre) Alors
            ListePriorités=ListePriorités(FilsGauche(Arbre),Point)+
                           ListePriorités(FilsDroit(Arbre),Point)
        Sinon
            ListePriorités=ListePriorités(FilsGauche(Arbre),Point)+
                           ListePriorités(FilsDroit(Arbre),Point)
        Fsi
    Fsi
Fin
```

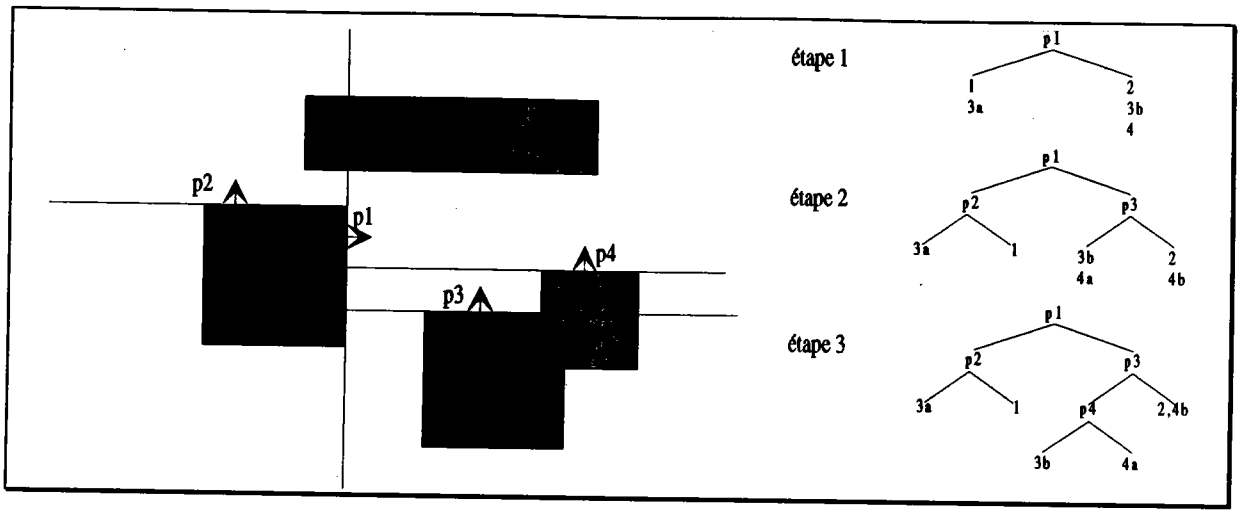


Figure 10 : Construction d'un arbre BSP de volumes englobants (exemple en 2 dimensions).

On peut noter qu'en raison de la nature particulière des objets manipulés, des volumes englobants parallélépipédiques à bords parallèles aux plans des axes, cette méthode ne nécessite aucun calcul ni pour le classement ni pour la subdivision. Les seules opérations utilisées sont des comparaisons. Les algorithmes mis en oeuvre sont extrêmement simples. On insiste également sur le fait qu'elle donne une liste classée de volumes englobants éventuellement partiels et non pas une liste d'objets. Le seul point réellement délicat de la construction de l'arbre est la recherche du plan de subdivision. Le problème est analogue à celui de la recherche de la face dans la méthode classique (§2.3.2). Les mêmes stratégies peuvent être mises en oeuvre pour optimiser la taille de l'arbre BSP.

L'objectif souhaité pour l'optimisation du lancer de rayons est une liste d'objets ordonnés du plus proche au plus éloigné d'un point et dans une direction donnée. Or, les algorithmes précédents permettent de classer des englobants par priorités décroissantes mais pas par rapport à une direction donnée. Il faut réduire la liste pour ne conserver que ceux situés dans la direction appropriée. Une telle opération est trop coûteuse pour un seul rayon. On va donc rechercher une méthode générale pour résoudre ce problème lorsque de nombreux rayons partent du même point et dans une direction cohérente. Ce qui est le cas notamment des rayons primaires, d'ombrage et certains rayons secondaires.

3.1. Optimisation des rayons primaires.

Dans le cas des rayons primaires, la méthode est semblable à un découpage (clipping) en deux dimensions. Tout d'abord, on élimine les englobants se situant entièrement derrière le plan

passant par l'observateur et parallèle à l'écran. Ensuite, pour ceux restants, on projette les huit points qui les délimitent sur le plan écran. On reconstruit alors le rectangle englobant de cette projection. C'est un englobant en deux dimensions de l'objet. Certes, ce n'est pas le meilleur puisque c'est un englobant d'englobant, mais c'est le plus simple et le plus rapide à calculer. Enfin on peut encore éliminer tous les rectangles ou parties de rectangles qui sont en dehors de l'écran (voir fig. 11). Il devient alors très facile pour chaque pixel de déterminer la liste classée de volumes englobants se trouvant sur la trajectoire du rayon primaire.

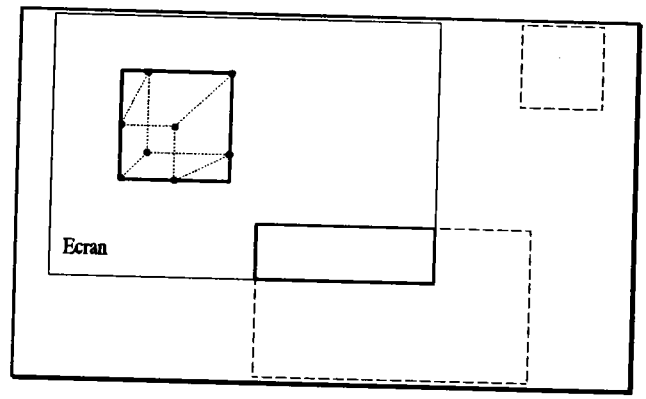


Figure 11 : réduction de la liste des objets.

Il est possible de tirer encore meilleur parti de cette projection. En effet, il n'est pas nécessaire d'étudier les rayons qui n'appartiennent à aucun rectangle englobant. Donc, plutôt que d'étudier systématiquement tout l'écran, il suffit de balayer l'un après l'autre les rectangles englobants en marquant au fur et à mesure les pixels étudiés. De plus, puisque la liste est classée, le nombre d'objets à considérer à chaque étape diminue de un.

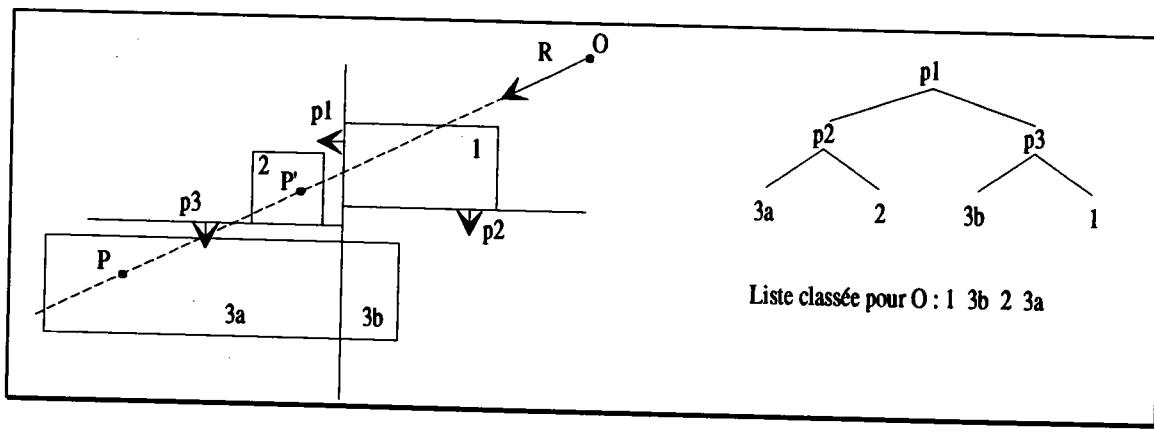


Figure 12 : Englobants partiels.
Les objets sont étudiés dans l'ordre de la liste de priorités. Le rayon ne coupe pas l'objet de l'englobant 1. L'étude se poursuit avec l'objet 3. Le point P trouvé n'appartient pas à l'englobant. Il est alors rejeté et l'objet 2 est étudié. Le point P' respecte la condition et est donc conservé.

Pour un rayon, les objets seront étudiés dans l'ordre décrit par la liste des volumes englobants. Or, puisque ces derniers peuvent éventuellement être partiels, il est possible de trouver un point d'intersection qui n'appartient pas au volume englobant considéré (voir fig. 12). Pour que l'ordre de la liste soit respecté, le point doit être ignoré. Le test d'appartenance d'un point à un volume englobant n'est pas très coûteux. Il suffit de comparer les coordonnées du point avec celles de l'englobant, soit six tests.

On a vu précédemment que lorsque plusieurs volumes englobants se recoupent, il n'est pas possible de les classer les uns par rapport aux autres. Donc quand un rayon coupe l'objet correspondant à l'un de ces englobants, il faut encore étudier les objets contenus dans les autres. De part la construction de l'arbre BSP et de la liste de priorités, ces objets sont contigus dans la liste. L'étude du rayon est terminée dès que l'englobant considéré ne coupe pas son suivant.

L'algorithme du lancer de rayons primaires est le suivant :

```
Procédure LanceRayonsPrimaires( V : Point de vue; Proj : Projection)
Début
    L=Liste ordonnée des objets de la projection primaire Proj
    Tant que L≠∅ Faire
        O=Premier(L)
        Pour tous les pixels P non marqués de RectangleEnglobant(O) Faire
            { VP vecteur directeur du rayon primaire partant de V et passant par P}
            I=CalculIntersectionObjet(Objet(O),VP)
            Si I existe et I∈ VolumeEnglobant(O) Alors
                marqué(P)=Vrai
                Tant qu'il existe O' tel que (O'∈ Suivant(L)) et
                (VolumeEnglobant(O')∩VolumeEnglobant(O)≠∅) Faire
                    I'=CalculIntersectionObjet(Objet(O'),VP)
                    Si I' existe et I'<I Alors
                        I=I'
                    Fsi
                Ftant
            Fsi
            Couleur=Eclairement(I)
            Afficher(P,Couleur)
        Fpour
        L=Suivant(L)
    Ftant
Fin
```

On peut noter que dans cet algorithme, il est possible de calculer plusieurs fois la même intersection entre un rayon et un objet. Pour éviter ces calculs inutiles, une "boîte aux lettres"[BOU 87] est associée à chaque objet. Elle contient d'une part le numéro du dernier rayon l'ayant coupé et d'autre part toutes les informations relatives au point d'intersection éventuellement trouvé. Ainsi, si le numéro du rayon étudié est égal à celui de la boîte aux lettres, il suffit de reprendre ces informations. Ce principe est également appliqué aux primitives puisqu'elles peuvent être partagées par plusieurs arbres VSG.

3.2. Optimisation des rayons d'ombrage.

Nous supposons dans un premier temps que la source lumineuse (ponctuelle) se situe à l'extérieur de la scène, c'est-à-dire qu'il existe un plan passant par elle tel que tous les objets soient contenus dans le même demi-espace. La position d'une source lumineuse permet de déterminer, grâce à l'arbre BSP, la liste ordonnée des volumes englobants. A partir d'une direction choisie pour la source lumineuse (vers le centre de la scène par exemple), on définit un plan virtuel perpendiculaire à cette direction et situé à une distance arbitraire. On y projette alors les volumes englobants afin d'en construire les rectangles englobants. Il n'est pas possible de réduire davantage cette liste par un découpage, comme pour les rayons primaires, car il n'existe pas de clôture équivalente à l'écran. Chaque point pour lequel on étudie un rayon d'ombrage est lui-même projeté sur ce plan. Il suffit alors d'étudier dans l'ordre ceux des objets tels que le point appartienne à leurs rectangles englobants. L'étude commence par l'objet le plus proche de la source lumineuse et peut s'arrêter sur l'objet contenant le point. Il faut considérer alors, comme pour les rayons primaires, tous les objets tels que leurs englobants se coupent et qui sont placés après lui dans la liste (voir fig. 13).

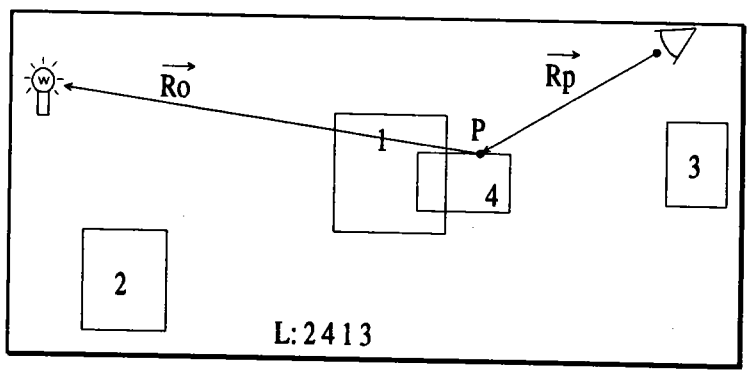


Figure 13 : Lancé d'un rayon d'ombrage. Après avoir calculé les intersections entre le rayon Ro et les objets 2 et 4 sans résultat, il faut encore étudier 1 qui peut produire une ombre car son englobant coupe celui de 4.

L'algorithme de lancé d'un rayon d'ombrage est le suivant :

```
Procédure LanceRayonOmbrage( P : Point; S : Source; Proj : Projection; Trouvé )
Début
    M=Projection de P sur le plan de la projection d'ombrage Proj
    L=Liste ordonnée des objets de la projection
    Trouvé = Faux
    Tant que non Vide(L) et non Trouvé Faire
        O=Premier(L)
        Si M∈RectangleEnglobant(O) Alors
            {PS vecteur directeur du rayon d'ombrage partant de P et s'arrêtant en S}
            I=CalculIntersectionObjet(Objet(O),PS)
            Si I existe et P≤I≤S Alors
                Trouvé=Vrai
            Sinon Si Objet(P)=Objet(O) Alors
                Tant qu'il existe O' tel que (O'∈ Suivant(L)) et
                (VolumeEnglobant(O')∩VolumeEnglobant(O)≠∅) et
                non Trouvé Faire
                    I=CalculIntersectionObjet(Objet(O'),PS)
                    Si I existe et P≤I≤S Alors
                        Trouvé=Vrai
                    Fsi
                Ftant
            Fsi
        Fsi
        L=Suivant(L)
    Ftant
Fin
```

Lorsque la source est située à l'extérieur de la scène, les directions des rayons d'ombrage décrivent une demi-sphère. A partir du moment où la direction choisie pour la source n'est pas opposée à la direction de cette demi-sphère, un seul plan de projection suffit. Lorsque la source est à l'intérieur de la scène, les directions des rayons peuvent former une sphère complète. Une solution à ce problème est l'utilisation de la technique du Lightbuffer [HAI 86]. Elle consiste à englober la source lumineuse dans un cube de côté arbitraire dont les faces sont parallèles aux plans des axes. A chaque face est alors associée une projection. Un rayon d'ombrage ne peut couper qu'une seule des six faces.

3.3. Optimisation des rayons secondaires.

On a montré que l'optimisation par classement a priori n'est rentable que dans la mesure où les englobants de la liste de priorités peuvent être projetés sur un plan lié à une direction particulière. Le problème à résoudre est le même que celui rencontré dans la décomposition adaptative [VIV 93a] ou le Beam-Tracing [HEC 84]. On en déduit donc les mêmes limites pour les rayons secondaires. Seules les faces planes de la scène permettent la mise en oeuvre de la méthode. La contrainte est encore plus forte lorsque les objets sont transparents puisque le indice de réfraction doit être égal à un.

La modélisation CSG impose encore certaines restrictions. Toutes les faces planes des primitives (si elles en contiennent) de la scène ne doivent pas être prises en compte. En effet, certaines d'entre elles sont invisibles, notamment dans le cas d'une différence. Mais le problème posé est beaucoup plus important. La question est : "est-ce que toutes les faces doivent être étudiées ou encore, quelles sont les faces de la scène qui valent la peine d'être étudiées ?". Y répondre nécessite l'étude de critères difficiles à évaluer. Par exemple, est-ce qu'une face d'une primitive appartient à l'enveloppe extérieure de l'objet ? (voir fig. 14 a.). Ou bien, est-ce que la surface apparente d'une face depuis l'observateur est suffisamment grande pour faire l'objet d'une étude particulière (voir fig. 14 b.). De nombreux problèmes restent à résoudre dans ce domaine. Actuellement, nous avons choisi d'étudier toutes les faces planes appartenant à une primitive positive c'est-à-dire correspondant à de la matière. Pour tous les autres cas, les rayons secondaires sont optimisés par une méthode classique basée sur une subdivision spatiale.

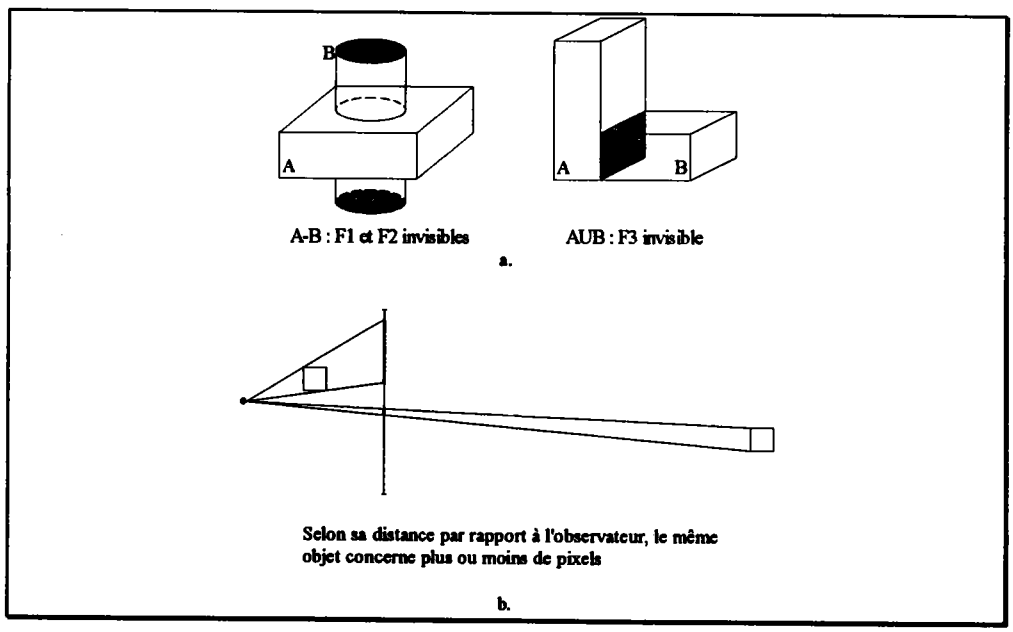


Figure 14 : Faces invisibles ou de faible influence sur l'image.

3.3.1. Optimisation des rayons réfléchis.

Lorsqu'un observateur regarde une surface plane réfléchissante, l'image qu'il voit est celle qu'il verrait s'il était situé symétriquement de l'autre côté de cette surface (voir fig. 15). Il est donc possible de retrouver les informations indispensables à l'application de la méthode. L'observateur secondaire (O') peut être considéré comme l'origine des rayons secondaires réfléchis. De la même façon on peut définir la direction secondaire comme étant la direction de visée symétrique (v').

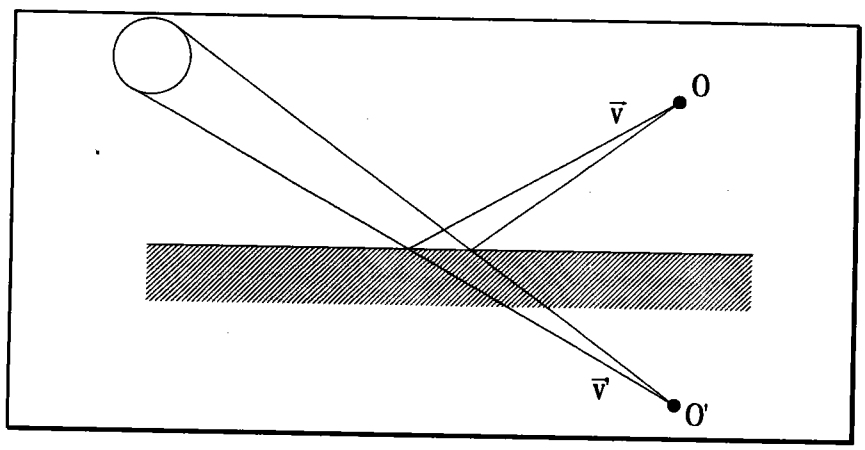


Figure 15 : Observateur secondaire.

L'observateur secondaire permet d'obtenir, grâce à l'arbre BSP de la scène, la liste ordonnée des objets. Cette liste est réduite en ne considérant que ceux situés devant la face réfléchissante. Ceux situés derrière ne peuvent pas être vus par réflexion. Ensuite les englobants des objets sont projetés sur un plan virtuel placé à une distance arbitraire de l'observateur secondaire (voir fig. 16).

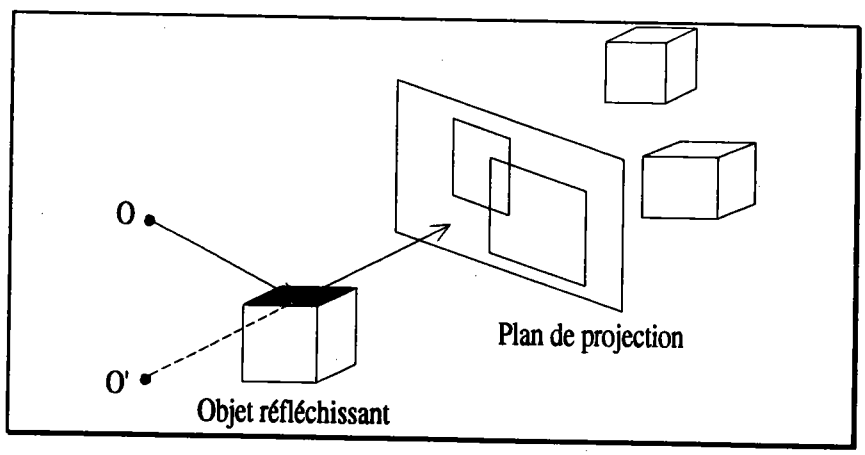


Figure 16 : Projection secondaire pour les rayons réfléchis.

On peut remarquer que dans le cas des rayons réfléchis, il est inutile d'associer une projection aux faces arrières car elles ne peuvent pas renvoyer de lumière vers l'observateur. Ce qui élimine naturellement au moins la moitié des faces potentielles de la scène.

3.3.2. Optimisation des rayons transmis.

Lorsqu'un objet est transparent, si son indice de réfraction est différent de un, cela signifie que les rayons lumineux sont déviés par la matière. Malheureusement, il n'existe pas un point unique correspondant à l'origine de ces rayons réfléchis. Il n'est donc pas possible d'appliquer le principe de la projection. Lorsque l'indice est égal à un, les rayons ne subissent aucune déviation. Dans ce cas l'observateur secondaire est l'observateur primaire, et la direction secondaire est la direction de visée. On retrouve alors les mêmes conditions que pour les rayons réfléchis (voir fig. 17). Les objets situés derrière la face considérée sont éliminés de la liste car ils ne peuvent plus être rencontrés.

Contrairement aux rayons réfléchis, on applique la méthode uniquement aux faces arrières de l'objet transparent. En effet, il est inutile de prendre en compte les faces avant puisqu'elles correspondent à l'entrée dans la matière du rayon. Le but étant d'optimiser les rayons qui en sortent.

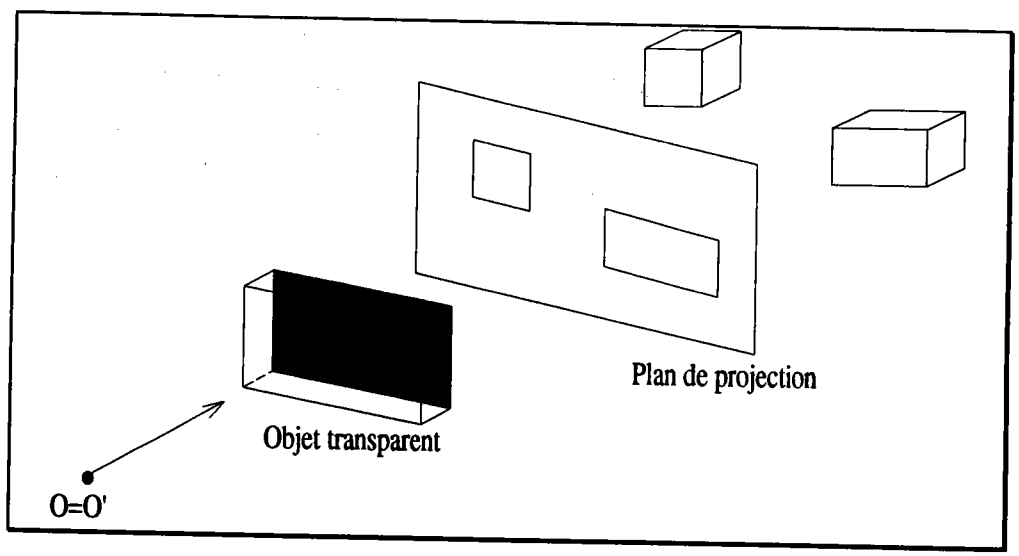


Figure 17 : Projection secondaire pour les rayons transmis.

3.3.3. Algorithme de lancer d'un rayon secondaire.

Comme pour un rayon d'ombrage, il faut projeter l'origine du rayon secondaire sur le plan de la projection. Ensuite, l'algorithme est pratiquement le même que celui d'un rayon primaire.

```
Procédure LanceRayonSecondaire( P : Point; V : Vecteur; Proj : Projection ; C : Couleur )
Début
    M=Projection de P sur le plan de projection secondaire
    L=Liste ordonnée des objets de la projection
    Trouvé=Faux
    Tant que non Vide(L) et non Trouve Faire
        O=Premier(L)
        Si M∈RectangleEnglobant(O) Alors
            I=CalculIntersectionObjet(Objet(O),V)
            Si I existe et I∈ VolumeEnglobant(O) Alors
                Trouvé=Vrai
                Tant qu'il existe O' tel que (O'∈ Suivant(L)) et
                (VolumeEnglobant(O')∩VolumeEnglobant(O)≠∅) Faire
                    I'=CalculIntersectionObjet(Objet(O'),V)
                    Si I' existe et I'<I Alors
                        I=I'
                    Fsi
                Ftant
                C=Eclairement(I)
            Fsi
        Fsi
        L=Suivant(L)
    Ftant
Fin
```

3.3.4. Construction d'un arbre de projections.

L'observateur de la scène va engendrer un observateur secondaire pour chaque face plane réfléchissante ou transparente et appartenant à une primitive positive. A chacun de ces observateurs secondaires est associée une projection qui permet d'optimiser les rayons secondaires du premier niveau de l'arbre de rayons (cf. chap.1 §3.1.3.4.) qui partent de ces faces. Si l'on souhaite optimiser de la même façon les rayons du deuxième niveau il suffit d'appliquer le processus récursivement pour chacun des observateurs secondaires. On obtient

finalement un arbre de projections où les noeuds et les feuilles contiennent une liste de projections (voir fig. 18).

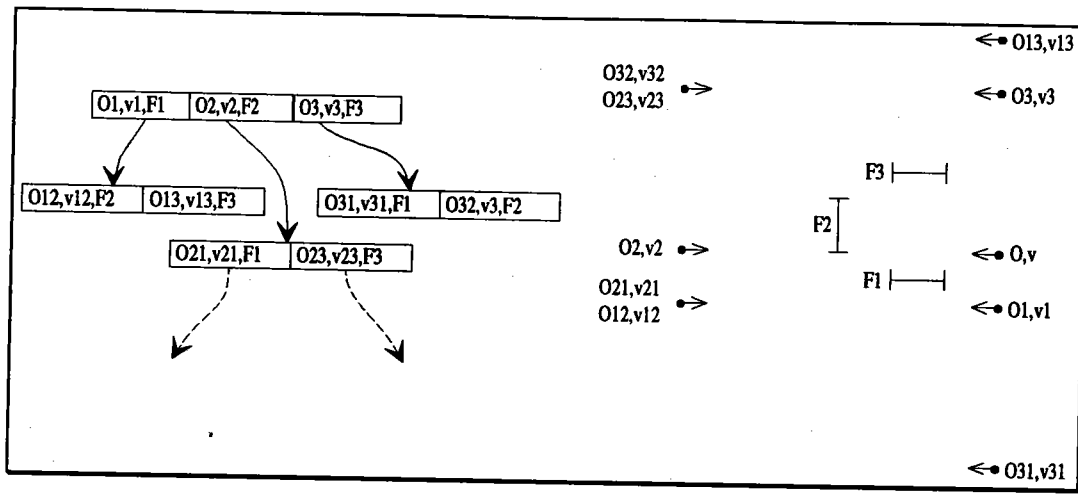


Figure 18 : Arbre de projections.

Pour un rayon secondaire donné, on recherche dans la liste la projection correspondante d'après la face contenant son origine. S'il engendre un nouveau rayon secondaire, on considère alors la liste fille de cette projection.

Un inconvénient majeur est que la taille de l'arbre a tendance à croître de manière exponentielle avec la profondeur. Par exemple, si la scène contient dix faces se réfléchissant mutuellement, le deuxième niveau contiendra cent projections et le troisième dix mille! La gestion d'un nombre aussi important de projections pose non seulement des problèmes d'occupation mémoire mais également des problèmes de temps de construction et de recherche. Cela limite encore les conditions d'application de la méthode.

4. Conclusion.

Le lancer de rayons est encore aujourd'hui le principal algorithme à produire des images réalistes à partir du modèle CSG exact. L'étude des méthodes d'optimisation existantes a montré l'intérêt d'avoir une représentation spécifique pour la visualisation. Dans le deuxième chapitre, nous avons donné les bases de ce qui peut être un modèle de visualisation. Les propriétés liées aux arbres VSG ont quelque peu modifié l'approche classique du lancer de rayons. Dans ce troisième chapitre, nous avons proposé une méthode qui s'efforce de tenir compte de ces nouvelles données.

Le classement a priori appliqué aux arbres VSG s'appuie sur deux aspects particuliers de cette représentation :

- L'arbre BSP de la scène est construit entièrement à partir des englobants des objets. Or les englobants des arbres VSG sont minimaux (cf. chap. 2 §3.3.1.). La méthode proposée devrait donc bénéficier de cette propriété. Notamment, leurs projections sur l'écran devraient réduire le nombre de pixels à étudier.

- Un arbre CSG est décomposé en un ou plusieurs arbres VSG. Ces derniers peuvent être considérés comme des objets distincts et être affichés dans un ordre quelconque. Par conséquent, le classement a priori devrait réorganiser leur affichage d'une manière plus cohérente avec le processus de visualisation, évitant ainsi de nombreux calculs d'intersection.

Dans le cas des rayons primaires et d'ombrage, la projection des englobants est relativement simple à mettre en oeuvre. En revanche, elle pose des problèmes importants dans le cas des rayons secondaires. La projection n'est possible que sous certaines conditions très restrictives et la taille de l'arbre des projections secondaires croît de manière exponentielle avec le niveau du lancer de rayons et devient rapidement impossible à construire. Il est donc impératif de prévoir une autre optimisation qui puisse prendre le relai dans les cas qui ne sont pas traités.

Enfin cette optimisation étant essentiellement basée sur les englobants des objets, il est possible de l'appliquer sur des types d'objets quelconques.

Chapitre 4

Etude des performances

1. Introduction.

De toutes les étapes qui composent l'élaboration d'une technique nouvelle, la réalisation pratique est souvent la plus cruciale. Elle permet souvent de révéler des aspects ignorés lors de l'étude théorique, ou de mettre en évidence les conséquences réelles de détails négligés, ou pire encore, de montrer l'infaisabilité du projet. C'est l'expérimentation qui permet d'estimer dans quelle mesure la proposition est valide. C'est d'autant plus vrai en informatique où la conception des algorithmes en particulier, est faite en faisant abstraction des structures de données et des contraintes matérielles. C'est pourquoi, après avoir développé les spécifications d'un nouveau modèle de visualisation et démontré plusieurs propriétés remarquables, il est indispensable d'implanter les algorithmes décrits dans les chapitres 2 et 3, et de les confronter à des cas réels.

Nous comparons dans un premier temps les performances du lancer de rayons sur un objet modélisé par un arbre CSG, avec celles obtenues sur le même objet mais modélisé par des arbres VSG. Le fait ne n'avoir qu'un seul objet permet d'éviter toutes considérations liées à la complexité de la scène et donc aux éventuelles optimisations utilisées. L'intention est de simplement comparer les deux modèles, en tenant compte dans le cas des arbres VSG des outils spécifiques que sont le test d'appartenance et le classement a priori.

Le mixage d'algorithmes est un aspect important de la synthèse d'images. Dans la recherche du réalisme, les méthodes proposées ne résolvent que partiellement les problèmes posés. Par exemple, lancer de rayons et radiosité sont associés pour obtenir à la fois les effets spéculaires et diffus [SIL 89]. Les méthodes utilisées ne sont parfois pas toujours les mieux adaptées ou les plus performantes. Jahami [JAH 91] utilise l'algorithme d'Atherton pour l'élimination des parties cachées des objets polyédriques qui ne sont ni réfléchissants ni transparents, et le lancer de rayons sur le reste de la scène et pour obtenir les effets de rendu réaliste. Pour l'optimisation du lancer de rayons, la remarque est la même. Les algorithmes proposés ne prennent pas en compte toutes les informations (pas seulement géométriques) contenues dans la scène. Vivian [VIV 93] change de méthode lorsque celle qu'il propose ne permet pas d'optimiser le rayon étudié. Dans la mesure où l'optimisation par classement a priori s'applique dans un cadre plus général qu'au seul modèle VSG, nous comparons cette méthode avec une optimisation classique du lancer de rayons du type subdivision spatiale. L'objectif n'est pas tant de dire qu'une méthode est meilleure que l'autre. Il s'agit plutôt d'étudier les situations qui font que l'une est meilleure que l'autre et d'en déduire des règles qui permettent d'organiser le processus de visualisation en tenant compte de leurs avantages respectifs. Nous montrons également que dans ce contexte, le modèle VSG introduit une notion de souplesse supplémentaire.

2. Comparaison des performances sur les modèles CSG et VSG.

2.1. Conditions de tests.

Les tests présentés dans ce paragraphe ont été réalisés sur une station de travail SUN Sparc 10 équipée du système UNIX. Les temps sont donnés en secondes CPU et correspondent au temps de calcul de l'image entière. La taille des images calculées est de 500*500 pixels pour tous les objets. Les seuls optimisations prises en compte pour accélérer l'évaluation de l'arbre CSG sont celles décrites dans l'algorithme de base [ROT 82]. L'étude du fils droit est fonction du résultat de l'intersection avec le fils gauche. A chaque noeud est associé un englobant parallélépipédique. Un noeud est examiné si le rayon coupe son englobant.

Pour chaque objet, nous donnons dans les tableaux suivants le nombre des éléments qui le composent ainsi que le temps de calcul du passage du modèle CSG au modèle VSG. Afin d'avoir une meilleure idée de la géométrie des objets nous donnons leurs représentations fil de fer d'après les englobants du modèle VSG.

	Pièce	Barillet	Equerre	Rectifieuse
Nombre d'arbres CSG	1	1	1	1
Nombre d'arbres VSG	1	2	11	18
Nombre de primitives	13	21	24	40

Tableau 1 : Composition des objets.

Pièce	Barillet	Equerre	Rectifieuse
0.01	0.012	0.012	0.013

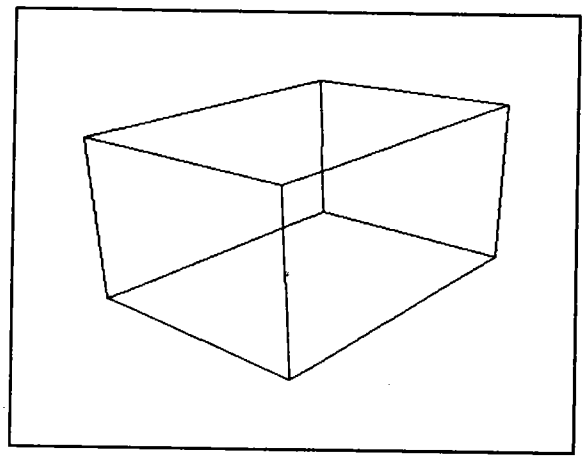
Tableau 2 : Temps de passage du modèle CSG au modèle VSG.

Le tableau 1 montre l'influence de la normalisation sur la composition des objets. On peut constater que l'équerre et la rectifieuse sont composées de nombreux éléments indépendants. Le tableau 2 montre que cette transformation est peu coûteuse puisqu'elle correspond à moins d'un millième de du temps total du calcul de l'image (cf. §2.2).

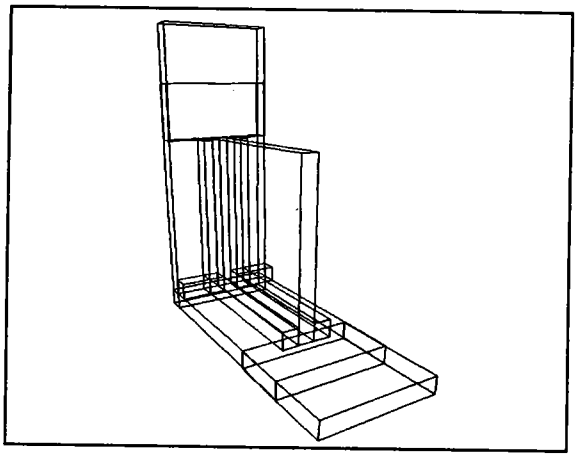
La scène étant composée d'un seul objet, le problème de la sélection des objets à étudier ne se pose pas. Les temps obtenus permettent bien de comparer les performances du calcul d'intersection rayon/arbre CSG et rayon/arbre VSG. Nous n'avons donc calculé que les rayons primaires. Les conclusions que nous en tirons sont valables pour n'importe quel type de rayon.

Pour éviter de lancer des rayons dans les zones vides de l'image, les englobants des objets sont d'abord projetés sur l'écran pour reconstruire de nouveaux englobants rectangulaires. Un rayon primaire est lancé pour chaque pixel appartenant à un rectangle.

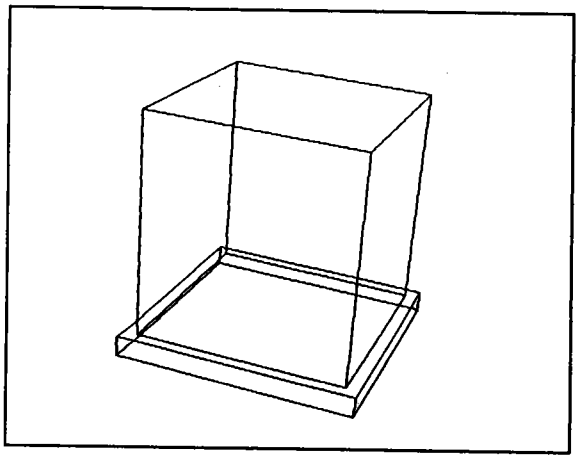
En plus des performances, nous étudions certains chiffres jugés représentatifs des propriétés du modèle VSG.



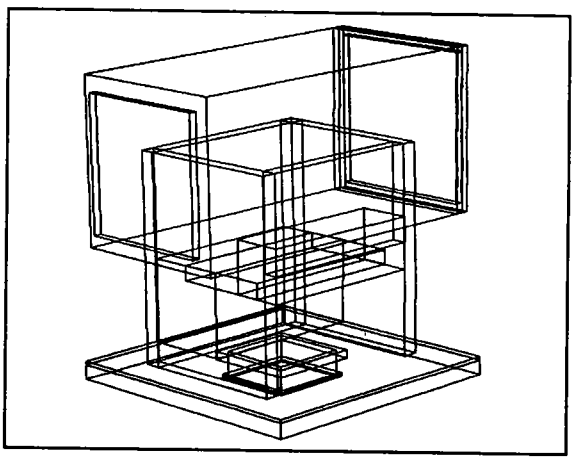
Pièce VSG



Equerre VSG



Barillet VSG



Rectifieuse VSG

2.2. Présentation des résultats.

La pièce est un objet intéressant puisque sa représentation VSG est composée d'un seul arbre. Les transformations ont simplement consisté à réarranger l'arbre CSG initial. La pièce est en fait un bloc percé de 12 trous.

	CSG	VSG
Temps de calcul	43	22
Nb de rayons calculés	154 445	154 445
Nb de primitives étudiées	417 392	233 596
Nb de tests d'appartenance	0	880 280
Nb d'englobants étudiés	1 224 103	0

Tableau 3 : Comparaison CSG/VSG sur la pièce.

Le seul passage au modèle VSG a permis de diviser le temps de calcul par deux. Le nombre total d'intersections rayon/primitive devant être calculées est 154 445*13=2 007 785. Le lancer de rayons classique sur l'arbre CSG réduit ce nombre grâce aux englobants. La représentation VSG cumulée au test d'appartenance est plus efficace. Le nombre d'intersections calculées est divisé par 1,8 alors que le nombre de tests est très inférieur. La structure particulière des arbres VSG est donc un facteur d'accélération important. Les résultats obtenus sur le barillet sont encore plus frappants. C'est un objet relativement voisin de la pièce. Il est composé de l'union de deux cylindres percés de 19 trous.

	CSG	VSG
Temps de calcul	51	20
Nb de rayons calculés	107 604	97 411
Nb de primitives étudiées	812 123	167 644
Nb de tests d'appartenance	0	1 356 857
Nb d'englobants étudiés	1 378 329	0

Tableau 4 : Comparaison CSG/VSG sur le barillet.

Le temps de calcul est divisé par 2,5. Grâce au test d'appartenance, le nombre de primitives étudiées est divisé par 5 par rapport au modèle CSG. La transformation en deux arbres VSG n'a pas vraiment d'influence sinon de réduire de 10 % le nombre de rayons primaires.

L'équerre est beaucoup plus composite que les deux objets précédents. La multiplication des arbres VSG met en évidence une caractéristique importante du modèle. L'augmentation de la complexité engendrée par la normalisation de l'arbre CSG, constitue un avantage dans la mesure où elle est bien traitée. Le choix d'une optimisation destinée à réduire cette complexité devient primordial. C'est dans cette optique qu'a été proposé le classement a priori. D'autres solutions peuvent être mises en oeuvre mais nous ne les discuterons pas ici. Toutefois, s'il en existe de meilleures que le classement a priori, elles ne pourront que favoriser encore l'utilisation du modèle VSG.

	CSG	VSG et classement a priori
Temps de calcul	44	20
Nb de rayons calculés	186 045	106 634
Nb de primitives étudiées	531 156	229 897
Nb de tests d'appartenance	0	210 851
Nb d'englobants étudiés	1 678 778	0

Tableau 5 : Comparaison CSG/VSG sur l'équerre.

Les englobants minimaux des arbres VSG jouent un rôle très important. Leurs projections occupent seulement 57 % de la surface du rectangle englobant de l'arbre CSG. L'accélération obtenue est 2,2. Cependant compte tenu des résultats obtenus sur la pièce et le barillet, on pouvait attendre un écart plus important. Si l'objet est plus composite, chaque arbre VSG contient beaucoup moins de trous puisqu'ils sont composés en moyenne de deux primitives. Par conséquent, le test d'appartenance n'entre pas vraiment en jeu. On peut constater d'ailleurs, que le nombre d'appels au test d'appartenance est peu important par rapport au nombre de primitives de l'objet (21).

L'objet appelé rectifieuse (en fait le bâti d'une rectifieuse de soupapes !) a les mêmes caractéristiques que l'équerre. Il est très composite (18 arbres VSG). Les arbres VSG sont moins répartis et ne réduisent que de 25 % le nombre de rayons calculés. L'accélération est environ égale à 2. Là encore, le nombre moyen de primitives par arbre VSG est deux, ce qui limite l'influence du test d'appartenance.

	CSG	VSG et classement a priori
Temps de calcul	57	29
Nb de rayons calculés	152 064	110 594
Nb de primitives étudiées	843 910	278 371
Nb de tests d'appartenance	0	648 243
Nb d'englobants étudiés	1 763 651	0

Tableau 6 : Comparaison CSG/VSG sur la rectifieuse.

2.3. Conclusion.

Le modèle VSG est très avantageux puisqu'en moyenne les images sont calculées deux fois plus vite. Le coût de la transformation peut être considéré comme négligeable devant le temps de calcul d'une image. Il représente moins d'un millième du temps total. Ces tests ont montré

également l'influence de certaines propriétés remarquables de ce modèle de visualisation. Les englobants minimaux des arbres VSG permettent de réduire parfois considérablement le nombre de rayons primaires étudiés. Le test d'appartenance évite de nombreux calculs d'intersection avec les primitives. Son efficacité varie avec la quantité de trous que contient l'objet. Enfin, la transformation d'un arbre CSG peut produire un ou plusieurs arbres VSG. L'utilisation du classement a priori permet d'utiliser ce découpage pour organiser la visualisation d'une manière plus cohérente.

3. Comparaison des performances obtenues par un classement a priori des objets et par une subdivision spatiale.

3.1. Présentation des optimisations.

3.1.1. Optimisation par classement a priori des objets.

Lors de la présentation du classement a priori dans le chapitre 3, nous avons évoqué le problème du choix du plan de subdivision dans la construction de l'arbre BSP. Le nombre de subdivisions a tout intérêt à être réduit au minimum. Une augmentation du nombre de noeuds augmente la taille de la liste de priorités et donc diminue les performances du lancer de rayons.

L'une des stratégies proposées consiste à sélectionner aléatoirement un nombre fixé de candidats, puis de placer à la racine de l'arbre BSP celui qui entraîne le moins de subdivisions. Le temps de construction de l'arbre BSP des englobants étant quasiment négligeable devant le temps de calcul d'une image, nous avons étudié tous les candidats possibles. Afin de montrer l'intérêt de rechercher un meilleur plan, nous avons comparé les résultats obtenus avec ceux de la méthode simple qui consiste à prendre systématiquement le premier candidat.

Le coût de la construction de l'arbre BSP en recherchant le meilleur plan est évidemment plus important que celui de la méthode simple. L'écart varie en fonction du nombre d'englobants. Pour les scènes qui seront présentées dans le paragraphe 3.3.1., les temps sont multipliés par 10 pour les scènes les plus simples (étagère, bride), et par plus de 60 pour la voiture. Pour cette dernière, le temps de construction n'est plus négligeable. En contrepartie, le traitement de la liste de priorités est accéléré. Le temps de calcul des différentes projections est environ divisé par deux. Il est, sur ce point, difficile de définir avec précision l'écart entre les deux méthodes car les temps avoisinent le millième de seconde. Mais le point réellement intéressant apparaît lors du calcul de l'image. La recherche du meilleur plan permet de réduire jusqu'à un tiers (la voiture) les temps de calcul. Là encore, l'écart varie en fonction de la complexité de la scène. Dans tous les cas, il est préférable de dépenser quelques dixièmes de seconde supplémentaires pour affiner la construction de l'arbre BSP, car cela peut se traduire par plusieurs dizaines de secondes en moins lors de l'affichage.

3.1.2. Optimisation par subdivision spatiale.

Le choix d'une décomposition spatiale récursive allait à l'encontre des solutions généralement retenues. En effet, une étude de l'existant tendait à prouver que la solution de l'arbre octal n'était qu'une alternative théorique élégante, difficilement exploitable sans une importante adaptation des algorithmes de visualisation et une restriction sévère des informations contenues dans les cellules [BOU 87]. Le principal reproche fait à cette structure est le coût engendré par l'évaluation obligatoire de l'arbre lors du parcours des différents types de rayon.

Cependant, la similitude avec le classement a priori, dans l'analyse des objets, permet de comparer objectivement les deux approches.

Pour ne pas retrouver les principaux handicaps d'une décomposition par arbre octal, nous avons mis en oeuvre un ensemble d'optimisations et d'adaptations qui affine l'analyse et simplifie le contenu de chacune des cellules de l'arbre.

3.1.2.1. Décomposition spatiale.

L'évaluation de l'arbre octal s'avérant, comme pour l'arbre CSG, très pénalisante lors de la traversée de la structure par un rayon, il était intéressant de pouvoir lui associer une structure plus simple sur laquelle nous puissions appliquer des algorithmes de traversée incrémentaux. Pour cela nous avons transformé et adapté l'algorithme de Bouatouch, Madani, Priol et Arnaldi [BOU 87] développé pour une décomposition non régulière.

Le principe est de ne pas utiliser l'arbre octal mais une structure de type grille 3D qui sera en fait une transformation de l'arbre octal. Pour cela on définit un tableau à trois dimensions dont la taille de la cellule de référence sera la même que celle de la plus petite cellule de l'arbre (voir fig. 1). Chaque feuille est numérotée. Ce numéro est reporté pour toutes les cellules de la grille se référant à une même feuille de l'arbre. C'est la seule information qui sera rangée dans les cellules de la grille.

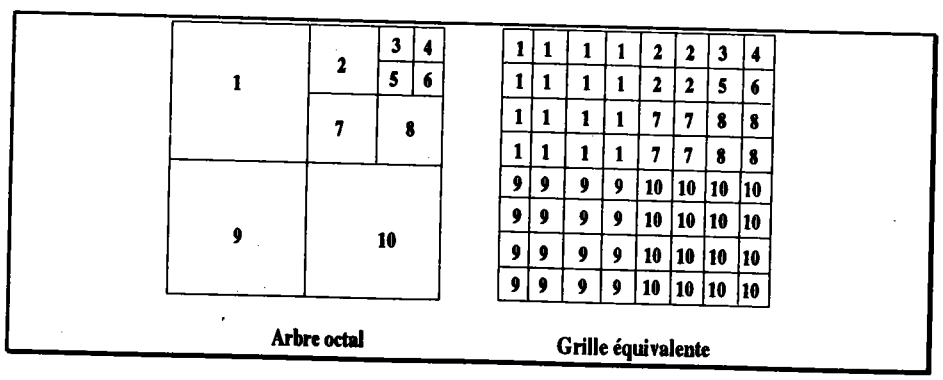


Figure 1 : Passage de l'arbre octal à la grille 3D.

Nous obtenons ainsi une structure dont les éléments de base ont tous la même taille alors que le contenu de ces différentes cases est représentatif d'une décomposition récursive par principe irrégulière. Nous pouvons alors appliquer sur la grille équivalente un algorithme incrémental de traversée.

Pour ne pas à avoir à évaluer la totalité de l'arbre octal à chaque rayon, les listes d'objets associées à chacune des feuilles sont rangées séquentiellement dans un tableau (voir fig. 2).

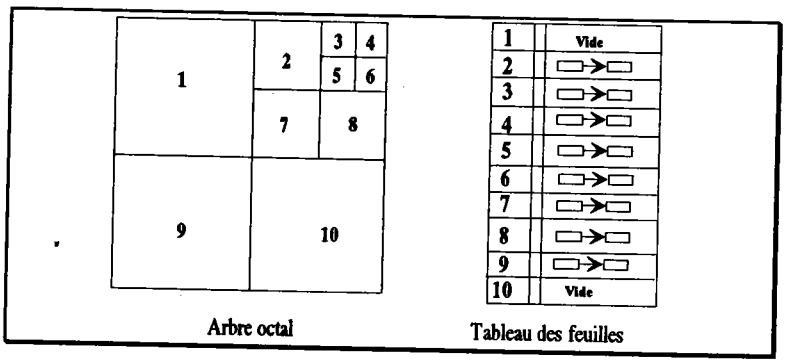


Figure 2 : Tableau des feuilles de l'arbre octal.

On accède à la liste des objets du noeud N en utilisant ce nombre comme indice du tableau. L'arbre octal n'est plus évalué et peut être supprimé. Tous les calculs s'effectuent sur la grille 3D et le tableau contenant la liste des objet (Tableau des feuilles). L'analyse de la traversée est très efficace puisque le déplacement est calculé de manière incrémentale et l'accès aux objets est direct (voir fig. 3).

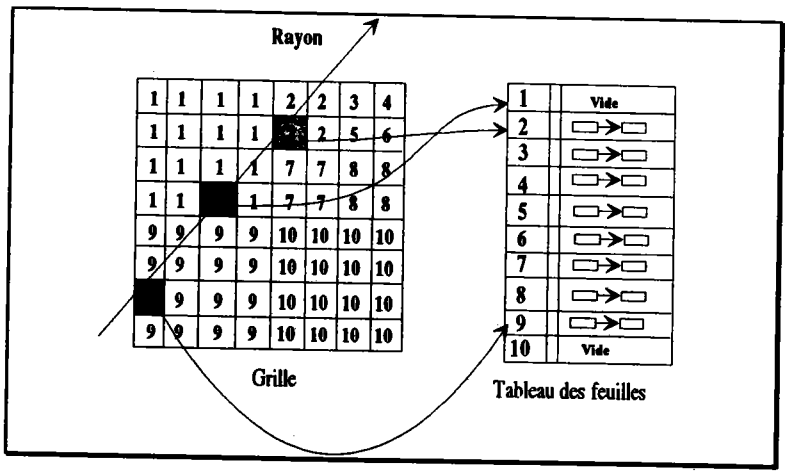


Figure 3 : Traversée de la grille. En fait, seules trois cellules (9, 1 et 2) seront examinées.

3.1.2.2. Optimisation de la décomposition.

Afin de réduire le contenu des cellules, à la décomposition ont été associées plusieurs techniques propres à la modélisation CSG. La recherche du contenu des cellules s'appuie sur les englobants des primitives. Il est donc essentiel qu'ils soient le plus précis possible, voire même minimaux. Nous avons donc mis en oeuvre la méthode de raffinement proposée par Cameron [CAM 92], décrite dans le chapitre 1 (cf. 4.2.2.). Cette optimisation ne s'applique que dans le cas d'une représentation CSG de la scène. L'une des propriétés importantes du modèle VSG est d'être approché naturellement par l'englobant minimal. Une autre optimisation consiste à restreindre l'arbre CSG initial à partir des primitives contenues (complètement ou partiellement) dans la cellule. Nous ne décrivons pas cette technique qui a déjà été détaillée dans le chapitre 1 (cf. 4.2.1.).

3.1.2.3. Traversée de la grille.

Nous avons repris et adapté un algorithme de traversée proposé par Amanatides et Woo [AMA 87]. Cet algorithme est plus complet qu'un algorithme de discrétisation de segment appliqué au 3D. En effet, il examine toutes les cases effectivement traversées par le rayon. Variation dépouillée de l'algorithme de Fujimoto [FUJ 86], il présente l'avantage d'une mise en oeuvre plus simple, car aucune case n'est déduite d'une trajectoire initiale.

On appelle X, Y et Z les coordonnées du point d'entrée du rayon dans une case de la structure régulière. XSortie, YSortie et ZSortie sont les coordonnées de sortie du rayon de la structure. L'équation du rayon est de la forme $\vec{u} + t\vec{v}$ pour $t > 0$. StepX, StepY et StepZ sont initialisés à 1 ou -1 suivant l'orientation du rayon.

A partir du point d'entrée, on recherche l'intersection du rayon avec la première frontière verticale du voxel. Cette valeur est rangée dans la variable TMaxX (elle est exprimée en valeur de t). On effectue le même calcul pour TMaxY et TmaxZ. Le minimum des ces trois valeurs indique la distance que l'on peut parcourir sur le rayon sans changer de voxel.

On détermine enfin, pour les trois directions, des valeurs appelées TDeltaX, TDeltaY et TDeltaZ (toujours en valeur de t) telles que le mouvement suivant ces directions soit égal à une longueur, largeur et profondeur de voxel (pas obligatoirement cubique). L'algorithme s'écrit alors très simplement :

```
Tant que non sorti de la grille Faire
    Si TMaxX < TMAxY Alors
        Si TMaxX < TMaxZ Alors
            X = X + StepX
            Analyse (Case (X,Y,Z)); /* Traitement intersection rayon / Objets */
```

```
                TMaxX = TMaxX + TDeltaX
            Sinon
                Z = Z + StepZ;
                Analyse (Case (X,Y,Z)); /* Traitement intersection rayon / Objets */
                TMaxZ = TMaxZ + TDeltaZ
            Fsi
        Sinon
            si TMaxY < TMaxZ Alors
                Y = Y + StepY;
                Analyse(Case(X,Y,Z)); /* Traitement intersection rayon / Objets */
                TMaxY = TMaxY + TDeltaY;
            Sinon
                Z = Z + StepZ;
                Analyse (Case (X,Y,Z)); /* Traitement intersection rayon / Objets */
                TMaxZ = TMaxZ + TDeltaZ;
            Fsi
        Fsi
    Ftant
```

Nous avons quelque peu modifié cet algorithme pour ne pas à avoir à étudier deux cases portant le même numéro de noeud (voir fig. 4).

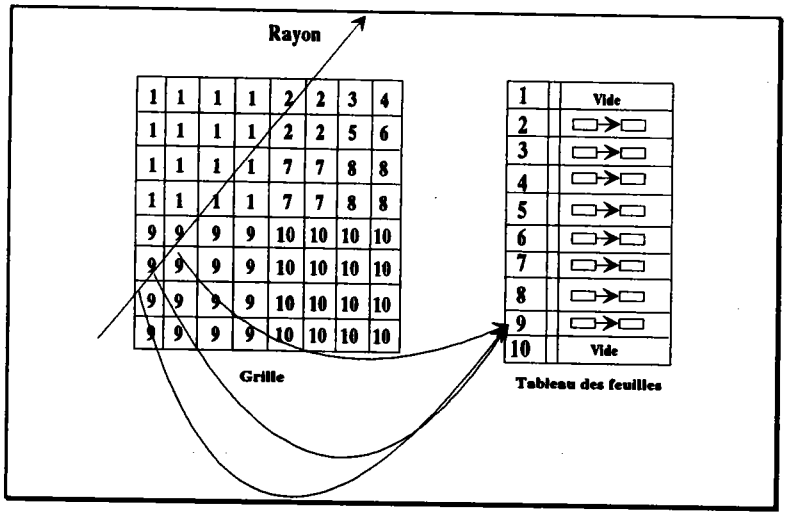


Figure 4 : Optimisation de la traversée.

L'algorithme commence par étudier plusieurs cases ayant le même numéro d'entrée sur la table des objets (9). Si aucune intersection n'est trouvée lors de l'analyse de la première case, aucune autre analyse sur une case portant le même numéro ne pourra retourner une

intersection entre le rayon et la liste des objets de la case. Par conséquent, une cellule n'est étudiée que si aucune autre cellule portant le même numéro ne l'a été.

Il faut prendre certaines précautions lors de la mise en oeuvre de cet algorithme. En effet un point d'intersection n'appartient pas forcement au voxel traversé (voir fig. 5).

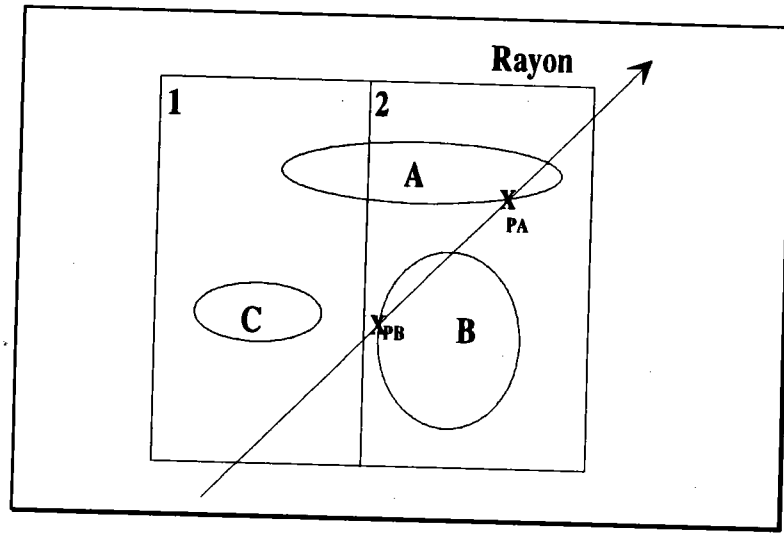


Figure 5 : Lorsqu'un point d'intersection est trouvé, il doit appartenir au voxel étudié.

Lors de l'analyse de la case 1, on recherche une intersection entre le rayon et une liste contenant les objets C et A. On trouve pour A un point d'intersection PA. Or ce point n'appartient pas au voxel étudié. Il ne doit pas être pris en compte car d'autres objets en dehors du voxel peuvent se situer devant PA (B par exemple). Cette remarque correspond à celle faite pour le classement a priori où un point d'intersection n'est valide que s'il appartient à l'englobant considéré (chap. 3 §3.1.).

3.2. Approche théorique de la complexité des deux méthodes.

Dans le cadre de cette approche théorique, nous ne cherchons pas à déterminer le coût exact des optimisations. Une étude de ce type a été réalisée par Devillers [DEV 88] sur les subdivisions spatiales. Nous souhaitons simplement comparer leurs comportements, voir ce qui conditionne leurs performances. A travers des algorithmes simplifiés, nous étudions le lancer d'un rayon quelconque et dénombrons les opérations effectuées.

3.2.1. Optimisation par classement a priori des objets.

Nous avons décrit dans le chapitre 3, un algorithme recherchant un point d'intersection entre un rayon et une liste ordonnée d'objets et ce pour les trois types de rayon (primaires,

d'ombrage et secondaires). Son principe consiste à étudier les objets dans l'ordre de la liste de priorités. L'origine du rayon étant projeté sur le plan (de l'écran pour les rayons primaires, virtuels dans les autres cas), ce point est comparé aux rectangles englobants des objets. S'il y a appartenance, l'intersection est calculée entre le rayon et l'objet correspondant. Dans la mesure où nous n'étudions qu'un rayon qui est le même pour les deux optimisations, les circonstances qui stoppent son parcours sont supposées être les mêmes.

L'algorithme simplifié est le suivant :

```
O=premier objet de la liste
Tantque étude du rayon R non terminée Faire
        Si R appartient au rectangle englobant de O Alors
                · Recherche une intersection entre R et O
        Fsi
        O=Objet suivant
Ftant
```

Cet algorithme met en oeuvre deux types d'opération, la recherche d'intersection entre un rayon et un objet, et le test d'appartenance d'un rayon dans un rectangle englobant. Afin de simplifier le raisonnement, on associe à chacune d'entre elles un temps d'exécution constant. Soit Δi le coût d'une recherche d'intersection et Δt le coût du test d'appartenance à un rectangle englobant.

On peut distinguer deux catégories d'objets :

1) les objets pour lesquels, après avoir testé l'appartenance au rectangle, on recherche effectivement une intersection. On note p leur nombre.

2) les objets ayant été écartés par le test d'appartenance. Soit e leur nombre. $p+e$ est égal au nombre d'objets dans la scène.

On en déduit que le temps Δr consacré à l'étude d'un rayon dans la méthode du classement a priori s'écrit :

$$\Delta r = p\Delta i + (p+e)\Delta t \quad (1)$$

3.2.2. Optimisation par subdivision spatiale.

L'algorithme simplifié du lancer d'un rayon dans une subdivision spatiale régulière est le suivant :

C = rechercher la première cellule traversée
Tantque étude du rayon R non terminée **Faire**
Pour chaque objet O de C **Faire**
Recherche une intersection entre R et O
Fpour
C = rechercher la cellule suivante
Ftant

Dans cet algorithme, les deux opérations mises en oeuvre sont la recherche d'une intersection entre un rayon et un objet, et la recherche d'une cellule. On associe à cette dernière un coût constant Δc.

Le lancer d'un rayon va consister à rechercher une intersection avec les objets contenus dans les m cellules traversées. On note k la somme ce ces objets. Le temps $\Delta r'$ consacré à l'étude d'un rayon optimisé par une subdivision spatiale régulière s'écrit de la manière suivante :

$$\Delta r' = k\Delta i + m\Delta c \qquad (2)$$

3.2.3. Comparaison théorique des deux méthodes.

Pour un rayon donné, le temps théorique qui lui est consacré est :

$\Delta r = p\Delta i + (p+e)\Delta t$ pour le classement a priori des objets et

$\Delta r' = k\Delta i + m\Delta c$ pour la subdivision spatiale.

Or, on peut supposer que les p objets se trouvant sur la trajectoire du rayon pour le classement a priori le seront aussi pour la subdivision. Ils se retrouveront donc dans les m cellules traversées. D'où $k = p+q$. Le nombre q représente les objets contenus dans les cellules traversées mais ne se trouvant pas directement sur la trajectoire. Il dépend de la taille des cellules. Lorsqu'elle tend vers 0, q tend également vers 0 et p ne varie pas. En revanche m tend vers l'infini. Par conséquent on peut prévoir qu'il existe une taille optimale telle qu'en dessous, le coût de la traversée des cellules supplémentaires (même si elles sont vides) compense les gains obtenus par la diminution de q.

Finalement, en remplaçant k dans (2) on obtient :

$$\Delta r' = p\Delta i + q\Delta i + m\Delta c$$

soit $\Delta r' = \Delta r - (p+e)\Delta t + q\Delta i + m\Delta c$

On en déduit que l'optimisation par classement a priori donne de meilleures performances que la subdivision lorsque :

$$(p+e)\Delta t < q\Delta i + m\Delta c$$

Un nombre peu important d'objets devrait donc favoriser le classement a priori car le test d'appartenance à un rectangle englobant est nettement moins coûteux qu'un calcul d'intersection ($\Delta t << \Delta i$). En revanche si la complexité de la scène augmente, $p+e$ croît de manière identique alors que la subdivision récursive permet de réduire le nombre q. Par conséquent, en faisant varier le nombre d'objets, on devrait observer un seuil en dessous duquel le classement a priori donne de meilleures performances que la subdivision spatiale. Au dessus de ce seuil, le rapport entre les deux méthodes s'inverse. L'étude pratique doit permettre de déterminer sa valeur. Elle constitue alors un critère permettant d'adapter les optimisations en fonction des situations.

3.3. Etude pratique des performances sur des scènes complexes.

3.3.1. Conditions de test.

Les tests présentés dans ce paragraphe ont été réalisés sur une station de travail SUN Sparc 10 équipée du système UNIX. Les temps sont donnés en secondes CPU et correspondent au lancé de 100 000 rayons. La taille des images calculées est de 400*250 pixels pour l'étagère et 500*500 pixels pour les autres scènes. Toutes les scènes sont éclairées par quatre sources lumineuses. Compte tenu des remarques faites sur la difficulté de mettre en oeuvre le classement a priori dans le cas des rayons secondaires, nous ne comparons les deux méthodes que sur un niveau de réflexion.

Pour chacune des scènes, nous donnons dans les tableaux suivants le nombre des éléments qui les composent ainsi que les temps de calcul des différentes structures de données utilisées par les optimisations.

	Etagère	Bride	Axe	Engrenages	Voiture
Nombre d'arbres CSG	6	12	15	10	39
Nombre d'arbres VSG	31	39	54	81	289
Nombre de primitives	88	72	109	273	385

Tableau 7 : Composition des scènes tests.

Etagère	Bride	Axe	Engrenages	Voiture
0.008	0.008	0.003	0.02	0.06

Tableau 8 : Temps de passage du modèle CSG au modèle VSG.

	Etagère	Bride	Axe	Engrenages	Voiture
Calcul de la grille 3D	0.003	0.001	0.003	0.006	0.36
Calcul de l'arbre BSP	0.001	0.008	0.01	0.005	0.05
Calcul de la proj. primaire	0.001	0.001	0.002	0.001	0.003
Calcul des proj. d'ombrage	0.001	0.003	0.006	0.001	0.01
Calcul des proj. secondaires	0.008	0.02	0.01	0.005	0.01

Tableau 9 : Temps de calcul des structures de données pour le modèle CSG.

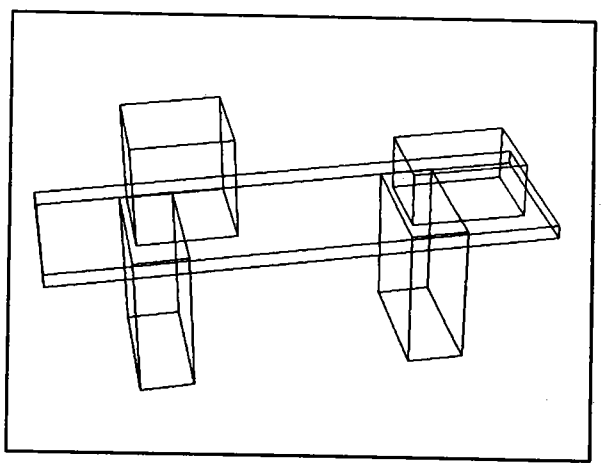
	Etagère	Bride	Axe	Engrenages	Voiture
Calcul de la grille 3D	0.02	0.03	0.04	0.05	0.1
Calcul de l'arbre BSP	0.04	0.09	0.2	0.6	3.8
Calcul de la proj. primaire	0.001	0.003	0.006	0.02	0.03
Calcul des proj. d'ombrage	0.007	0.01	0.03	0.09	0.15
Calcul des proj. secondaires	0.03	0.04	0.07	0.16	0.43

Tableau 10 : Temps de calcul des structures de données pour le modèle VSG.

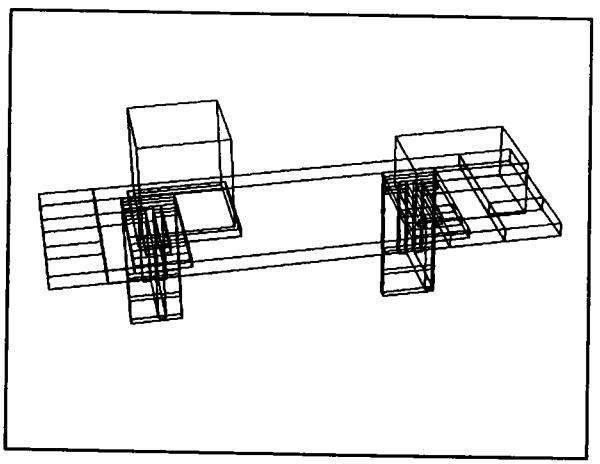
Le temps de calcul des différentes structures de données sont nettement plus importants dans le cas de la représentation VSG. Cet accroissement correspond à l'augmentation de la complexité de la scène. Cependant, dans tous les cas, ils sont très inférieurs au temps de calcul d'une image. L'écart est tel qu'ils peuvent être considérés comme négligeables. De plus, la décomposition spatiale ainsi que l'arbre BSP sont indépendant de l'observateur et ne sont donc calculés qu'une seule fois.

Nous présentons dans le paragraphe 3.3.2. des résultats concernant la subdivision spatiale utilisée. Nous présentons ensuite les différents tests effectués sur les deux méthodes d'optimisation. Le paragraphe 3.3.3. compare leurs performances réelles sur le modèle CSG. Le paragraphe 3.3.4. les compare sur le modèle VSG. Enfin dans le paragraphe 3.3.5. nous faisons une synthèse à partir de certains résultats et en déduisons des règles de mixage des algorithmes et des modèles de visualisation.

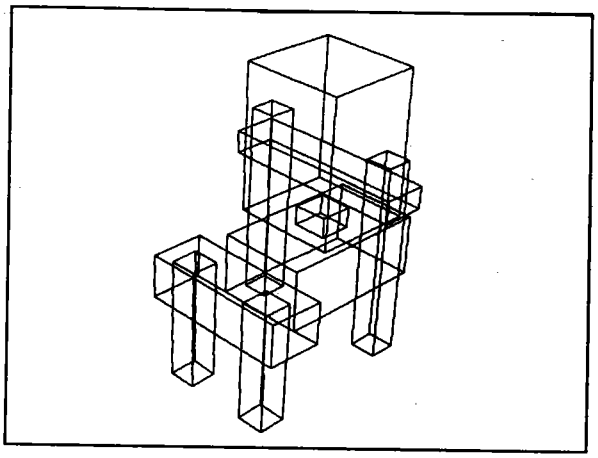
Les graphiques représentant les différents tests sont à examiner en tenant compte de deux remarques. D'une part, le type de graphique choisi est la courbe. Pour des raisons de présentation, l'échelle de l'axe des abscisses n'est pas respectée. Par conséquent, les courbes sont déformées par rapport à la réalité. De plus le nombre de points de passage étant relativement faible (5), elles ne peuvent donner qu'une vision imprécise du comportement des optimisations. D'autre part, les scènes visualisées correspondent à des cas réels. Leur géométrie, tant du point de vue de leur forme que de leur répartition, n'est pas uniforme. Son influence sur les algorithmes mis en oeuvre est difficile à estimer mais n'est pas négligeable. Elle peut notamment expliquer certaines perturbations de la courbe qui semblent anormales. Dans les scènes habituellement utilisées pour valider une optimisation, les objets sont régulièrement répartis, ce qui permet de mieux interpréter les résultats obtenus. Cependant, les conclusions peuvent être faussées par cette régularité. Nous avons préféré utiliser des scènes correspondant à des objets et des situations réelles qui reflètent mieux notre domaine d'application privilégié. De fait, il est un peu plus délicat d'en tirer des conclusions précises. Ces courbes, associée à l'étude théorique, permettent malgré tout de montrer une tendance relativement fiable dans le comportement des deux optimisations testées. Pour avoir un meilleur aperçu des scènes, nous en donnons la représentation fil de fer d'après leurs englobants et pour les deux modèles.



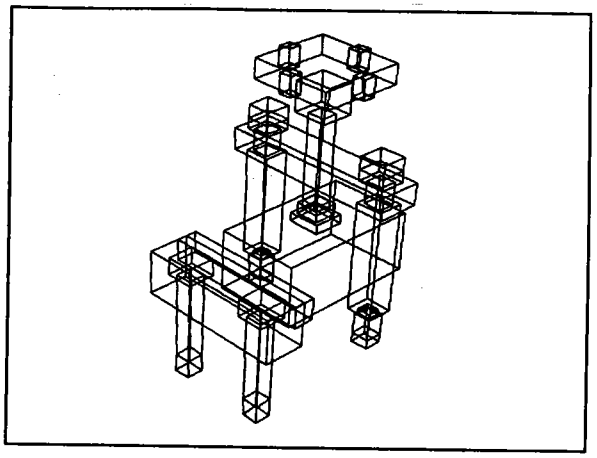
Etagère CSG



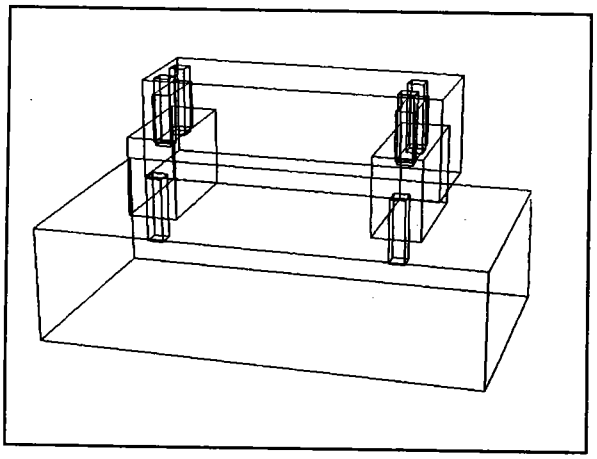
Etagère VSG



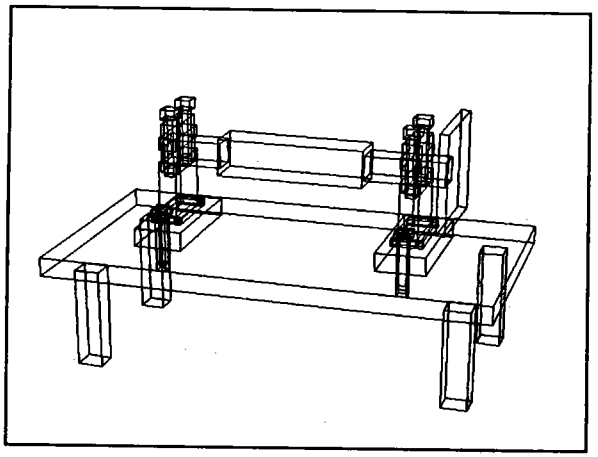
Bride CSG



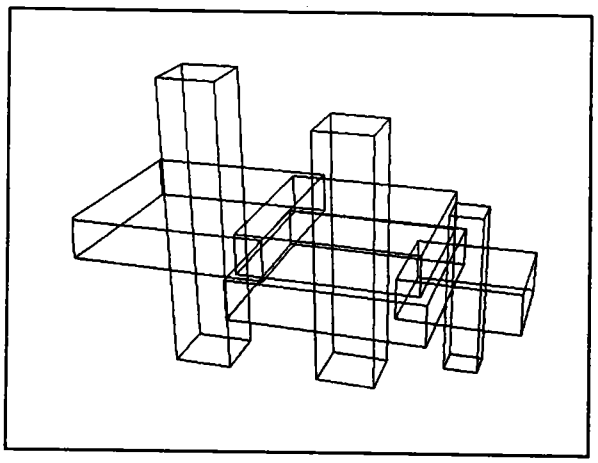
Bride VSG



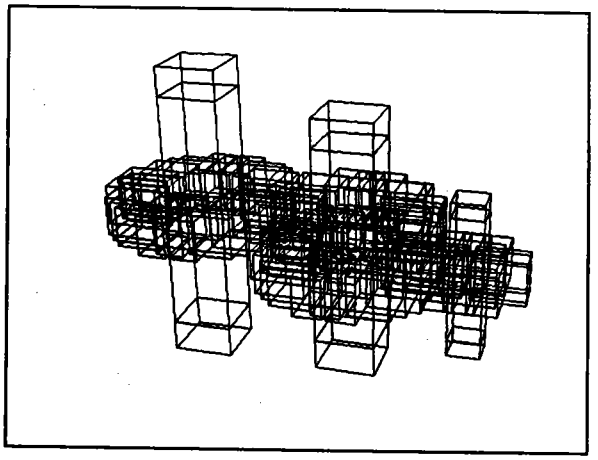
Axe CSG



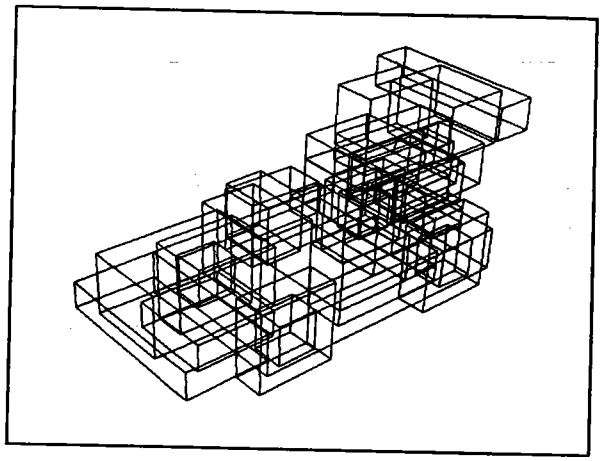
Axe VSG



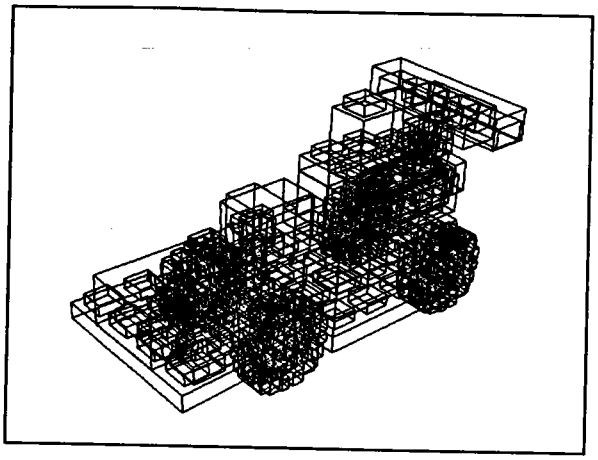
Engrenages CSG



Engrenages VSG



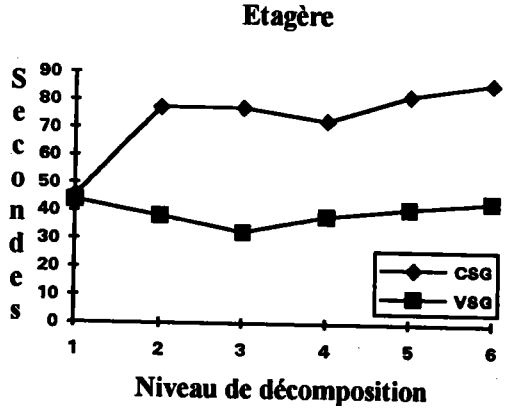
Voiture CSG

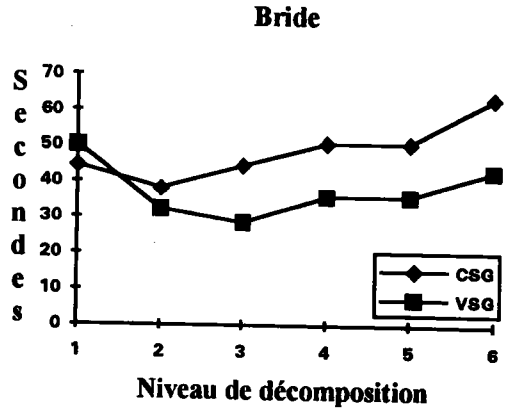


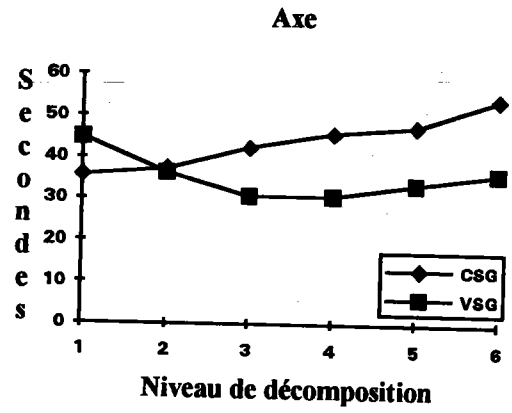
Voiture VSG

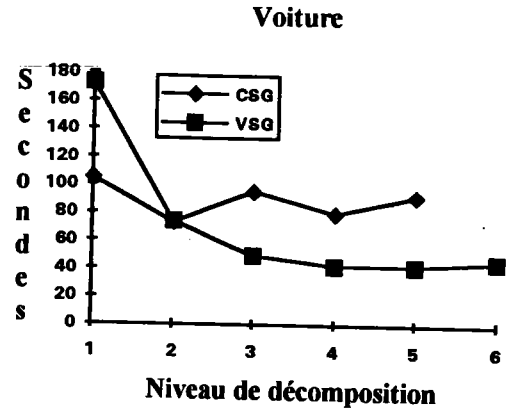
3.3.2. Etude de la meilleure subdivision spatiale.

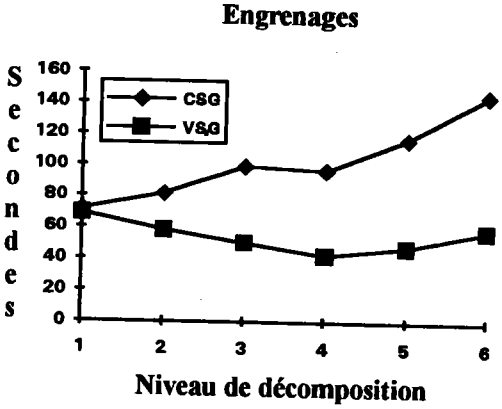
Dans le paragraphe 3.2.3., nous avons supposé l'existence d'un seuil correspondant à la subdivision optimale. Elle permet d'approcher au mieux la scène sans avoir des coûts de traversée trop importants. Dans la méthode implantée (cf. §3.1.2.) la taille des voxels est liée au niveau de décomposition de l'arbre octal initial. Nous avons donc recherché pour chacune des cinq scènes le niveau de décomposition idéal. Les temps indiqués dans les tableaux qui suivent correspondent au lancer des rayons primaires. L'étude a été menée sur les arbres CSG et sur les arbres VSG.











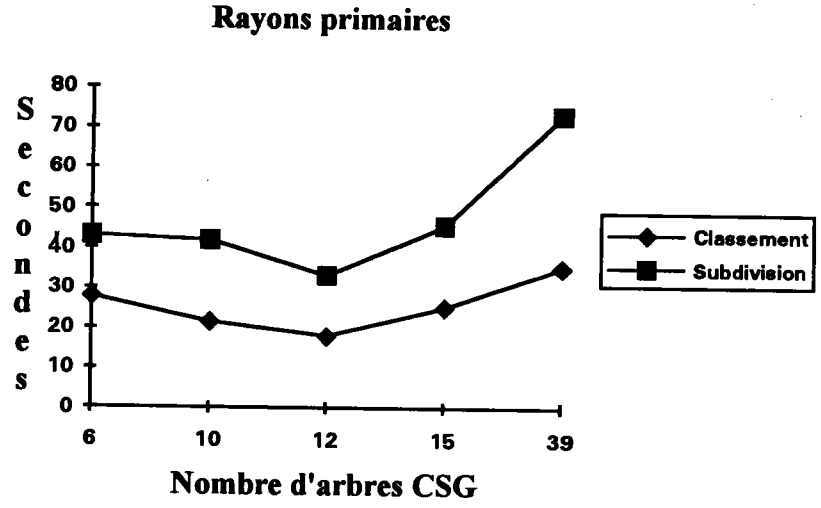
En ce qui concerne le modèle CSG, le niveau de décomposition idéal est soit 1 soit 2 pour toutes les scènes. Un niveau de décomposition égal à 0 équivaudrait à une optimisation par les englobants. La subdivision ne présente que peu d'intérêt en tout cas pour ces scènes tests. Ce phénomène s'explique essentiellement par la répartition des objets. On remarque sur les représentations fil de fer des englobants que les objets sont regroupés au centre de la scène et enchevêtrés les uns dans les autres. C'est pourquoi la subdivision ne permet que d'isoler des parties vides de la scène. Les voxels contenant des objets, en contiennent pratiquement tous le même nombre. La complexité de la scène n'est pas réduite contrairement à l'objectif visé.

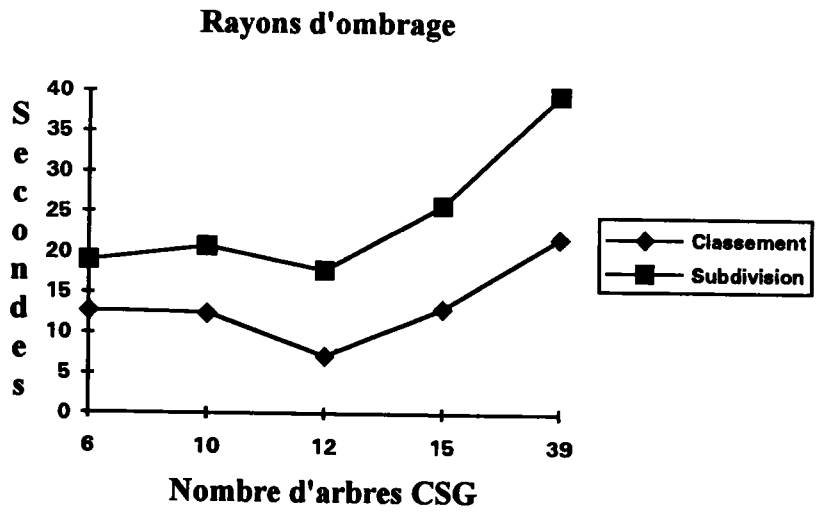
Le passage du modèle CSG au modèle VSG augmente de manière significative (x3 à x8) la complexité de la scène. De plus les objets manipulés sont de taille plus réduite par rapport aux arbres CSG initiaux et semblent un peu plus répartis. Il paraît donc logique que le niveau de décomposition idéal soit supérieur à celui obtenu pour le modèle CSG. Pour les cinq scènes testées, la valeur trouvée est soit 3 soit 4.

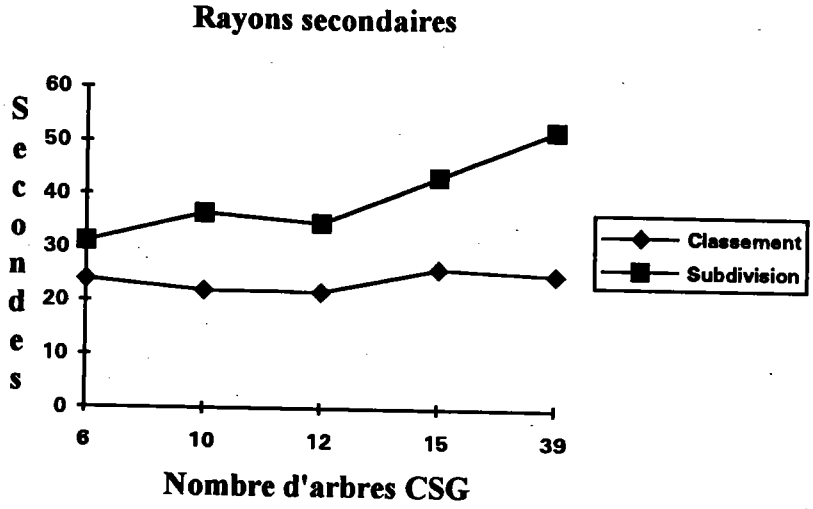
On peut supposer que la valeur optimale trouvée pour les rayons primaires est la même pour tous les types de rayons. Les tests réalisés dans les paragraphes suivants tiennent compte de ces résultats.

3.3.3. Comparaison des performances sur le modèle CSG.

Nous ne donnons ici que les courbes résumant les temps de calcul pour les scènes modélisées par des arbres CSG. Les tableaux contenant les chiffres exacts sont donnés en annexes B.





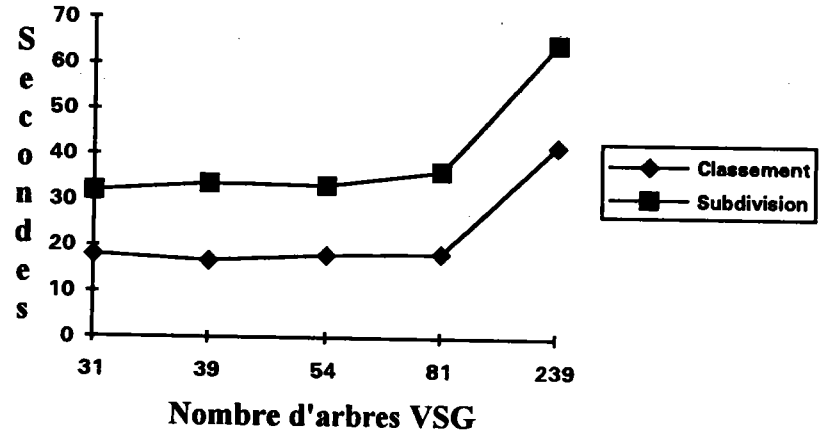


Ces courbes montrent que quel que soit le type de rayons, les deux méthodes ont pratiquement le même comportement. Elles semblent relativement parallèles, ce qui va à l'encontre de l'hypothèse faite dans le paragraphe 3.2.3.. Mais cette remarque est à pondérer avec le fait que le nombre d'arbres CSG est peu élevé (de 6 à 39). Par conséquent, le coût lié à la gestion des structures de données est quasiment négligeable par rapport au temps consacré au calcul des intersections. Ces courbes ne caractérisent qu'un intervalle qui n'est peut-être pas représentatif de leurs formes générales. Cependant, dans la mesure où la complexité de la scène se situe dans cet intervalle, l'écart moyen entre les deux méthodes est relativement stable. Le classement a priori est en moyenne 1.8 fois plus rapide que la partition spatiale dans le cas des rayons primaires, 1.9 fois pour les rayons d'ombrage et 1.7 fois pour les rayons secondaires.

3.3.4. Comparaison des performances sur le modèle VSG.

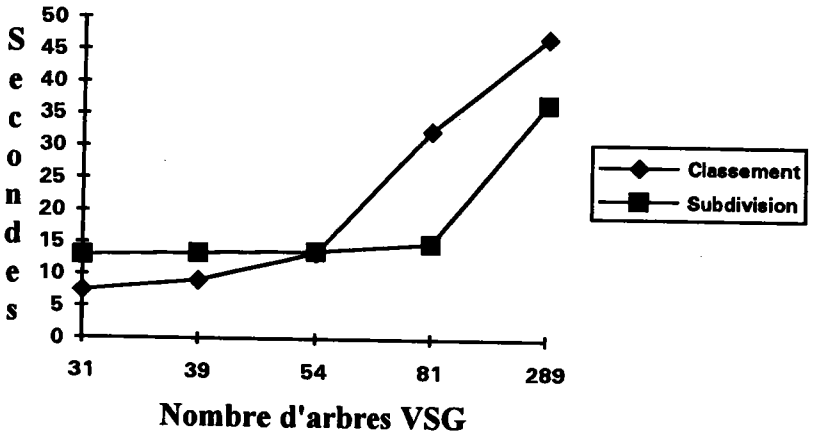
En utilisant cette fois le modèle VSG, nous avons recalculé les images avec les mêmes paramètres que pour le modèle CSG. Les résultats visuels sont évidemment identiques. Cependant, les courbes illustrant les performances obtenues sont sensiblement différentes.

Rayons primaires



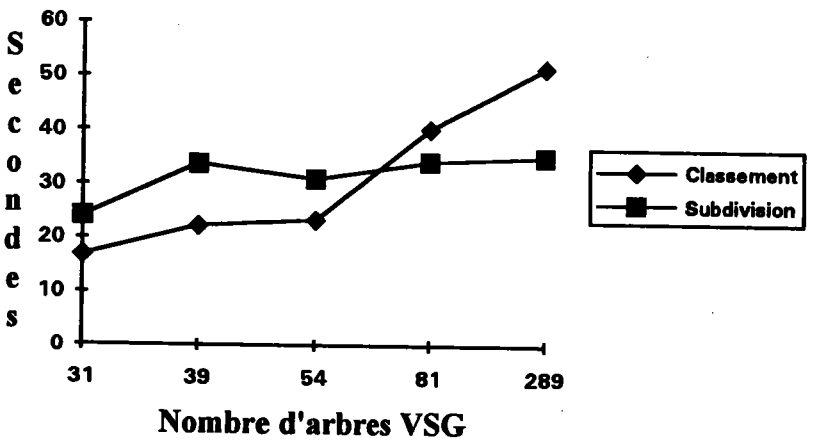
Nombre d'arbres VSG

Rayons d'ombrage



Nombre d'arbres VSG

Rayons secondaires



Nombre d'arbres VSG

Dans le cas du modèle VSG, l'intervalle couvert par les cinq scènes est beaucoup plus large (de 31 à 289). On constate cette fois que dans le cas des rayons d'ombrage et secondaires les courbes ne sont plus parallèles mais se croisent. Elles confirment donc

l'hypothèse du paragraphe 3.2.3.. A partir d'un nombre d'objets environ égal à 50, l'optimisation par partition spatiale est plus avantageuse qu'un classement a priori.

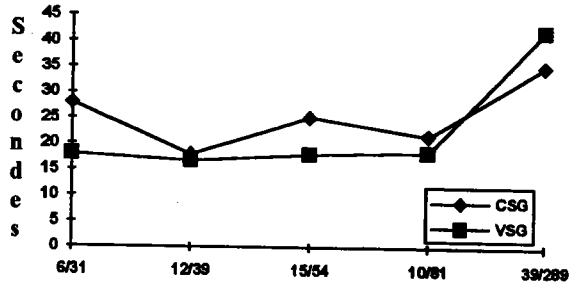
Dans le cas des rayons d'ombrage et secondaires, la liste des objets est étudiée rayon par rayon. Par contre, dans le cas des rayons primaires, elle est étudiée objet par objet et est réduite d'un élément à chaque étape (cf. chap. 3 §3.1.). Il ne correspond pas précisément à l'algorithme servant de base à l'étude théorique. On ne peut donc pas en tirer les mêmes conclusions. L'implantation spécifique aux rayons primaires de l'algorithme explique la différence d'aspect des courbes. Elles restent parallèles quel que soit le nombre d'objets contenus dans la scène. L'écart moyen est le même que dans le cas du modèle CSG c'est-à-dire 1,8.

En conclusion, quel que soit le modèle choisi, l'optimisation la plus performante pour les rayons primaires est le classement a priori. Pour les rayons d'ombrage et secondaires, les performances des deux optimisations comparées dépendent de la complexité de la scène. En réalité, la meilleure solution consiste à choisir le meilleur algorithme d'après les caractéristiques de la scène. Notre étude a mis en évidence un des critères de choix possible. Elle a permis de déterminer empiriquement un nombre d'objets en dessous duquel il est préférable d'utiliser le classement a priori plutôt que la subdivision spatiale. Nous montrons dans le paragraphe suivant qu'il permet également de déterminer le type de représentation le plus adapté.

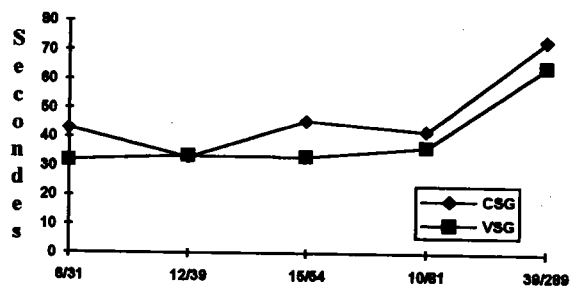
3.3.5. Comparaison des performances entre les deux modèles.

Nous avons montré que pour le modèle VSG, le choix de l'optimisation dépend de la complexité de la scène. On peut se demander alors si le choix du modèle de visualisation n'en dépend pas non plus. Nous allons donc comparer les temps de calcul obtenus pour les modèle CSG et VSG par les deux méthodes d'optimisation.

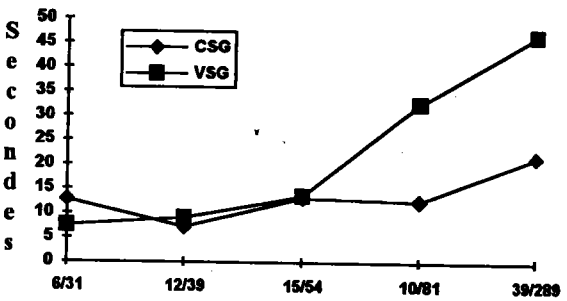
Rayons primaires et classement a priori



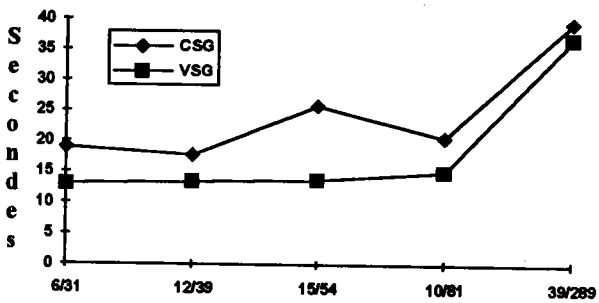
Rayons primaires et subdivision spatiale



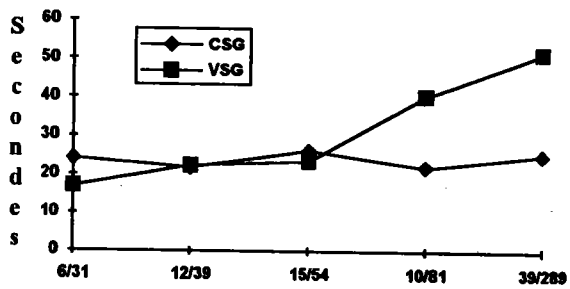
Rayons d'ombrage et classement a priori



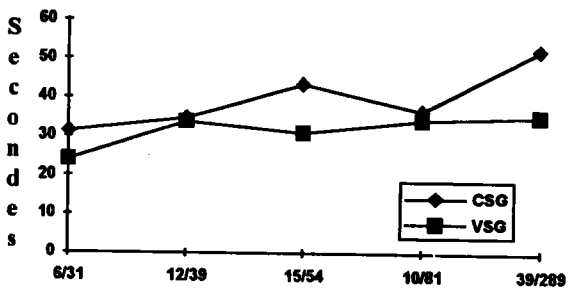
Rayons d'ombrage et subdivision spatiale



Rayons secondaires et classement a priori



Rayons secondaires et subdivision spatiale



Les arbres VSG s'adaptent bien à la subdivision spatiale puisque dans tous les cas les performances obtenues sont meilleures. En revanche, dans le cas du classement a priori on observe un basculement très net dans le cas des rayons d'ombrage et secondaires. Or, si l'on observe ces courbes conjointement avec celles comparant la subdivision spatiale et le classement a priori sur le modèle VSG, on constate que ce basculement correspond au croisement déjà constaté. Ce qui signifie que lorsque la complexité dépasse une certaine limite, la subdivision spatiale devient plus performante que le classement a priori sur les arbres VSG mais dans le même temps cette représentation devient pénalisante par rapport aux arbres CSG. Ce seuil indique non seulement lorsqu'il faut changer d'optimisation mais également lorsqu'il faut changer de modèle. On voit aussi apparaître un seuil pour les rayons primaires. On pourrait donc résumer l'ensemble de ces observations par l'algorithme suivant :

```
/* Recherche de l'optimisation et du modèle de visualisation   */
/* pour les rayons primaires                                   */
        OptimPrimaire = Classement a priori
        Si NbArbresVSG<SeuilPrimaire Alors
                ModèlePrimaire = VSG
        Sinon
                ModèlePrimaire = CSG
        Fsi

/* Recherche de l'optimisation et du modèle de visualisation   */
/* pour les rayons d'ombrage et secondaires                    */
        Si NbArbresVSG<SeuilSecondaire Alors
                ModèleSecondaire = ModèleOmbrage = VSG
                OptimSecondaire = OptimOmbrage = Classement a priori
        Sinon
                Si NbArbresCSG<SeuilSecondaire Alors
                        OptimSecondaire = OptimOmbrage = Classement a priori
                        ModèleSecondaire = ModèleOmbrage = CSG
                Sinon
                        OptimSecondaire = OptimOmbrage = Subdivision spatiale
                        ModèleSecondaire = ModèleOmbrage = VSG
                Fsi
        Fsi
```

3.3. Conclusion.

L'étude pratique des performances a confirmé les prévisions faites lors de l'étude théorique. Pour un petit nombre d'objets le classement a priori donne de meilleurs résultats que la subdivision spatiale. Lorsque la complexité de la scène augmente, on voit l'écart se réduire pour finalement devenir à l'avantage de la deuxième méthode. La solution idéale n'est pas d'utiliser toujours le même algorithme, mais de choisir éventuellement pour chaque rayon, la méthode la plus performante ou la plus adéquate. On remarque également que cette même complexité influence le choix du type de représentation. Nous avons montré que la complexité de la scène permet, dans le contexte de notre application et du type de scène que nous visualisons, de déterminer de manière relativement sûre l'association

optimisation/modèle la plus rapide. On pourrait encore affiner ce choix en étudiant d'autres facteurs comme le rapport $\frac{\text{Nb arbres VSG}}{\text{Nb arbres CSG}}$ ou la répartition des objets.

Cette étude a confirmé l'intérêt d'une approche adaptative de la visualisation. Celle que nous proposons ne repose pas sur un modèle et un algorithme unique. Une analyse "intelligente" de la scène à partir de critères bien déterminés doit permettre d'adapter en plus des algorithmes de visualisation, la représentation de la scène. Le modèle VSG permet alors de parvenir à cet objectif.

Conclusion.

Le modèle CSG est l'un des modèles géométriques les plus répandus en CAO. Cependant, le fait de ne pas décrire explicitement les frontières des objets implique une évaluation plus ou moins complète du modèle qui pénalise les opérations de calcul et notamment la visualisation. La réduction des temps de calcul est encore une préoccupation importante des systèmes de CFAO.

Nous avons présenté une approche originale pour améliorer la visualisation en général et le lancer de rayons en particulier. Nous avons étudié ce problème en partant de la source c'est-à-dire de la modélisation. En nous plaçant dans le cadre d'une approche multimodèles des systèmes de CFAO, nous avons proposé une nouvelle représentation appelée VSG (Visual Solid Geometry). La construction des arbres VSG repose sur deux étapes fondamentales. La normalisation permet de décomposer un arbre CSG en une liste de termes qui, du point de vue de la visualisation, peuvent être considérés séparément. Chaque terme est ensuite réorganisé pour obtenir une forme caractéristique. Un arbre VSG est une différence entre une intersection n-aire de primitives et une union n-aire de primitives. Nous avons montré que le modèle VSG inclut ou facilite naturellement certaines optimisations classiques du lancer de rayons. De plus, les propriétés liées à cette représentation permettent la mise en oeuvre de nouveaux outils qui améliorent l'efficacité de cet algorithme. L'un de nos objectifs est de montrer que le modèle VSG peut trouver des applications intéressantes dans d'autres domaines tels que la détection des collisions entre objets ou la détection de certaines caractéristiques de forme.

Nous avons également proposé une méthode d'optimisation du lancer de rayons pour utiliser au mieux les caractéristiques du modèle de visualisation. Elle s'appuie sur la notion d'arbre BSP et de classement a priori des objets. L'utilisation des projections a cependant imposé des limites importantes dans l'application de cette méthode aux rayons secondaires.

La réalisation de ces différentes propositions a montré leur intérêt réel sur plusieurs aspects. Les transformations qui permettent de passer du modèle CSG au modèle VSG sont négligeables devant le temps de calcul d'une image. L'intersection entre un rayon et un objet est environ deux fois plus rapide lorsque ce dernier est représenté par des arbres VSG. Les arbres VSG non seulement sont compatibles avec les optimisations existantes comme les subdivisions spatiales, mais augmentent leur efficacité.

Enfin, la comparaison des deux méthodes d'optimisation, classement a priori et subdivision spatiale, a souligné l'intérêt du mixage d'algorithmes. Nous avons montré que les deux solutions ne sont pas concurrentes mais plutôt complémentaires. La complexité de la scène est un facteur déterminant dans le choix de l'optimisation, mais le résultat le plus important est que le mixage s'applique aussi à la modélisation. Ceci est rendu possible grâce au modèle VSG qui,

en restant équivalent au modèle CSG, permet le passage d'un modèle à l'autre sans remettre en cause tout le processus de visualisation.

Nous avons décrit la visualisation comme un processus comprenant deux étapes fondamentales: la modélisation et le rendu. Elles sont souvent réalisées de manière rigide, c'est-à-dire que les objets sont représentés dans un modèle géométrique unique et affichés à partir d'un algorithme unique (ou d'une optimisation unique dans le cas du lancer de rayons). Les travaux présentés dans cette thèse contribuent à l'assouplissement de ce processus en offrant la possibilité d'avoir plusieurs modèles et plusieurs optimisations du lancer de rayons, tout en les faisant cohabiter de façon à accélérer les opérations. Les recherches sur cet aspect sont cependant loin d'être terminées. Les paramètres qui conditionnent l'organisation de ce processus de visualisation sont évidemment beaucoup plus nombreux et complexes que le nombre d'objets contenus dans la scène. La géométrie, c'est-à-dire la forme des objets mais aussi leur répartition, joue un rôle essentiel dans ce domaine. Le mixage des algorithmes et des représentations pose encore de nombreux problèmes que nous nous efforcerons d'étudier.

Annexes

Annexe A.

Nous présentons dans cette annexe les procédures d'intersection rayon/primitive et les procédures de test d'appartenance point/primitive. Dans tous les cas, les primitives sont unitaires et impliquent par conséquent des changements de repère pour l'origine du rayon et le vecteur directeur (qui est unitaire). Les procédures d'intersection ne renvoient pas des points, ce qui nécessiterait un changement de repère inverse, mais des déplacements qui sont identiques dans le repère unitaire et dans le repère du monde. Le nombre des opérations indiqué dans le tableau qui suit chaque procédure est donné à titre indicatif. Il correspond au profil d'exécution dans le pire des cas.

Les procédures décrites sont celles effectivement mises en oeuvre lors des calculs d'image. Pour des raisons de présentation, certains détails ont été volontairement omis. Ces procédures ont été écrites en tenant compte avant tout de la lisibilité. Un certain nombres d'opérations pourrait donc être "économisées" sans toutefois remettre en cause les ordres de grandeur donnés dans le chapitre 2.

A1. Changement de repère et distance.

Points et vecteurs sont décrits par un triplet de réels. Les matrices sont homogènes et la notation américaine est utilisée.

```
Procédure ChangeRepèrePoint( P : Point; M : Matrice; VAR P' : Point )
Début
    P'x = Px*M[1,1]+Py*M[2,1]+Pz*M[3,1]+M[4,1]
    P'y = Px*M[1,2]+Py*M[2,2]+Pz*M[3,2]+M[4,2]
    P'z = Px*M[1,3]+Py*M[2,3]+Pz*M[3,3]+M[4,3]
    H = Px*M[1,4]+Py*M[2,4]+Pz*M[3,4]+M[4,4]

    P'x = P'x/H
    P'y=P'y/H
    P'z=P'z/H
Fin
```

+/-	*/+	√	Test
12	15	0	0

Procédure ChangeRepèreVecteur(V : Vecteur; M : Matrice; VAR V' : Vecteur)
Début
V'x = Vx*M[1,1]+Vy*M[2,1]+Vz*M[3,1]
V'y = Vx*M[1,2]+Vy*M[2,2]+Vz*M[3,2]
V'z = Vx*M[1,3]+Vy*M[2,3]+Vz*M[3,3]
Fin

+/-	*/÷	√	Test
6	9	0	0

A2. Cube

Procédure IntersectionCubeUnitaire(O : Point; V : Vecteur; VAR l1,l2 : Réel ; B : Bool)
Début
l1=∅; l2=∅; B1=Faux; B2=Faux
Si Vz≠0 **Alors**
{Plan Z=0}
l=-Oz/Vz
Si l≥0 **Alors**
x=Ox+l*Vx
y=Oy+l*Vy
Si (0≤x) **et** (x≤1) **et** (0≤y) et (y≤1) **Alors**
l1 = l; B1=Vrai
Fsi
Fsi

{Plan Z=1}
l=(1-Oz)/Vz
Si l≥0 **Alors**
x=Ox+l*Vx
y=Oy+l*Vy
Si (0≤x) **et** (x≤1) **et** (0≤y) **et** (y≤1) **Alors**
Si B1 **Alors** l2 = l; B2 = Vrai **Sinon** l1 = l; B1 = Vrai **Fsi**
Fsi
Fsi
Fsi

```
Si !B2 et Vy≠0 Alors
        {Plan Y=0}
        l=-Oy/Vy
        Si l≥0 Alors
                x=Ox+l*Vx
                z=Oz+l*Vz
                Si (0≤x) et (x≤1) et (0≤z) et (z≤1) Alors
                        Si B1 Alors l2 = l; B2 = Vrai Sinon l1 = l; B1 = Vrai Fsi
                Fsi
        Fsi

        {Plan Y=1}
        l=(1-Oy)/Vy
        Si l≥0 Alors
                x=Ox+l*Vx
                z=Oz+l*Vz
                Si (0≤x) et (x≤1) et (0≤z) et (z≤1) g
                        Si B1 Alors l2 = l; B2 = Vrai Sinon l1 = l; B1 = Vrai Fsi
                Fsi
        Fsi
Fsi

Si !B2 et Vx≠0 Alors
        {Plan X=0}
        l=-Ox/Vx
        Si l≥0 Alors
                y=Oy+l*Vy
                z=Oz+l*Vz
                Si (0≤y) et (y≤1) et (0≤z) et (z≤1) Alors
                        Si B1 Alors l2 = l; B2 = Vrai Sinon l1 = l; B1 = Vrai Fsi
                Fsi
        Fsi

        {Plan X=1}
        l=(1-Ox)/Vx
        Si l≥0 Alors
                y=Oy+l*Vy
                z=Oz+l*Vz
```

```
                Si (0≤y) et (y≤1) et (0≤z) et (z≤1) Alors
                    l2 = 1
                Fsi
            Fsi
        Fsi

        B = B1 && B2
    Fin
```

+/-	*/+	√	Test
15	18	0	35

```
Procédure IntersectionCube( O : Origine; V : Vecteur; M : Matrice; VAR ListeSegments )
Début
    ChangeRepèrePoint( O,M,O' )
    ChangeRepèreVecteur( V,M,V' )
    IntersectionCubeUnitaire( O',V',l1,l2,B )
    Si B Alors
        P1=(Ox+l1*Vx, Oy+l1*Vy, Oz+l1*Vz)
        P1=(Ox+l2*Vx, Oy+l2*Vy, Oz+l2*Vz)
        Si l1<l2 Alors
            ListeSegments = [P1,P2]
        Sinon
            ListeSegments = [P2,P1]
        Fsi
    Sinon
        ListeSegments = ∅
    Fsi
Fin
```

+/-	*/+	√	Test
39	48	0	37

Procédure AppartientCube(P : Point; M : Matrice) : Booléen
Début
ChangeRepèrePoint(P,M,P')
Si (0≤P'x) **et** (P'x≤1) **et** (0≤P'y) **et** (P'y≤1) **et** (0≤P'z) **et** (P'z≤1) **Alors**
AppartientCube = Vrai
Sinon
AppartientCube = Faux
Fsi
Fin

+/-	*/+	√	Test
12	15	0	6

A3. Sphère.

Procédure IntersectionSphèreUnitaire(O : Point; V : Vecteur; VAR l1,l2 : Réel; B : Bool)
Début
l1=∅; l2=∅; B=Faux
a = Vx*Vx+Vy*Vy+Vz*Vz
b = 2*(Ox*Vx+Oy*Vy+Oz*Vz)
c = Ox*Ox+Oy*Oy+Oz*Oz
Δ = b*b-4*a*c
Si Δ>0 **et** a≠0 **Alors**
rΔ = √Δ
da = 1/(2*a)
l1 = (-b-rΔ)*da
l2 = (-b+rΔ)*da
B = Vrai
Fsi
Fin

+/-	*/+	√	Test
9	17	1	2

Procédure IntersectionSphère(O : Point; V : Vecteur; M : Matrice; VAR ListeSegments)
Début
ChangeRepèrePoint(O,M,O')
ChangeRepèreVecteur(V,M,V')
IntersectionSphèreUnitaire(O',V',l1,l2,B)
Si B **Alors**
P1=(Ox+l1*Vx, Oy+l1*Vy, Oz+l1*Vz)
P1=(Ox+l2*Vx, Oy+l2*Vy, Oz+l2*Vz)
Si l1<l2 **Alors**
ListeSegments = [P1,P2]
Sinon
ListeSegments = [P2,P1]
Fsi
Sinon
ListeSegments = ∅
Fsi
Fin

+/-	*/÷	√	Test
33	47	1	4

Procédure AppartientSphère(P : Point; M : Matrice) : Booléen
Début
ChangeRepèrePoint(P,M,P')
Si P'x*P'x+P'y*P'y+P'z*P'z ≤ 1 **Alors**
AppartientSphère = Vrai
Sinon
AppartientSphère = Faux
Fsi
Fin

+/-	*/÷	√	Test
14	18	0	1

A4. Cylindre.

```
Procédure IntersectionCylindreUnitaire( O : Point; V : Vecteur; VAR l1,l2 : Réel; B : Bool )
Début
    l1=0; l2=0; B1 = Faux; B2 = Faux
    Si Vz≠0 Alors
        {Plan Z=0}
        l=-Oz/Vz
        Si l≥0 Alors

            x=Ox+l*Vx
            y=Oy+l*Vy
            Si (0≤x) et (x≤1) et (0≤y) et (y≤1) Alors
                l1=l; B1=Vrai
            Fsi
        Fsi

        {Plan Z=1}
        l=(1-Oz)/Vz
        Si l≥0 Alors
            x=Ox+l*Vx
            y=Oy+l*Vy
            Si (0≤x) et (x≤1) et (0≤y) et (y≤1) Alors
                Si B1 Alors l2=l; B2=Vrai Sinon l1=l; B1=Vrai Fsi
            Fsi
        Fsi
    Fsi

    Si !B2 Alors
        a = Vx*Vx+Vy*Vy
        b = 2*(Ox*Vx+Oy*Vy)
        c = Ox*Ox+Oy*Oy
        Δ = b*b-4*a*c
        Si Δ>0 et a≠0 Alors
            rΔ = √Δ
            da = 1/(2*a)
            ll1 = (-b-rΔ)*da
            ll2 = (-b+rΔ)*da
```

```
                z1 = Oz+ll1*Vz
                Si (0≤z1) et (z1≤1) Alors
                        Si B1 Alors l2=ll1; B2=Vrai Sinon l1=ll1; B1=Vrai Fsi
                Fsi

                z2 = Oz+ll2*Vz
                Si (0≤z2) et (z2≤1) Alors
                        l2=ll2; B2 = Vrai
                Fsi
            Fsi
        Fsi
Fin
```

+/-	*/+	√	Test
13	20	1	18

```
Procédure IntersectionCylindre(O : Point; V : Vecteur; M : Matrice; VAR ListeSegments)
Début
    ChangeRepèrePoint( O,M,O' )
    ChangeRepèreVecteur( V,M,V' )
    IntersectionCylindreUnitaire( O',V',l1,l2, B )
    Si B Alors
        P1=(Ox+l1*Vx, Oy+l1*Vy, Oz+l1*Vz)
        P1=(Ox+l2*Vx, Oy+l2*Vy, Oz+l2*Vz)
        Si l1<l2 Alors
            ListeSegments = [P1,P2]
        Sinon
            ListeSegments = [P2,P1]
        Fsi
    Sinon
        ListeSegments = ∅
    Fsi
Fin
```

+/-	*/+	√	Test
37	50	1	20

```
Procédure AppartientCylindre( P : Point; M : Matrice ) : Booléen
Début
        ChangeRepèrePoint( P,M,P' )
        Si (0≤P'z) et (P'z≤1) et (P'x*P'x+P'y*P'y ≤ 1) Alors
                AppartientCylindre = Vrai
        Sinon
                AppartientCylindre = Faux
        Fsi
Fin
```

+/-	*/+	√	Test
13	17	0	3

A5. Cône.

```
Procédure IntersectionCôneUnitaire( O : Point; V : Vecteur; VAR p1,p2 : Point )
Début
    l1 = 0; l2 = 0; B1 = Faux; B2 = Faux
    Si Vz≠0 Alors
        l = -Oz/Vz
        Si l≥0 Alors
            x = Ox+l*Vx
            y = Oy+l*Vy
            Si x*x+y*y ≤ 1 ALors
                l1 = l; B1 = Vrai
            Fsi
        Fsi
    Fsi

    a = Vx*Vx+Vy*Vy-Vz*Vz
    b = 2*(Ox*Vx+Oy*Vy-Oz*Vz)
    c = Ox*Ox+Oy*Oy-Oz*Oz
    Δ = b*b-4*a*c
    Si Δ>0 et a≠0 Alors
        rΔ = √Δ
        da = 1/(2*a)
        ll1 = (-b-rΔ)*da
        ll2 = (-b+rΔ)*da
```

```
        z1 = Oz+ll1*Vz
        Si (0≤z1) et (z1≤1) Alors
            Si B1 Alors l2=ll1; B2=Vrai Sinon l1=ll1; B1=Vrai Fsi
        Fsi

        z2 = Oz+ll2*Vz
        Si (0≤z2) et (z2≤1) Alors
            l2=ll2; B2 = Vrai
        Fsi
    Fsi
Fin
```

+/-	*/+	√	Test
14	23	1	9

Procédure IntersectionCône(O : Point; V : Vecteur; M,M^{-1} : Matrice; VAR ListeSegments)
Début

```
    ChangeRepèrePoint( O,M,O' )
    ChangeRepèreVecteur( V,M,V' )
    IntersectionCôneUnitaire( O',V',l1,l2, B )
    Si B Alors
        P1=(Ox+l1*Vx, Oy+l1*Vy, Oz+l1*Vz)
        P1=(Ox+l2*Vx, Oy+l2*Vy, Oz+l2*Vz)
        Si l1<l2 Alors
            ListeSegments = [P1,P2]
        Sinon
            ListeSegments = [P2,P1]
        Fsi
    Sinon
        ListeSegments = ∅
    Fsi
Fin
```

+/-	*/+	√	Test
48	53	1	11

Procédure AppartientCône(P : Point; M : Matrice) : Booléen
Début
ChangeRepèrePoint(P,M,P')
Si (0≤P'z) **et** (P'z≤1) **et** (P'x*P'x+P'y*P'y ≤ P'z*P'z) **Alors**
AppartientCône = Vrai
Sinon
AppartientCône = Faux
Fsi
Fin

+/-	*/+	√	Test
13	18	0	3

A6. Tore.

Calculer l'intersection entre un rayon et un tore revient à résoudre un polynôme de degré 4. Pour ce faire, nous utilisons la méthode de Newton dans laquelle un certain nombre de modifications ont été faites pour profiter au mieux du contexte géométrique. Nous ne décrivons pas en détail la procédure IntersectionTore car elle n'est pas significative des calculs réellement effectués.

Procédure AppartientTore(P : Point; M : Matrice) : Booléen
Début
ChangeRepèrePoint(P,M,P')
XY = Px*Px+Py*Py
D = XY+Pz*Pz+1-r*r
D = D*D-4*XY
Si D≤0 **Alors**
AppartientTore = Vrai
Sinon
AppartientTore = Faux
Fsi
Fin

+/-	*/+	√	Test
17	21	0	1

Annexe B.

Nous présentons, dans cette annexe, les résultats complets obtenus lors des calculs des images mentionnées dans le chapitre 4. Les abbréviations suivantes sont utilisées :

Nb. R : nombre de rayons lancés (primaires, d'ombrage ou secondaires selon les circonstances).

Nb. I : nombre de rayons pour lesquels au moins une intersection a été trouvée.

Nb. O : nombre d'objets étudiés lors du lancé des Nb. R rayons.

Nb. P : nombre des primitives étudiées pour les Nb. R rayons.

Nb. E : nombre d'englobants étudiés pour les Nb. R rayons.

Nb. T : nombre de tests d'appartenance calculés pour les Nb. R rayons.

Tous les temps sont donnés en secondes CPU.

B1. La pièce.

Nombre d'arbres CSG : 1

Nombre d'arbres VSG : 1

Nombre de primitives : 13

Temps de passage du modèle CSG au modèle VSG : 0.01

Temps de calcul de l'arbre BSP : ≈ 0

Temps de calcul de la projection pour les rayons primaires : ≈ 0

Taille de l'image : 500x500

	CSG	VSG
Temps	43	22
Nb. R	154 445	154 445
Nb. I	83 665	83 665
Nb. O	154 445	154 445
Nb. P	417 392	233 596
Nb. E	1 224 103	0
Nb. T	0	880 280

B2. Le barillet.

Nombre d'arbres CSG : 1
Nombre d'arbres VSG : 2
Nombre de primitives : 21
Temps de passage du modèle CSG au modèle VSG : 0.012
Temps de calcul de l'arbre BSP : ≈ 0
Temps de calcul de la projection pour les rayons primaires : ≈ 0
Taille de l'image : 500x500

	CSG	Englobants/VSG	Classement/VSG
Temps	51	21	20
Nb. R	107 604	97 411	97 411
Nb. I	79 411	79 411	79 411
Nb. O	107 604	194 822	127 687
Nb. P	812 123	157 273	167 644
Nb. E	1 378 329	194 822	0
Nb. T	0	1 356 857	1 356 857

B3. L'équerre.

Nombre d'arbres CSG : 1
Nombre d'arbres VSG : 11
Nombre de primitives : 24
Temps de passage du modèle CSG au modèle VSG : 0.012
Temps de calcul de l'arbre BSP : 0.001
Temps de calcul de la projection pour les rayons primaires : 0.003
Taille de l'image : 500x500

	CSG	Englobants/VSG	Classement/VSG
Temps	44	31	20
Nb. R	186 045	106 634	106 634
Nb. I	59 672	59 672	59 672
Nb. O	186 045	1 172 974	188 697
Nb. P	531 156	172 375	229 897
Nb. E	1 678 778	1 172 974	0
Nb. T	0	226 732	210 851

B4. La rectifieuse.

Nombre d'arbres CSG : 1
Nombre d'arbres VSG : 18
Nombre de primitives : 40
Temps de passage du modèle CSG au modèle VSG : 0.013
Temps de calcul de l'arbre BSP : 0.001
Temps de calcul de la projection pour les rayons primaires : 0.004
Taille de l'image : 500x500

	CSG	Englobants/VSG	Classement/VSG
Temps	57	49	29
Nb. R	152 064	110 594	110 594
Nb. I	73 830	73 830	73 830
Nb. O	152 064	1 990 692	296 502
Nb. P	843 910	322 452	278 371
Nb. E	1 763 651	1 990 692	0
Nb. T	0	944 268	648 243

B5. L'étagère.

Nombre d'arbres CSG : 6

Nombre d'arbres VSG : 31

Nombre de primitives : 88

Temps de passage du modèle CSG au modèle VSG : 0.008

Temps de calcul de l'arbre BSP/CSG : 0.001 VSG : 0.04

Temps de calcul de la grille/CSG : 0.003 VSG : 0.02

Temps de calcul de la projection pour les rayons primaires/CSG : 0.001 VSG : 0.001

Temps de calcul de la projection pour les rayons d'ombrage/CSG: 0.001 VSG : 0.007

Temps de calcul de la projection pour les rayons secondaires/CSG : 0.008 VSG : 0.03

Nombre de sources lumineuses : 4

Niveau de profondeur des rayons secondaires : 1

Taille de l'image : 400x250

PRIMAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	43	28	32	18
Nb. RP	100 000	100 000	100 000	100 000
Nb. I	100 000	100 000	100 000	100 000
Nb. O	278 631	182 364	224 908	145 860
Nb. P	438 643	412 288	146 423	157 307
Nb. E	1 270 880	605 480	400 690	0
Nb. T	0	0	339 355	210 924

OMBRAGE

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	76	51	52	30
Nb. RP	400 000	400 000	400 000	400 000
Nb. I	186 802	186 802	186 802	186 802
Nb. O	1 635 800	678 074	1 241 969	477 433
Nb. P	1 397 880	1 357 065	963 112	516 152
Nb. E	2 563 420	1 743 847	0	0
Nb. T	0	0	639 029	498 687

SECONDAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	26	20	20	14
Nb. RP	82 976	82 976	82 976	82 976
Nb. I	33 312	33 312	33 312	33 312
Nb. O	176 488	58 336	221 358	54 586
Nb. P	370 799	196 141	141 693	65 197
Nb. E	886 128	345 063	356 978	7 344
Nb. T	0	0	339 566	211 814

B6. La bride.

Nombre d'arbres CSG : 12
Nombre d'arbres VSG : 39
Nombre de primitives : 72
Temps de passage du modèle CSG au modèle VSG : 0.008
Temps de calcul de l'arbre BSP/CSG : 0.008 VSG : 0.09
Temps de calcul de la grille/CSG : 0.001 VSG : 0.03
Temps de calcul de la projection pour les rayons primaires/CSG : 0.001 VSG : 0.003
Temps de calcul de la projection pour les rayons d'ombrage/CSG: 0.003 VSG : 0.01
Temps de calcul de la projection pour les rayons secondaires/CSG : 0.02 VSG : 0.04
Nombre de sources lumineuses : 4
Niveau de profondeur des rayons secondaires : 1
Taille de l'image : 500x500

PRIMAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	83	45	84	42
Nb. RP	250 000	250 000	250 000	250 000
Nb. I	250 000	250 000	250 000	250 000
Nb. O	576 505	324 298	669 318	299 156
Nb. P	410 860	401 919	295 639	306 808
Nb. E	3 023 755	242 990	1 142 905	0
Nb. T	0	0	160 975	75 986

OMBRAGE

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	178	72	134	91
Nb. RP	1 000 000	1 000 000	1 000 000	1 000 000
Nb. I	218 361	218 361	218 361	218 361
Nb. O	9 730 518	1 337 565	4 167 357	1 125 314
Nb. P	3 326 837	1 586 240	2 473 342	1 164 683
Nb. E	5 189 203	890 347	0	0
Nb. T	0	0	194 506	223 915

SECONDAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	78	49	50	76
Nb. RP	224 537	224 537	224 537	224 537
Nb. I	219 336	219 336	219 336	219 336
Nb. O	620 120	280 957	828 026	266 102
Nb. P	638 451	371 876	476 725	275 760
Nb. E	2 739 733	253 587	1 227 044	0
Nb. T	0	0	155 685	155 685

B7. L'axe.

Nombre d'arbres CSG : 15

Nombre d'arbres VSG : 54

Nombre de primitives : 109

Temps de passage du modèle CSG au modèle VSG : 0.003

Temps de calcul de l'arbre BSP/CSG : 0.01 VSG : 0.2

Temps de calcul de la grille/CSG : 0.003 VSG : 0.04

Temps de calcul de la projection pour les rayons primaires/CSG : 0.002 VSG : 0.006

Temps de calcul de la projection pour les rayons d'ombrage/CSG: 0.006 VSG : 0.03

Temps de calcul de la projection pour les rayons secondaires/CSG : 0.01 VSG : 0.07

Nombre de sources lumineuses : 4

Niveau de profondeur des rayons secondaires : 1

Taille de l'image : 500x500

PRIMAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	114	63	83	45
Nb. RP	250 000	250 000	250 000	250 000
Nb. I	250 000	250 000	250 000	250 000
Nb. O	684 965	500 489	655 610	353 916
Nb. P	1 104 855	1 099 177	310 121	359 339
Nb. E	5 170 699	1 023 133	1 315 507	0
Nb. T	0	0	194 358	114 124

OMBRAGE

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	258	131	137	135
Nb. RP	1 000 000	1 000 000	1 000 000	1 000 000
Nb. I	204 349	204 349	204 349	204 349
Nb. O	12 127 876	2 099 451	4 702 288	1 264 054
Nb. P	5 035 444	4 010 561	2 886 535	1 281 360
Nb. E	12 088 838	3 527 789	0	0
Nb. T	0	0	205 205	235 347

SECONDAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	63	38	45	34
Nb. RP	145 224	145 224	145 224	145 224
Nb. I	128 185	128 185	128 185	128 185
Nb. O	545 682	217 850	524 205	178 650
Nb. P	756 274	546 763	297 720	181 070
Nb. E	2 734 755	555 628	784 620	0
Nb. T	0	0	60 123	44 302

B8. Les engrenages.

Nombre d'arbres CSG : 10
Nombre d'arbres VSG : 81
Nombre de primitives : 273
Temps de passage du modèle CSG au modèle VSG : 0.02
Temps de calcul de l'arbre BSP/CSG : 0.005 VSG : 0.6
Temps de calcul de la grille/CSG : 0.006 VSG : 0.05
Temps de calcul de la projection pour les rayons primaires/CSG : 0.001 VSG : 0.02
Temps de calcul de la projection pour les rayons d'ombrage/CSG: 0.001 VSG : 0.09
Temps de calcul de la projection pour les rayons secondaires/CSG : 0.005 VSG : 0.16
Nombre de sources lumineuses : 4
Niveau de profondeur des rayons secondaires : 1
Taille de l'image : 500x500

PRIMAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	105	54	91	46
Nb. RP	250 000	250 000	250 000	250 000
Nb. I	250 000	250 000	250 000	250 000
Nb. O	532 824	350 386	784 818	379 865
Nb. P	662 170	619 065	489 901	483 133
Nb. E	4 637 653	1 238 670	1 388 559	0
Nb. T	0	0	431 990	265 771

OMBRAGE

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	207	125	150	325
Nb. RP	1 000 000	1 000 000	1 000 000	1 000 000
Nb. I	135 315	135 315	135 315	135 315
Nb. O	9 744 799	1 607 353	5 271 628	1 614 179
Nb. P	3 741 053	2 422 909	3 858 419	2 017 738
Nb. E	12 217 502	5 637 657	0	0
Nb. T	0	0	731 216	721 306

SECONDAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	78	47	73	86
Nb. RP	213 557	213 557	213 557	213 557
Nb. I	200 888	200 888	200 888	200 888
Nb. O	596 747	243 192	886 084	255 120
Nb. P	578 871	363 045	502 175	297 409
Nb. E	3 027 305	539 042	1 151 073	0
Nb. T	0	0	449 346	105 088

B9. La voiture.

Nombre d'arbres CSG : 39

Nombre d'arbres VSG : 289

Nombre de primitives : 385

Temps de passage du modèle CSG au modèle VSG : 0.06

Temps de calcul de l'arbre BSP/CSG : 0.05 VSG : 3.8

Temps de calcul de la grille/CSG : 0.36 VSG : 0.1

Temps de calcul de la projection pour les rayons primaires/CSG : 0.003 VSG : 0.03

Temps de calcul de la projection pour les rayons d'ombrage/CSG: 0.01 VSG : 0.15

Temps de calcul de la projection pour les rayons secondaires/CSG : 0.01 VSG : 0.43

Nombre de sources lumineuses : 4

Niveau de profondeur des rayons secondaires : 1

Taille de l'image : 500x500

PRIMAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	183	88	161	105
Nb. RP	250 000	250 000	250 000	250 000
Nb. I	248 097	248 097	248 097	248 097
Nb. O	910 904	401 338	975 690	398 809
Nb. P	1 763 903	1 134 174	541 077	428 075
Nb. E	9 457 208	1 723 538	9 159 787	0
Nb. T	0	0	497 363	173 242

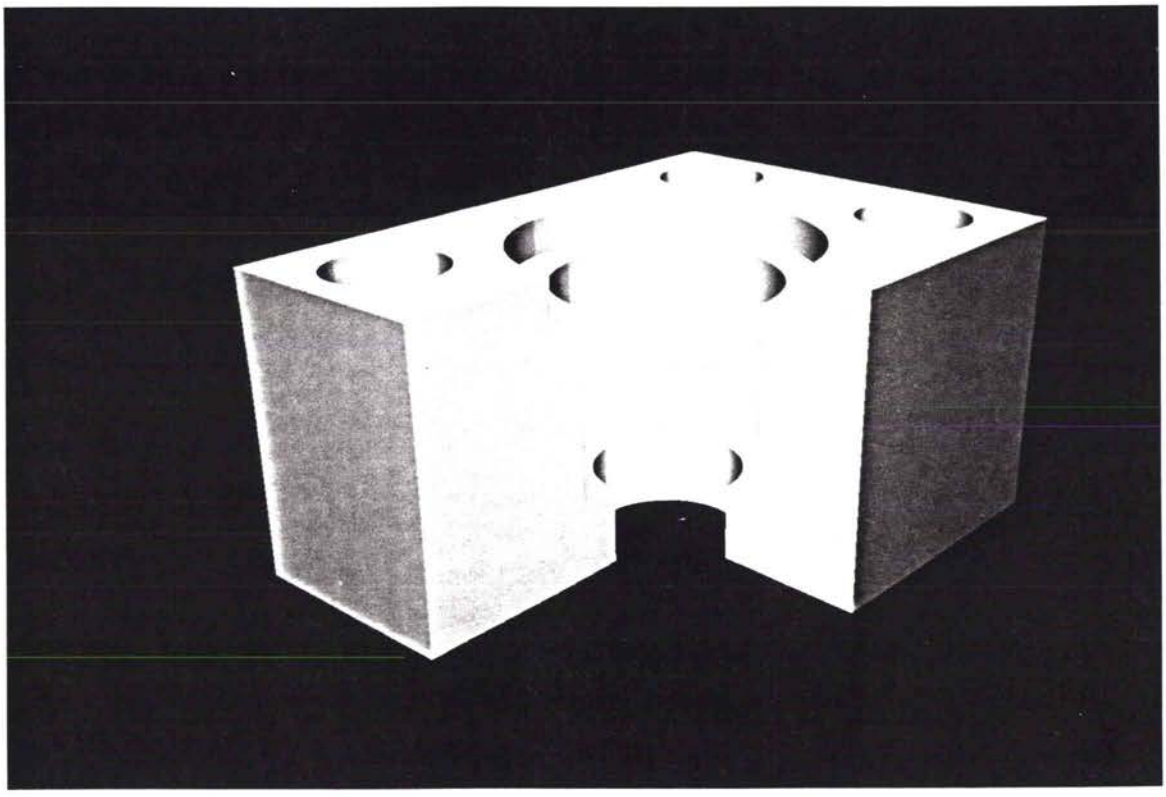
OMBRAGE

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	392	216	365	466
Nb. RP	992 388	992 388	992 388	992 388
Nb. I	152 945	152 945	152 945	152 945
Nb. O	24 094 148	1 971 540	21 780 147	1 310 503
Nb. P	8 321 711	3 149 141	19 468 702	1 404 103
Nb. E	20 203 801	4 599 336	0	0
Nb. T	0	0	447 861	450 412

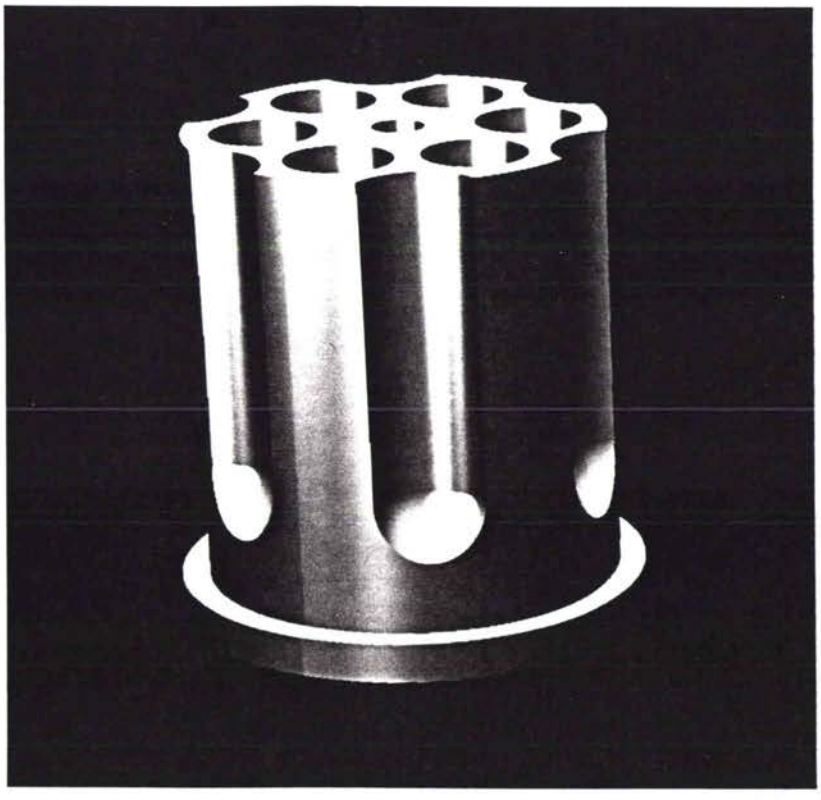
SECONDAIRES

	Grille/CSG	Classement/CSG	Grille/VSG	Classement/VSG
Temps	89	43	60	88
Nb. RP	170 458	170 458	170 458	170 458
Nb. I	33 079	33 079	33 079	33 079
Nb. O	375 332	115 998	531 898	92 105
Nb. P	1 006 747	538 497	339 032	111 523
Nb. E	6 539 260	968 065	4 203 884	0
Nb. T	0	0	398 247	224 072

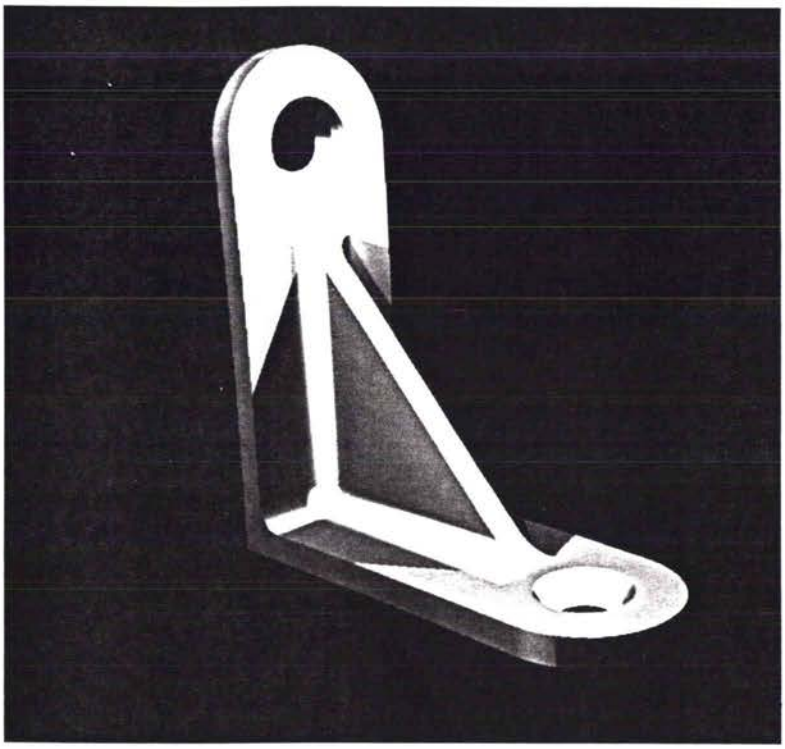
Annexe C.



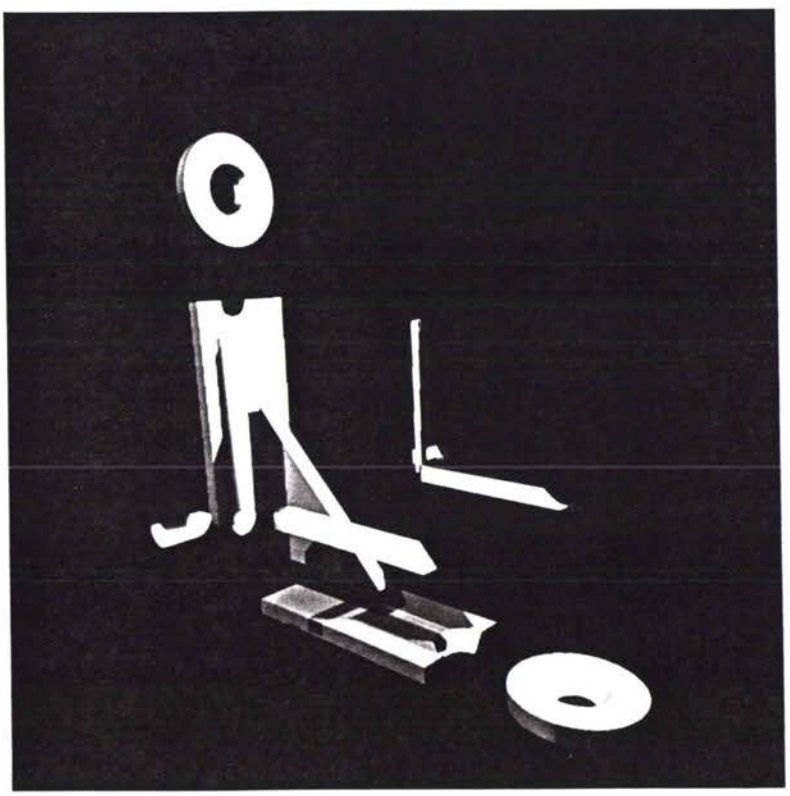
La pièce



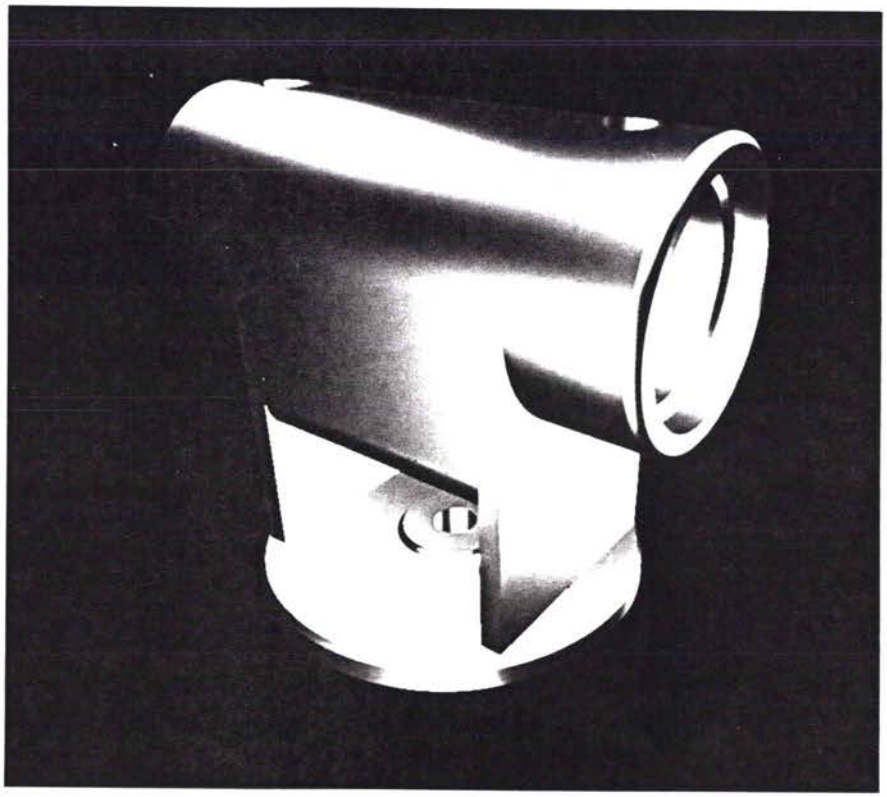
Le barillet



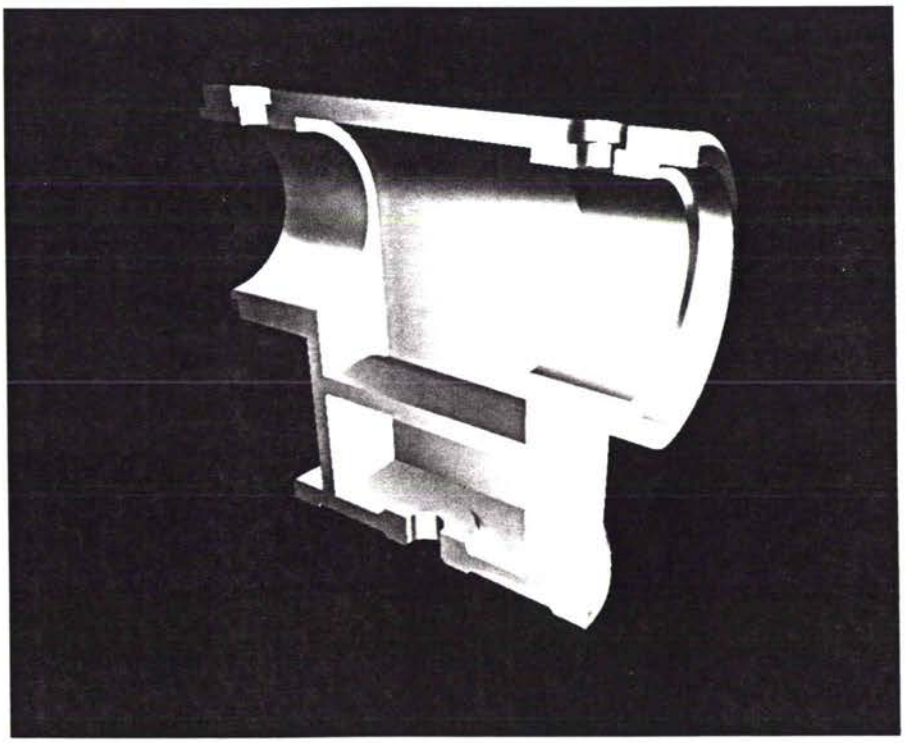
L'équerre



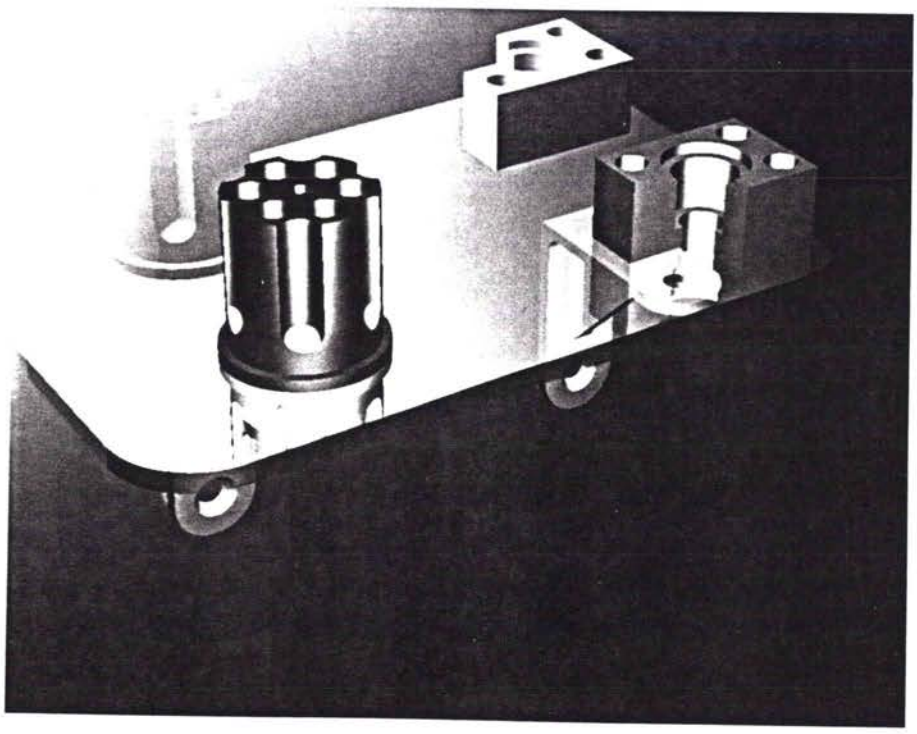
Vue éclatée de l'équerre.
Chaque partie correspond à un arbre VSG



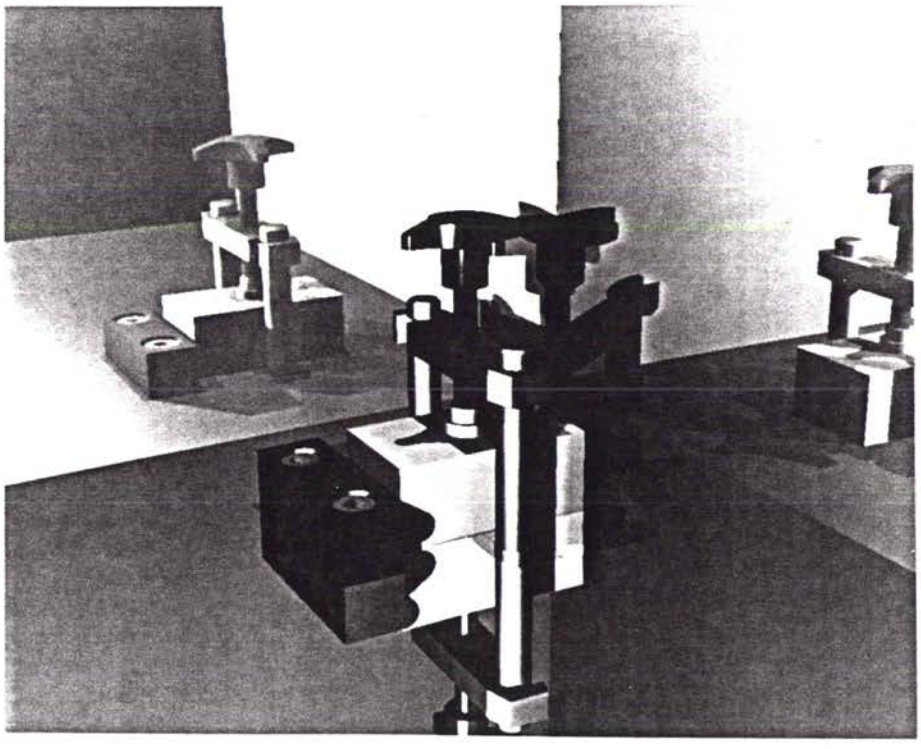
La rectifieuse



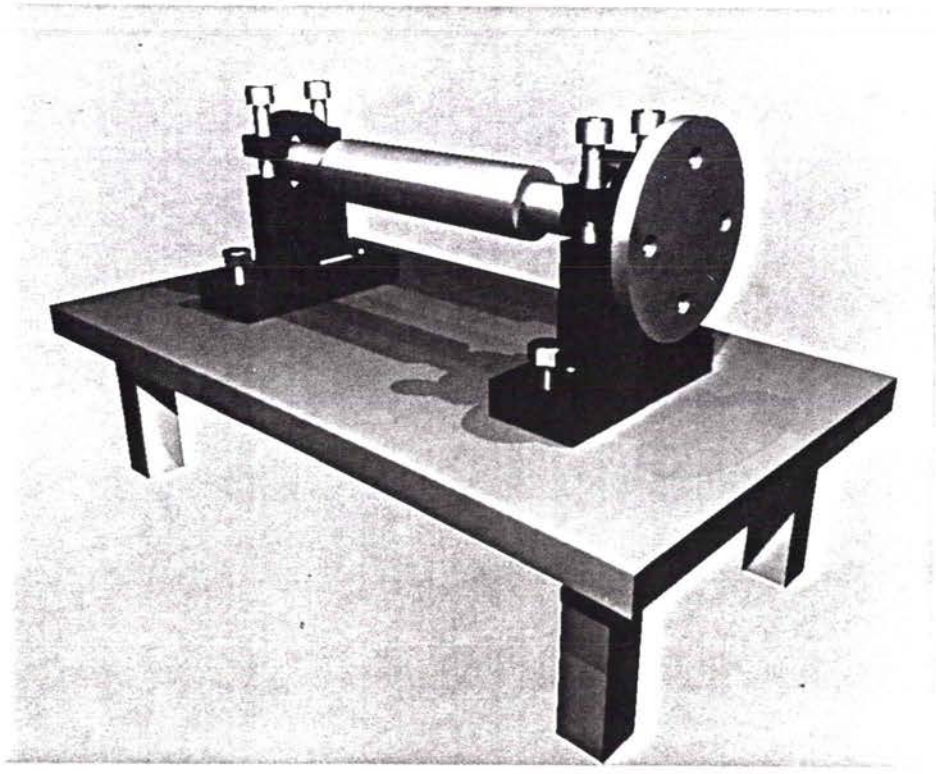
Vue en coupe de la rectifieuse



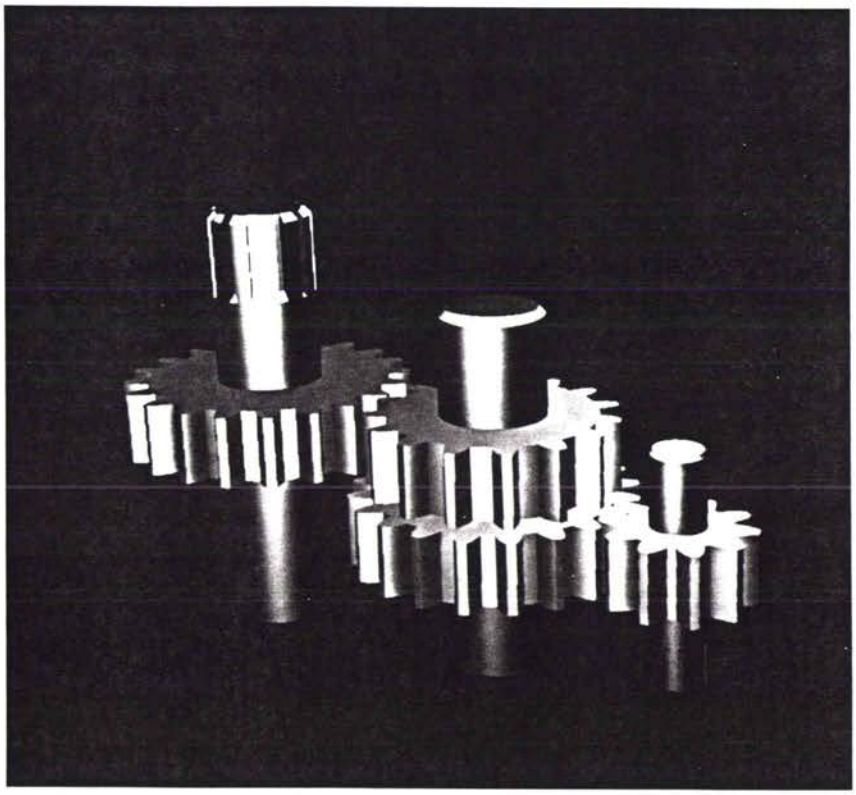
L'étagère



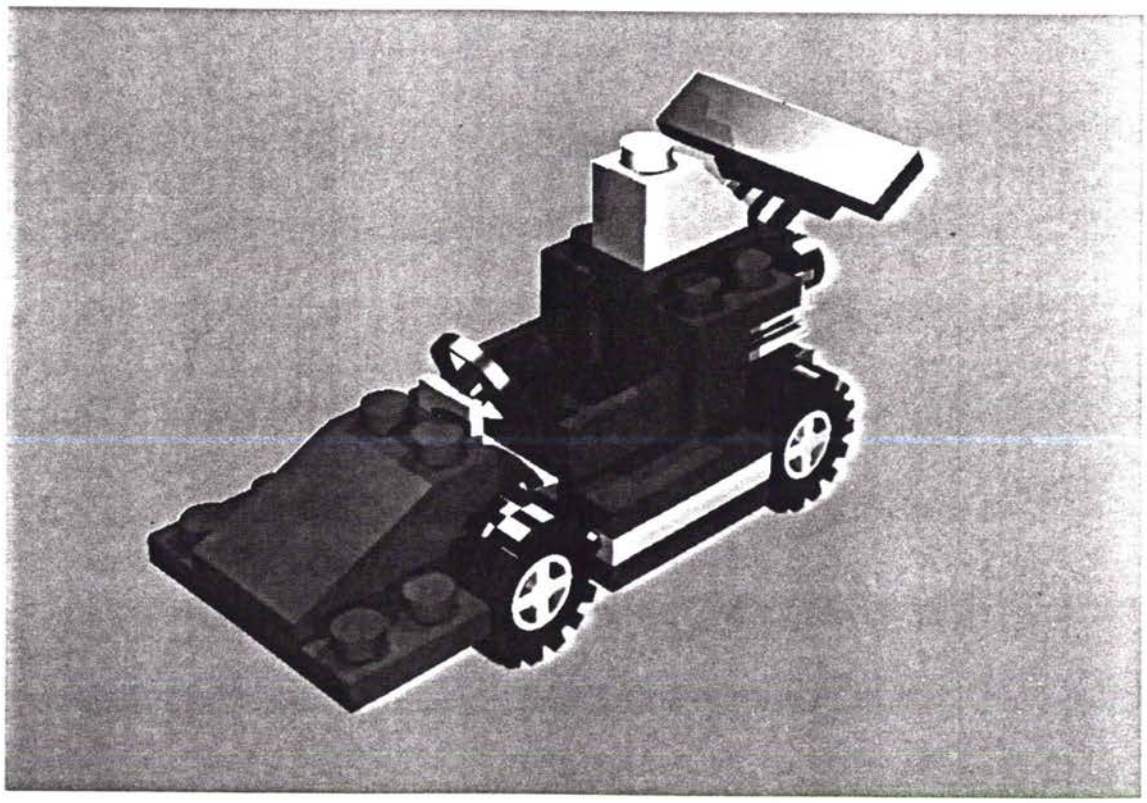
La bride



L'axe



Les engrenages



La voiture

Références bibliographiques

Ouvrages

[FOL 91] Foley, DJ, van Dam, A, Feiner, K S and Hughes J F "Computer Graphics : Principles and Practice", Seconde édition, Addison-Wesley Publishing Compagny.

[GLA 89] Glassner, A S "An Introduction to Ray Tracing", Academic Press, 1989.

[GAR 88] Gardan, Y "Matériels et logiciels de base", Hermes, 1988.

[GAR 91] Gardan, Y "La CFAO. Introduction, techniques et mise en oeuvre", 3ème édition, Hermes, 1991.

[PER 88] Peroche, B, Argence, J, Ghazanfarpour, D et Michelucci, D "La synthèse d'images", Editions Hermes, 1988.

[ROG 85] Rodgers, D F "Procedural Elements for Computer Graphics", McGraw-Hill, New York, 1985.

Bibliographie

[AMA 92] Amanatides, J and Woo, A "A Fast Voxel Traversal Algorithm for Ray Tracing", Proc. Eurographics 87, Elsevier Science Publ., 1987, pp. 1-10.

[APP 68] Appel, A "Some Techniques for Shading Machine Renderings of Solids", AFIPS 1968, Spring Joint Computer Conf., pp. 37-45.

[ATH 83] Atherton, P R "A Scan-Line Hidden Surface Removal Procedure For Constructive Solid Geometry", Computer Graphics(Proc. SIGGRAPH 83), Vol. 17, No. 3, July 1983, pp. 73.82.

[BEI 88] Beigbeder, "Un développement pour la modélisation et la visualisation en synthèse d'images : CASTOR", thèse, Ecole des mines de Saint Etienne, 1988.

[BLI 76] Blinn, J F and Newell, M E "Texture and Reflection in Computer Genarated Images", Communications of the ACM, Vol. 19, No. 10, 1976, pp. 542-547.

[BOU 87] Bouatouch, K, Madani, M O, Priol, T and Arnaldi, B "A New Algorithm of Space Tracing Using a CSG Model", Proc. Eurographics 87, Elsevier Science Publ., New York, 1987, pp. 65-77.

[BRO 84] Bronsvoort, W F, Van Wijk, JJ and Jansen, F W "Two Methods for Improving the Efficiency of Ray Casting in Solid Modelling", Computer-Aided Design, Vol. 16, No. 1, Jan. 1984, pp. 51-55.

[BRO 86] Bronsvoort, W F "Techniques for Reducing Boolean Evaluation Time in CSG Scan-line Algorithms", Computer-Aided Design, Vol. 18, No. 10, December 1986, pp. 533-538.

[BRO 87] Bronsvoort, W F "An Algorithm for Visible-Line and Visible-Surface Display of CSG Models", Visual Comput., Vol. 3, No. 4, December 1987, pp. 176-185.

[BRO 88] Bronsvoort, W F "Boundary Evaluation and Direct Display of CSG Models", Computer-Aided Design, Vol. 20, No. 7, September 1988, pp. 416-419.

[BRO 89] Bronsvoort, W F and Garnaat, H "Incremental Display of CSG Models Using Local Updating", Proc. Eurographics 89, Vol. 21, No. 4, May 1989, pp. 221-231.

[CAF 90] Campbell, A T and Fussel, D S "Adaptative Mesh Generation For Global Diffuse Illumination", Computer Graphics(Proc SIGGRAPH 90), Vol. 24, No. 4, August 1990, pp. 155-164.

[CAM 89] Cameron, S "Efficient Intersection Tests for Objets Defined Constructively", Int. J. Robotics Automation, Vol. 6, No. 3, June 1989, pp. 291-302.

[CAM 90] Cameron, S and Rossignac, J R "Relationship Between S-Bounds and Active Zones in Constructive Solid Geometry", In Theory and Practise of Geometric Mödelling, Springer-Verlag, Blaubeuren, Germany, 1988, pp. 369-382.

[CAM 92] Cameron, S and Yap, C K "Refinement Methods For Geometric Bounds in Constructive Solid Geometry", ACM Transaction on Graphics, Vol. 11, No. 1, January 1992, pp. 12-39.

[CHI 89] Chin, N and Feiner, S "Near Real-Time Shadow Generation Using BSP Trees", Computer Graphics(Proc SIGGRAPH 89), Vol. 23, No. 3, July 1989, pp. 99-106.

[COO 82] Cook, R L, Torrance, K E "A Reflectance Model for Computer Graphics", ACM Transaction on Graphics, Vol. 1, No. 1, January 1982, pp. 7-24.

[COO 84] Cook, R, Porter, T and Carpenter, L "Distributed Ray Tracing", Computer Graphics(Proc SIGGRAPH 84), Vol. 18, No. 3, July 1984, pp. 109-115.

[CRO 82] Crow, F C "A More Flexible Image Generation Environment", Computer Graphics, Vol. 16, No. 3, July 1982, pp. 9-18.

[DEV 88] Devillers, O "Méthodes d'optimisation du tracé de rayons", Thèse de l'université de Paris-Sud, Juin 88.

[EXC 88] Excoffier, T, Tosan, E "Une méthode d'optimisation du lancer de rayons", MICAD 87, pp. 549-563.

[FUC 80] Fuchs, H, Kedem, A and Naylor, B "On Visible Surface Generation By A Priori Tree Structures", Computer Graphics(Proc SIGGRAPH 80), Vol. 14, No. 3, July 1980, pp. 124-133.

[FUC 83] Fuchs, H, Abram, G D and Grant, A D "Near Real-Time Shaded Display of Rigid Objects", Computer Graphics(Proc SIGGRAPH 83), Vol. 17, No. 3, July 1983, pp. 65-72.

[FUJ 86] Fujimoto, A, Tanaka, T and Iwata, K "ARTS : Accelerated Ray Tracing System", Computer Graphics and Applications, Vol. 6, No. 4, April 1986, pp. 16-26.

[GAR 82] Gardan, Y "Eléments méthodologiques pour la réalisation de systèmes de CFAO et leur introduction dans les entreprises", thèse de doctorat d'état, INPG, décembre 1982.

[GAR 90] Gardan, Y, Paul, J.C. et Vivian, R "Amélioration du calcul des images de synthèse par une analyse de la scène", GROS PLAN 1990, pp. 1-13.

[GAR 91a] Gardan, Y et Zakari, A "L'intégration dans les systèmes CFAO : à travers le modèle ou à travers les outils ?", Conventions IA 1991, pp. 455-471.

[GAR 91b] Gardan, Y, Lanuel, Y et Vivian, R "Une décomposition adaptative pour le tracé de rayons: méthode et résultats", GROS PLAN 1991, pp. 47-55.

[GLA 84] Glassner, A S "Space Subdivision for Fast Ray Tracing", IEEE Computer Graphics and Applications, Oct. 1984, pp. 15-22.

[GOL 89] Goldfeather, J, Molnar,S, Turk, G, Fuchs, H "Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning", IEEE Computer Graphics and Applications, May 1989, pp. 20-28.

[GOR 91] Gordon, D and Chen, S "Front-to-Back Display of BSP Trees", IEEE Comput. Graph. Appl., pp. 79-85.

[HAI 86] Haines, E A, Greenberg, D P "The Light-Buffer : a Shadows Testing Accelerator", IEEE Computer Graphics and Applications, Vol. 6, No. 9, September 1986, pp. 6-16.

[HEC 84] Heckbert, P S, Hanrahan, P "Beam Tracing Polygonal Objects", Computer Graphics, Vol. 18, No. 3, July 1984, pp. 119-127.

[JAH 91] Jahami, G "Pour un système de synthèse d'images flexible et évolutif", Thèse de l'université de Saint-Etienne et de l'école nationale supérieure des mines de Saint-Etienne, Mars 1991.

[JAN 85] Jansen, F W "A CSG List Priority Hidden Surface Algorithm", Proc. Eurographics 85, Amsterdam, pp. 51-62.

[JAN 86] Jansen, F W "A Pixel-Parallel Hidden-Surface Algorithms For Constructive Solid Geometry", Proc. Eurographics 86, Amsterdam, pp. 51-62.

[JAN 91] Jansen, F W "Depth-Order Point Classification Techniques for CSG Display Algorithms", ACM Transactions on Graphics, Vol. 10, No. 1, January 1991, pp. 40-70.

[KAP 85] Kaplan, M R "SpaceTracing, a Constant Time Ray Tracer", In SIGGRAPH' 85 tutorial on the uses of spatial coherence in ray tracing, 1985.

[LAN 93] Lanuel, Y, et Gardan, Y "Propositions pour améliorer les performances des algorithmes de visualisation d'un arbre CSG", Rapport de recherche, Laboratoire de Recherche en Informatique de Metz, Août 93.

[LAN 94] Lanuel, Y et Gardan, Y "Les arbres VSG : une nouvelle approche multimodèles de la visualisation", MICAD 94.

[LEE 82] Lee, F W and Requicha, A A G "Algorithms For Computing the Volume and Other Integral Properties of Solids", Communications of the ACM, Vol. 25, No. 9, 1982, pp. 635-650.

[LEE 87] Lee, Y C and Fu, K S "Machine Understanding of CSG : Extraction and Unification of Manufacturing Features", IEEE Comput. Graph. Appl., Vol. 7, No. 1,1987, pp. 20-32.

[LUC 90] Lucas, M "Techniques de parallèlisation en synthèse d'images", Proc. MICAD 1990, pp. 1-27.

[MEA 81] Meagher, D "Geometric Modeling Using Octree Encoding", Computer Graphics and Image Processing, Vol. 19, 1982, pp. 129-147.

[NEW 72] Newell, M E, Newell, R G, Sancha, T L "A New Approach to the Shaded Picture Problem", Proc. ACM National Conference, 1972, pp. 443-450.

[NIE 89] Nie, Z et Peroche, B "Deformations libres et arbres de construction", BIGRE No. 67, Journées AFCET GROPLAN, Décembre 1989, pp. 98-109.

[PUE 87] Pueyo, X and Mendoza, J C "A New Scan Line Algorithm for the Rendering of CSG Trees", Proc. Eurographics 87, Amsterdam, pp. 347-361.

[PER 90] Perng, D B, Chen, Z and Li, R K "Automatic 3D Machining Feature Extraction From 3D CSG Solid Input", Computer Aided Design, Vol. 22, No. 5, June 1990, pp. 285-295.

[PER 94] Perrin, E et Gardan, Y "Un nouvel algorithme pour les opérations booléennes sur des solides par les frontières", MICAD 94.

[REQ 80] Requicha, A A G "Representations for Rigid Soldis: Theory, methods and systems", ACM Computing Surveys, Vol. 12, No. 4, Dec. 1980, pp.437-464.

[REQ 92] Requicha, A A G and Rossignac J R "Solid Modeling and Beyond", IEEE Comput. Graph. Appl., September 1992, pp. 31-44.

[ROS 86] Rossignac, J R and Requicha, A A G "Depth Buffering Display Techniques for Constructive Solid Geometry", IEEE Comput. Graph. Appl., Vol. 6, No. 9, September 1986, pp. 29-39.

[ROS 89] Rossignac, J R and Voelker, H "Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection, and Shading Algorithms", ACM Transaction on Graphics, Vol. 8, No. 1, January 1989, pp. 51-87.

[ROT 82] Roth, S D "Ray Casting for Modeling Solids", Computer Graphics and Image Processing, Vol. 18, No. 2, February 1982, pp. 109-144.

[SAK 90] Sakurai, H and Gossard, D C "Recognizing Shape Features in Solid Models", IEEE Comput. Graph. Appl., September 1990, pp. 22-32.

[SAM 85] Samet, H, Tamminen, M "Bintrees, CSG Trees, and Time", Computer Graphics (Proc. SIGGRAPH 85), Vol. 19, No. 3, July 1985, pp. 121-130.

[SAN 90] Sandouk, M Z, Caubet, R, Natallah, N et Djedi, N "Lancer de rayons optimisé pour des objets CSG : méthode d'accélération par un prétraitement de l'arbre CSG", MICAD 1990, pp. 720-735.

[SAT 85] Sato, H, Ishii, M, Sato, K, Ikesaka, M, Ishihata, H, Kakimoto, M, Hirota, K, Inoue, K "Fast Image Generation of Constructive Solid Geometry Using a Cellular Array Processor", Computer Graphics (Proc. SIGGRAPH 85), Vol. 19, No. 3, July 1985, pp. 95-102.

[SCH 69] Schumacker, R A, Brand, B, Gilliland, M and Sharp, W "Study For applying Computer-Generated Images to Visual Simulation", AFHRL-TR-69-14, U.S. Air Force Human Ressources Laboratory, Sept. 1969.

[SIL 89] Sillon, F "Simulation de l'éclairage pour la synthèse d'images : réalisme et interactivité", thèse de l'université de Paris-sud, Juin 1989.

[SIL 91] Sillon, F, Arvo, J R, Westin, S H and Greenberg, D P "A Global Illumination Solution for General Reflectance Distribution", Computer Graphics (Proc. SIGGRAPH 91), Vol. 25, No. 4, July 1991, pp. 187-196.

[SUT 74] Sutherland, I E, Sproull, R F and Schumacker, R A "A Characterization of Ten Hidden-Surface Algorithms", Computing Surveys, Vol. 6, No. 1, March 1974.

[TAM 84] Tamminen, M, Karonen, O, Mäntylä, M "Ray-Casting and Block Model Conversion Using a Spatial Index", Computer Aided Design, Vol. 16, No. 4, July 1984, pp. 203-208.

[THI 87] Thibault, W and Naylor, B "Set Operations on Polyhedra Using Binary Space Partionning Trees", Computer Graphics (Proc. SIGGRAPH 87), Vol. 21, No. 3, July 1987, pp. 153-162.

[TIL 80] Tilove, R B "Set Membership Classification : A Unified Approach to Geometric Intersection Problems", IEEE Trans. Computers, Vol. 29, No. 10, Oct. 1980, pp. 874-883.

[TIL 81] Tilove, R B "Exploiting Spatial Structural Locality in Geometric Modelling", Tech. Memo., No. 38, Production Automation Project, Univ. of Rochester, Oct. 1981.

[VER 87] Verroust, A "Visualization Algorithm for CSG Polyhedral Solids", Comput.-Aided Des., Vol. 19, No. 10, December 1987, pp. 527-533.

[VIV 93a] Vivian, R, "Définition d'une approche adaptative : application à la visualisation en CAO", Thèse de l'université de Metz, Janvier 1993.

[VIV 93b] Vivian, R, Gardan, Y et Lanuel, Y "Optimisation du tracé de rayons, une approche originale et adaptative située entre le ray-tracing et le beam-tracing", MICAD 1993.

[WAT 70] Watkins, G S "A Real-Time Visible Surface Algorithm", Computer Science Department, University of Utah, UTECH-CSC-70-101, June 1970.

[WHI 80] Whitted, T "An Improved Illumination Model For Shaded Display", Communication of the ACM, Vol. 23, June 1980, pp. 343-349.

[WIL 78] Williams, L "Casting Curved Shadows on Curved Surfaces", Computer Graphics (Proc. SIGGRAPH 78), pp. 270-274.

[WYV 86] Wyvill, G, Kunii, T L and Shirai, Y "Space Division for Ray Tracing in CSG", IEEE Comput. Graph. Appl., April 1986, pp. 28-34.

[WOO 90] Woo, A, Poulin, P and Fournier, A "A survey of Shadow Algorithms", IEEE Comput. Graph. Appl., Vol. 10, No. 6, Nov. 1990, pp. 13-32.

[WOO 93] Woo, A "Efficient Shadow Computations in Ray Tracing", IEEE Comput. Graph. Appl., September 1993, pp. 78-83.

Résumé.

La visualisation occupe une place importante dans les systèmes de CFAO (Conception et Fabrication Assistée par Ordinateur). C'est souvent le meilleur moyen d'appréhender les résultats des traitements réalisés par la machine. Le temps de calcul des images réalistes contenant réflexions, transparences, ombres portées, etc. sont encore aujourd'hui très important, ce qui interdit leur utilisation dans les phases interactives. Le problème abordé dans ce mémoire est de réduire ces temps de calcul en proposant, dans le cadre d'une approche multimodèles des systèmes de CFAO, un nouveau modèle de visualisation, appelé VSG (Visual Solid Geometry), dont les propriétés constituent à elles seules une amélioration. Nous cherchons en particulier à utiliser au mieux ce modèle avec l'algorithme du lancer de rayons qui permet d'obtenir le réalisme souhaité.