3. LA SYNTHESE D'IMAGES REALISTES

3.1 Les lignes et les surfaces cachées

3.1.1 Classification des algorithmes

Si nous considérons le dessin de la Figure 3.1-1a, nous sommes tout-à-fait incapables de décider si ce dessin représente deux cubes, comme ceux de la Figure 3.1-1b ou deux cubes comme ceux de la Figure 3.1-1c. En effet, le premier dessin est ambigü, tout simplement parce que les lignes qui devraient être cachées ne le sont pas.

Le problème des lignes cachées dans les objets a été étudié très tôt (1967) et on trouvera une description des principales techniques dans l'article de Sutherland et al. (1974). Mais on peut distinguer trois catégories d'algorithmes:

1. les algorithmes se déroulant dans l'espace de l'usager (3D). Dans ces algorithmes, tous les calculs et les comparaisons sont effectués entre les éléments géométriques définis par l'usager dans son espace. Ces algorithmes sont appelés algorithmes de lignes cachées, car ils produisent une liste de lignes visibles comme résultat. Le principal défaut de ces algorithmes est leur lenteur d'exécution; ils ont par contre l'avantage d'être exacts et de s'adapter particulièrement à la sortie sur traceur digital. Parmi de tels algorithmes, signalons celui de Appel (1967) et celui de Galimberti et Montanari (1969).

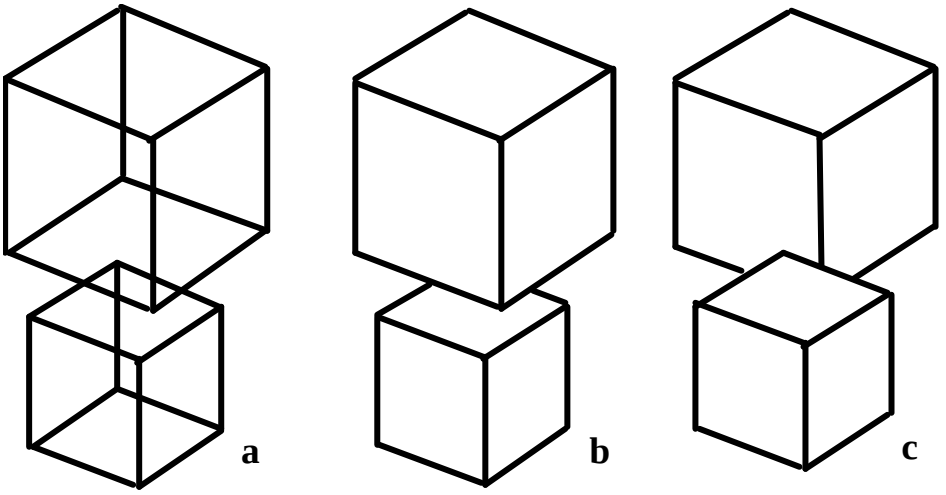


Figure 3.1-1. Le rôle des lignes cachées

2. les algorithmes se déroulant dans l'espace du dispositif graphique ou algorithmes de surfaces cachées. Ces algorithmes sont basés sur la technologie des terminaux. En particulier, ils conviennent aux terminaux de type "raster". En effet, ils sont généralement basés sur le principe que les objets sont composés de faces polygonales et on doit décider quelle face est visible à chaque pixel de l'écran; ce qui implique de garder l'information de profondeur. Les plus connus de ces algorithmes sont les algorithmes de Warnock (1969), de Watkins (1970) et de la

mémoire de profondeur (Catmull, 1975). Ces algorithmes sont encore actuellement les plus utilisés et nous les décrirons brièvement.

3. les algorithmes mixtes. Ces algorithmes forment un compromis entre les deux autres types d'algorithmes. Ils comportent deux phases principales:

    1° construction dans l'espace 3D d'une liste des priorités des objets selon leur profondeur,

    2° détermination de la visibilité des objets dans l'espace du dispositif graphique. Le principal algorithme de ce type est la méthode des priorités de Encarnacao (1970).

3.1.2 Les tests

3.1.2.1 Test de visibilité

Dans tous les algorithmes de lignes ou de surfaces cachées, le test de visibilité (voir Figure 3.1-2) joue un rôle fondamental. Il permet de déterminer quelles faces sont potentiellement visibles; ce qui permet d'éliminer très tôt les faces qui ne peuvent être vues par l'observateur, ce qui accélère considérablement les algorithmes.

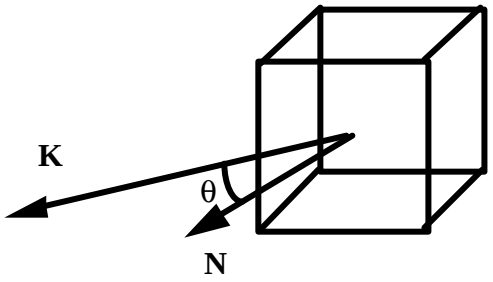


Figure 3.1-2. Test de visibilité

$$\theta = \arccos \frac{N \cdot K,}{\| N \| \| K \|}$$

où **N** est le vecteur normal à la face et **K** un vecteur en direction de l'observateur (oeil de la caméra synthétique).

Si $\theta = \pi/2$ la face est potentiellement visible et si $\theta > \pi/2$ la face est invisible. Le test est évidemment insuffisant pour déterminer la visibilité de faces relativement à d'autres.

3.1.2.2 Test de profondeur

Une face F est comparée avec un point-test $\mathbf{P_t}$ pour détecter si la face cache le point. Pour faire ce test, on fait passer une droite L par le point $\mathbf{P_t}$ et l'oeil **E** de la caméra virtuelle. On peut maintenant considérer le point d'intersection $\mathbf{P_f}$ entre la droite L et la face F. Comme le montre la Figure 3.1-3, $\mathbf{P_t}$ est caché si: distance ($\mathbf{E}$,$\mathbf{P_t}$) > distance ($\mathbf{E}$,$\mathbf{P_f}$)

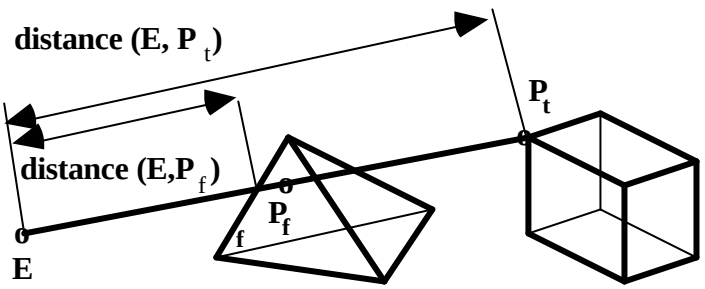


Figure 3.1-3. Test de profondeur

3.1.2.3 Test d'appartenance

1. Test sur les angles

On désigne par ? $_i$ l'angle entre les segments de droite $\mathbf{PS_i}$ et $\mathbf{PS_{i+1}}$, comme le montre la Figure 3.1-4. **P** est à l'intérieur de F si $\sum_i a_i = 2p$ et **P** est à l'extérieur de F si $\sum_i a_i = 0$.

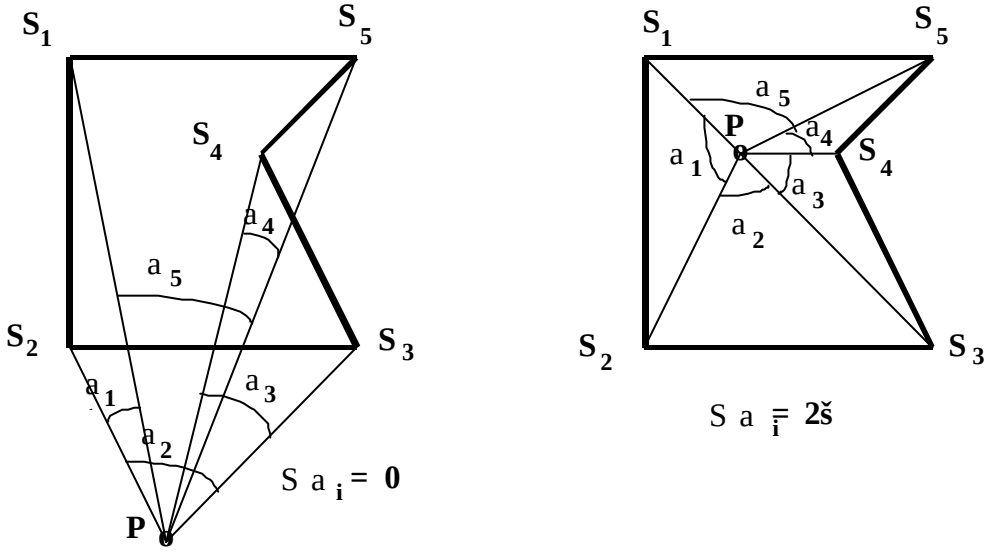


Figure 3.1-4. Test sur les angles

2. Test sur le nombre de points d'intersection

Considérons une demi-droite partant du point **P** et coupant au moins une arête de F sans passer par un de ses sommets (voir Figure 3.1-5). Si le nombre de points d'intersection de la demi-droite avec les arêtes de F est impair alors **P** est à l'intérieur de F; sinon il est à l'extérieur.

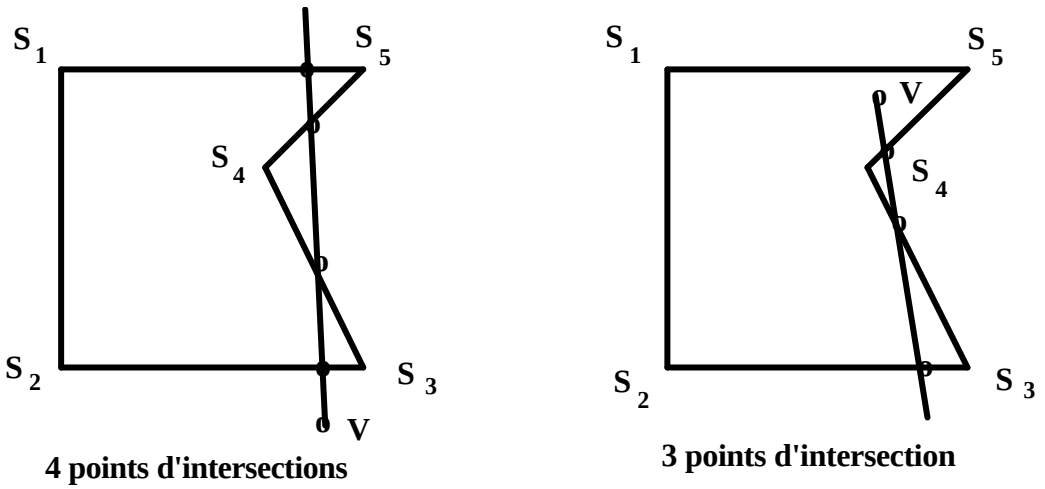


Figure 3.1-5. Test sur les points d'intersection

3.1.2.4 Test minmax

Le test minmax est un test simple permettant de détecter rapidement que deux polygones ne se recouvrent pas. Comme le montre la Figure 3.1-6, les polygones F et G ne se recouvrent pas si:

$$F_{bx} > G_{hx} \text{ ou } F_{hx} < G_{bx} \text{ ou } F_{by} > G_{hy} \text{ ou } F_{hy} < G_{by}$$

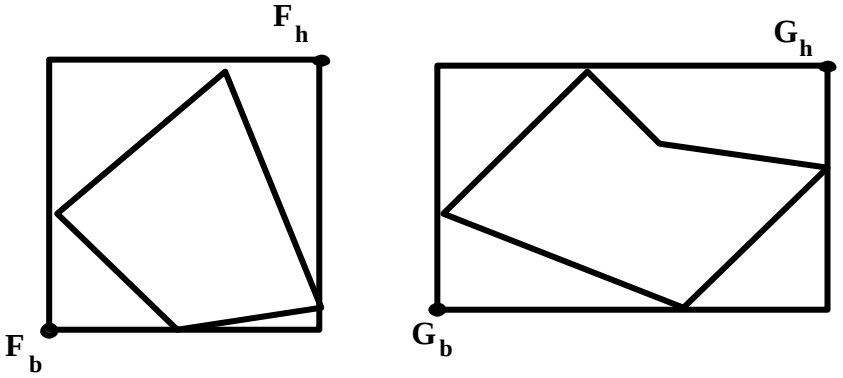


Figure 3.1-6. Test minmax

Le test minmax n'est pas concluant dans le cas de la Figure 3.1-7.

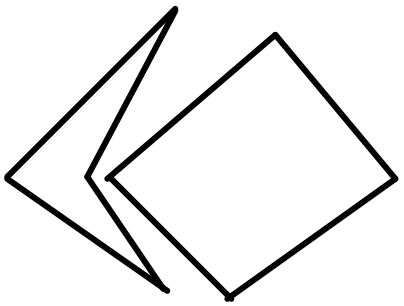


Figure 3.1-7. Test minmax non concluant

3.1.2.5 Algorithme de subdivision de Warnock

Le principe général de l'algorithme est extrèmement simple: l'écran est divisé en fenêtres rectangulaires. Trois cas sont alors considérés pour chaque fenêtre:

1. Il n'y a rien à voir dans la fenêtre, donc pas de problème.

2. Ce qui est visible dans la fenêtre est facile à dessiner, donc il n'y a pas de problème.

3. Ce qui est visible est trop difficile à dessiner, donc divisons la fenêtre en 4 fenêtres plus petites et recommençons récursivement.

Il faut noter que le processus récursif s'arrête dans trois cas:

a) Il n'y a rien à voir, donc la fenêtre est coloriée avec la couleur de fond

b) la fenêtre se reduit à un pixel

c) la fenêtre est facile à colorier, car un seul polygone la recouvre ou a une intersection avec

L'algorithme est typiquement récursif, se terminant avec l'une des 3 conditions suivantes:

1. Il n'y a rien à voir—la fenêtre est colorée avec la couleur de fond

2. La fenêtre est réduite à un pixel—comme aucune subdivision n'est possible, elle doit être colorée avec la couleur appropriée. La fenêtre de la taille d'un pixel est alors examinée pour voir si elle est entourée par un ou plusieurs polygones de la scene. Deux cas sont possibles:

    I) Il y a des polygones entourant la fenêtre: ceux-ci sont alors testés (au centre du pixel) pour voir lequel est le plus proche de l'observateur à ce pixel. Le pixel est affiché avec la couleur du polygone le plus proche.

    ii) Il n'y a pas de polygones entourant la fenêtre: Le pixel est affiché avec la couleur de fond.

3. La fenêtre est facile à colorer; ceci est possible quand:

    i) Un seul polygone entoure la fenêtre et il n'y a pas d'autres polygones dans la fenêtre. Dans ce cas, la fenêtre est colorée avec la couleur appropriée (Figure 3.1-8a)

    ii) Un seul polygone intersecte la fenêtre. Dans ce cas, la fenêtre est premièrement colorée avec la couleur de fond puis la portion du polygone intersectant la fenêtre est colorée avec la couleur appropriée (Figure 3.1-8b)

    iii) Le polygone le plus proche de l'oeil entoure la fenêtre. Dans ce cas, la fenêtre est colorée avec la couleur appropriée pour le polygone entourant (Figure 3.1-8c)

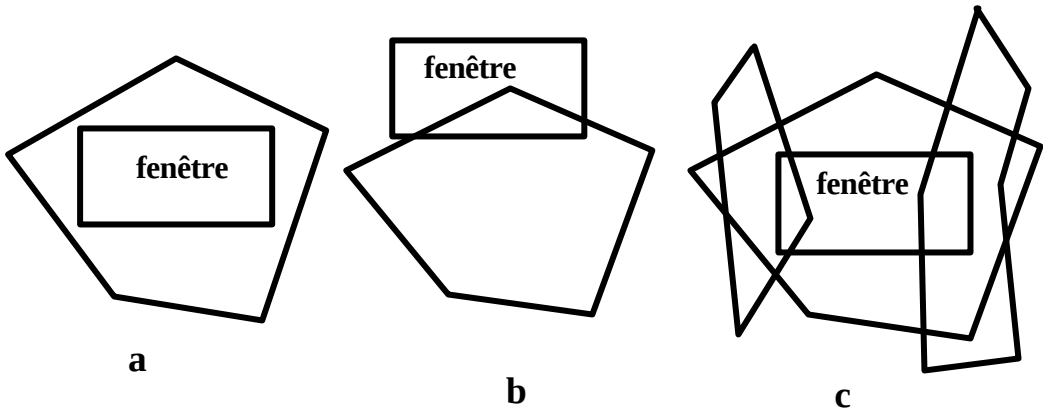


Figure 3.1-8. Fenêtres faciles à colorer

Pour vérifier qu'un polygone n'intersecte pas la fenêtre, on peut commencer par opérer un test minmax (voir Section 3.1.2.4). Si le test n'est pas concluant, alors on vérifie si tous les sommets de la fenêtre sont toujours dans le même demi-plan formé par une arête (voir Figure 3.1-8. Fenêtres faciles à colorer). S'il n'y pas d'intersection, alors soit la fenêtre est à l'intérieur du polygone soit à l'extérieur; il suffit alors de tester l'appartenance d'un point de la fenêtre (p.e. le centre) au polygone selon une des méthodes de la Section 3.1.2.3.

L'algorithme peut être optimisé en utilisant les rectangles englobant les polygones comme subdivisions, comme montré à la Figure 3.1-9.

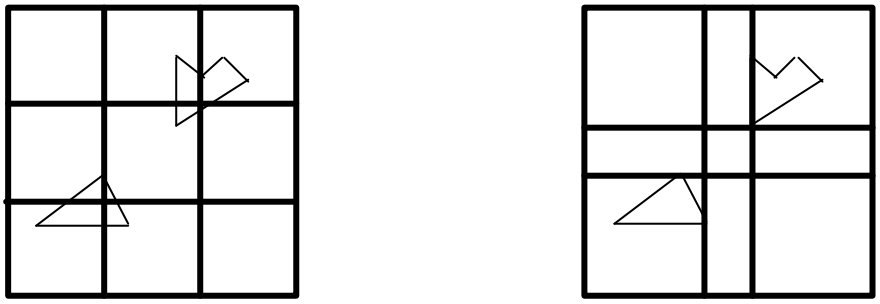


Figure 3.1-9. Subdivisions de fenêtres dans l'algorithme de Warnock. **a** non-optimisé; **b** optimisé

3.1.3 Algorithme de balayage de Watkins

Cet algorithme est basé sur le même principe que l'algorithme de remplissage de polygones vu à la Section 1.4.1.2. Avant de voir cet algorithme, nous allons étudier comment améliorer l'algorithme 2D.

3.1.3.1 Amélioration de l'algorithme de remplissage

Une première amélioration consiste à utiliser un calcul incrémentiel; si la ligne de balayage i a une intersection avec une arête en X_i, il y a une grande probabilité que la ligne de balayage X_{i+1} en ait aussi une qui peut se calculer par:

$$X_{i+1} = X_i + 1/m$$

où m est la pente de l'arête.

Une autre amélioration consiste à introduire une table des arêtes actives (TAA). Une arête est active si elle a eu une intersection avec la dernière ligne de balayage. Dans la TAA, les arêtes sont triées selon les coordonnées des extrémités. Une mise-à-jour de la TAA est faite après chaque balayage à partir de la table des arêtes (TA) qui contient pour chaque arête du polygone:

1. la plus petite coordonnée y (pour éliminer l'arête de la TAA)

2. la coordonnée x de l'extrémité de l'arête ayant la plus grande coordonnée y (pour introduire l'arête dans la TAA).

3. la valeur de 1/m

3.1.3.2 L'algorithme de surfaces cachées

Pour l'algorithme de surfaces cachées, on conserve deux tableaux à une dimension de longueur égale à la résolution en x du terminal. Le premier tableau contiendra les couleurs des pixels de la ligne courante et le second tableau contiendra la profondeur courante en chaque pixel de la ligne. Le principe de l'algorithme est alors le suivant et s'applique à chaque ligne de balayage: on commence par initialiser le tableau des profondeurs à une valeur maximale et le tableau des couleurs à la couleur de fond; ensuite pour chaque projection de polygone de la scène, on cherche les pixels de la ligne de balayage qui sont dans le polygone. Ensuite pour chaque pixel, on calcule la profondeur Z, si elle est inférieure à la profondeur courante (stockée dans le tableau), on remplace la valeur dans le tableau par Z et on modifie la valeur de couleur correspondante. A la fin du traitement de la ligne de balayage, le tableau des couleurs contient exactement la ligne à afficher. L'algorithme peut être considérablement amélioré en tenant compte de la cohérence des arêtes et en effectuant des tris judicieux selon x et y. Ainsi, on peut introduire une table des polygones actifs semblable à la TAA. On ajoute également dans la TA la coordonnée z_i initiale (à gauche normalement) pour une arête et la variation relative des z par rapport à une variation de 1 selon les x. On a ainsi un calcul incrémentiel:

$$z = z_i + n \cdot \delta z_{inc}$$

où n est le nombre d'unités selon l'axe X.

3.1.3.3 Décomposition en zones

Romney, Bouknight et Watkins ont proposé de tenir compte de la cohérence spatiale en décomposant en zones (spans). Considérons par exemple la Figure 3.1-10 qui nous montre un plan de balayage XZ.

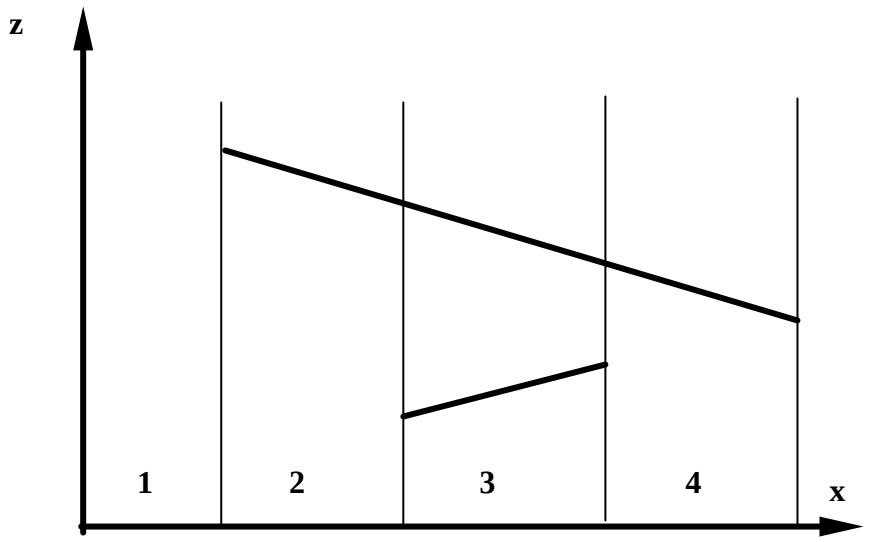


Figure 3.1-10. Décomposition en zones

On a, dans cet exemple, les zones suivantes:

1. zone vide (donc de la couleur de fond)

2. zone contenant un segment (couleur du polygone)

3. zone contenant deux segments (à déterminer par comparaison)

4. zone contenant un segment (couleur du polygone)

L'algorithme général devient alors:

```
Pour chaque arête
    trier selon les y
Pour chaque ligne de balayage
    Pour chaque polygone actif
            Déterminer le segment obtenu par l'intersection du plan
            de balayage et du polygone
    Trier les segments selon X
    Déterminer les extrémités des zones

    Pour chaque zone
            Couper les segments actifs aux extrémités des zones
            Déterminer la visibilité des segments dans la zone
            en comparant les 2
            Afficher le segment visible
```

Il faut remarquer que lorsque des segments se coupent, on doit délimiter une nouvelle zone à l'intersection.

3.1.3.4 Algorithme de la mémoire de profondeur (z-buffer)

Si on suppose qu'on est capable de mémoriser les profondeurs à chaque pixel pour toute l'image, on peut appliquer un algorithme semblable à celui de Watkins, mais beaucoup plus efficace, car on n'a

plus besoin d'ordonner les polygones. En effet, dans un algorithme de balayage, à chaque ligne, il faut trouver tous les polygones concernés; dans l'algorithme de la mémoire de profondeur, on peut traiter les polygones dans un ordre quelconque. Si l'algorithme est simple, il nécessite néanmoins une mémoire considérable et rapide. Un z-buffer typique, comme celui que l'on trouve dans les stations Silicon Graphics, a une instruction qui écrit un pixel seulement si la valeur z est plus petite que la valeur courante. En pseudo-code l'algorithme peut se décrire ainsi:

```
pour chaque pixel <x,y>
        intensité [x,y] := couleur de fond
        zbuffer [x,y] := valeur maximum de profondeur

pour chaque polygone dans la scène
        trouver tous les pixels <x,y> dans le polygone projeté
        pour chaque pixel
                calculer la profondeur z en <x,y>
                si z < zbuffer [x,y]
                alors
                        zbuffer [x,y] := z
                        intensité [x,y] := valeur à calculer selon la lumière
```

Il faut noter que pour calculer la profondeur z en <x,y>, on doit expliciter la coordonnée z de l'équation du plan de la surface:

$$z = \frac{-Ax - By - D}{C}$$

3.1.4 La transparence

3.1.4.1 La transparence simple

Généralement, lorsqu'un objet se trouve devant un autre, il le masque. En informatique graphique, on simule cette propriété en utilisant un algorithme d'élimination des surfaces cachées. Pourtant, la lumière peut traverser certains matériaux et dans ce cas, ces matériaux sont dits transparents et on ne doit plus cacher les objets situés derrière. Il faut alors représenter différemment ces objets vus à travers un corps transparent. Une technique évoluée comme le lancer de rayons (voir Section 3.4) tiendrait compte exactement des lois physiques et plus particulièrement des lois de la réfraction (voir Section 3.4.1). Cependant, tenir compte de ces lois est très coûteux et si on peut se permettre moins de réalisme, on peut se rabattre sur des techniques moins complexes. La plus simple peut s'expliquer par un exemple: supposons qu'on regarde une boite rouge à travers une vitre verte. La couleur de la boite sera un mélange de rouge et de vert. La proportion de rouge et de vert est dépendante du coefficient de transmission (transparence) de la vitre. Si la vitre était complètement opaque, on n'aurait que du vert; si la vitre était complètement transparente, la boite serait vraiment rouge; entre ces deux extrêmes, tous les mélanges sont possibles. On peut donc appliquer la formule suivante:

$$I = t*I_1 + (1-t)*I_2$$

où I_1 est l'intensité du point P_1 situé derrière le point P_2 dont l'intensité est I_2 ; t est le facteur de transparence du polygone auquel appartient P_2, sa valeur varie entre 0 (opaque) et 1 (complètement transparent).

3.1.4.2 La transparence pour des surfaces courbes

Cette technique donne des résultats acceptables avec des objets non courbes et ultra-minces. Pour un objet comme une sphère, il faut tenir compte de la courbure, en effet, la transparence est plus grande en direction du centre qu'aux bords, car le chemin parcouru par la lumière dans la sphère est plus important dans les bords, comme le montre la Figure 3.1-11.

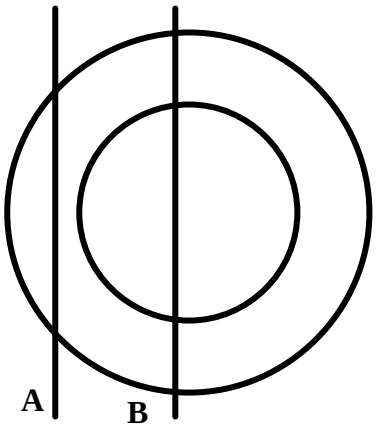


Figure 3.1-11. La distance parcourue dans l'objet transparent par le rayon A est plus grande que celle parcourue par B

Une solution pour résoudre ce problème est de calculer un facteur de transparence dépendant de la courbure de l'objet. Par exemple, en supposant un observateur placé sur l'axe z:

$$t = t_{min} + (t_{max} - t_{min})(1 - (1 - n_z))$$

où: t est le facteur de transparence pour une position donnée

t_{min} est la facteur de transparence minimal

t_{max} est le facteur de transparence maximal

n_z est la valeur absolue de la composante z du vecteur unitaire normal à la surface pour une position donnée

Ainsi quand la courbure augmente, n_z décroît et t tend vers t_{min}, tandis que quand la courbure décroît, n_z augmente et t tend vers t_{max}. Pour tenir compte de l'épaisseur du matériau, on peut introduire un exposant m dans l'équation:

$$t = t_{min} + (t_{max} - t_{min})(1 - (1 - n_z)^m)$$

Un verre mince est alors simulé par une grande valeur de m et un verre épais par une petite valeur de m.

3.1.4.3 Le A-buffer

Pour implanter les surfaces transparentes, on peut modifier un algorithme de balayage en utilisant la technique de buffer d'accumulation ou A-buffer (Carpenter, 1984). Les polygones transparents sont tout d'abord calculés dans le système de coordonnées de l'oeil, comme pour les autres polygones.

Le A-buffer est un tableau d'éléments de dimension de la ligne de balayage composés d'au moins deux champs pour chaque pixel: la profondeur du premier polygone opaque et un pointeur sur la liste des polygones transparents à ce pixel avec la valeur de transparence pour chacun. La liste des polygones transparents est évidemment triée selon la profondeur. La Figure 3.1-12 nous montre un exemple de A-buffer.

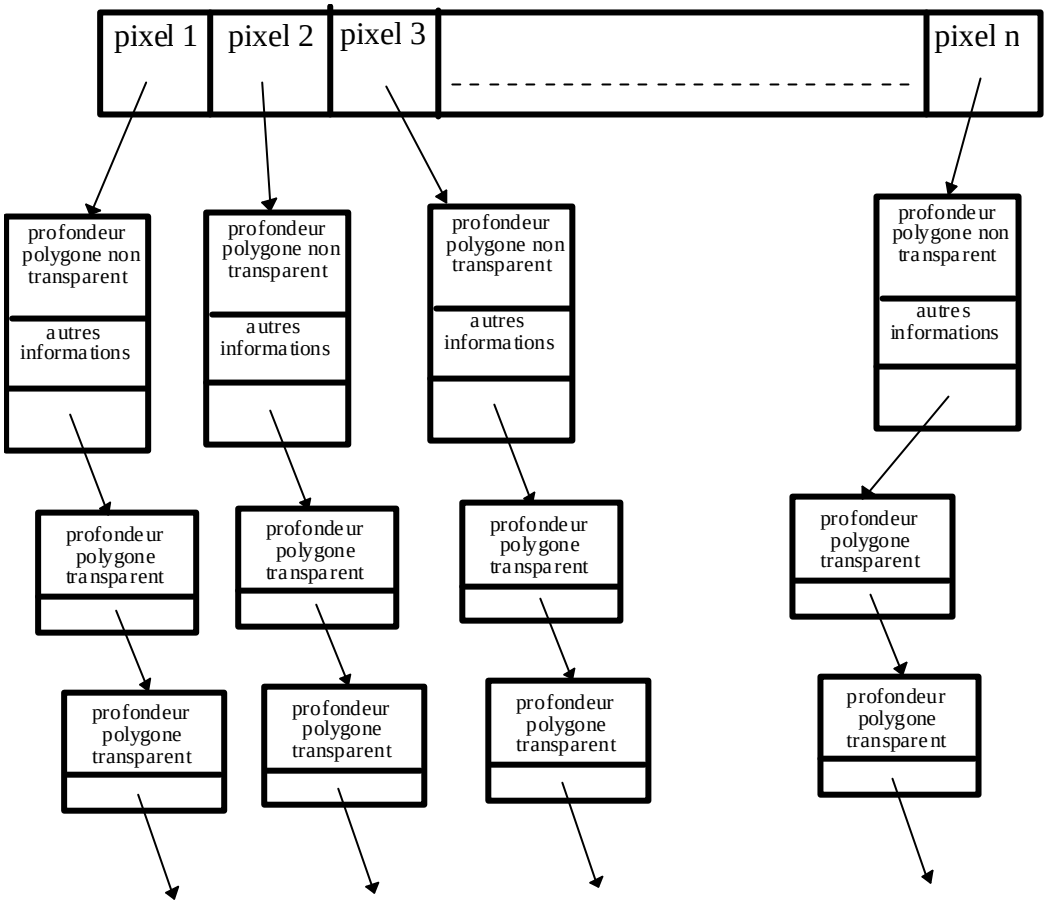


Figure 3.1-12. Exemple de A-buffer

3.2 La lumière de synthèse

3.2.1 Introduction

Dans les différents algorithmes de surfaces cachées, on doit colorier un pixel d'une couleur; mais laquelle ? La réponse n'est pas simple, en effet, si on veut dessiner une boule verte et qu'on colorie tous les pixels de la boule avec une même couleur verte, on aura un cercle vert qui n'aura absolument pas l'allure d'une sphère. Ce qui donne l'impression de volume est la répartition des intensités de couleurs sur la surface. Chaque point peut avoir une intensité différente et cette intensité est fonction de la lumière présente. Donc pour synthétiser une scène tridimensionnelle par ordinateur avec de la couleur, il faut aussi synthétiser de la lumière. On distingue généralement plusieurs types de sources de lumière:

1. la **lumière ambiante**, c'est une lumière qui éclaire toute la scène uniformément et qui est uniquement caractérisée par son intensité.

2. les **sources directionnelles**, ce sont des sources de lumière, supposées à l'infini et qui éclairent la scène avec des rayons parallèles à une direction donnée; elles sont donc caractérisées par leur intensité et leur direction.

3. les **sources ponctuelles**, ce sont des sources de lumière, supposées placées en un point précis et qui rayonnent la lumière radialement; elles sont donc caractérisées par leur intensité et leur position.

4. les **projecteurs** ou **spots**, ce sont des sources de lumière caractérisées par leur position, leur direction et un facteur de concentration.

3.2.2 L'illumination locale

3.2.2.1 Le modèle de Phong

Phong (1975) a introduit un modèle très utilisé pour le calcul de la lumière en un point donné. Dans ce modèle, l'intensité en un point P est donnée par:

$$I = I_a + I_d + I_s$$

où I_a est l'intensité due à la lumière ambiante, I_d est l'intensité de la lumière due à la diffusion sur la surface et I_s est l'intensité de la lumière réfléchie vers l'observateur formant un reflet (highlight) sur l'objet; cette lumière est appelée communément lumière spéculaire. Notons qu'un objet très mat a une forte composante de diffusion, tandis qu'un miroir a une forte composante I_s.

La composante ambiante I_a ne provient d'aucune source, c'est une illumination globale provenant des parois, des autres objets; on peut dire en fait que la lumière ambiante représente une source de lumière distribuée uniformément.

La diffusion se calcule facilement, car elle ne fait pas intervenir l'observateur (voir Figure 3.2-1), on peut donc utiliser la loi de Lambert qui calcule I_d comme la somme sur toutes les sources de lumière du produit scalaire NL, où N est le vecteur normal à la surface au point considéré et L est un vecteur dirigé vers la source de lumière.

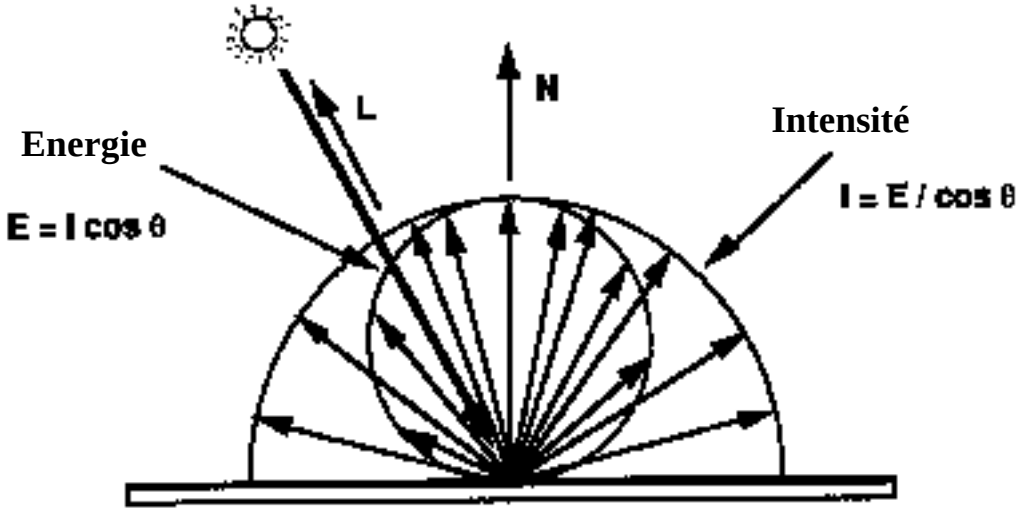


Figure 3.2-1.Réflexion diffuse

On a donc:

$$I_d = I_i \, k_d \, N \cdot L$$

où I_i est l'intensité de la i-ième source de lumière, k_d est la constante de réflexion diffuse qui varie d'un matériau à l'autre et dépend de la longueur d'onde de la lumière; sa valeur est située entre 0 et 1.

La composante I_s est beaucoup plus difficile à calculer. Le modèle le plus simple est de considérer un miroir parfait. Dans ce cas (voir Figure 3.2-2), un rayon incident est simplement réfléchi en un simple rayon formant avec la normale le même angle que le rayon incident.

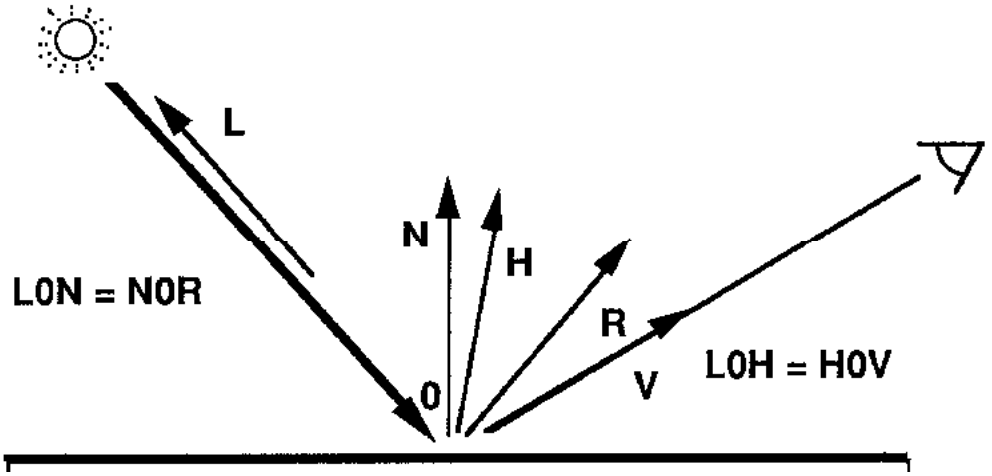


Figure 3.2-2. Réflexion spéculaire simple

Hélas, la situation est rarement aussi simple et il nous faut un modèle plus complexe de réflexion (voir Figure 3.2-3 et Figure 3.2-4).

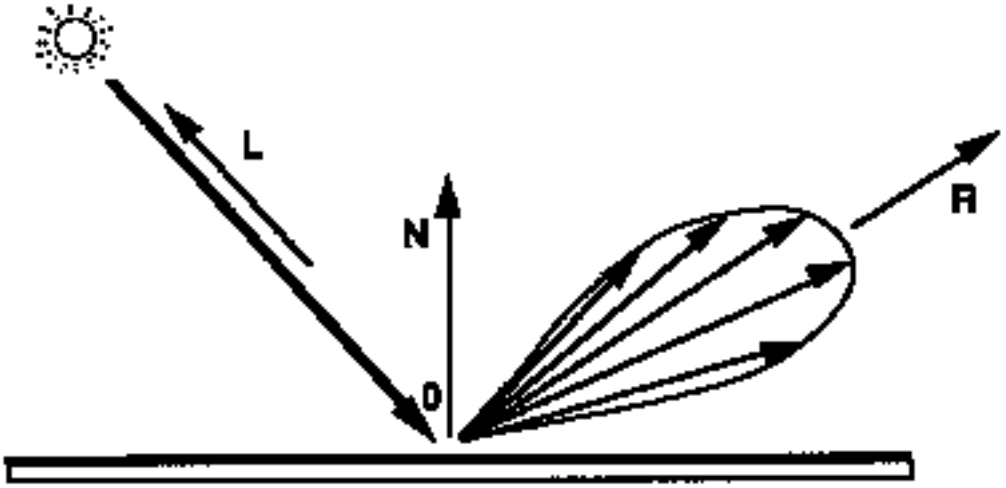


Figure 3.2-3. Réflexion spéculaire complexe

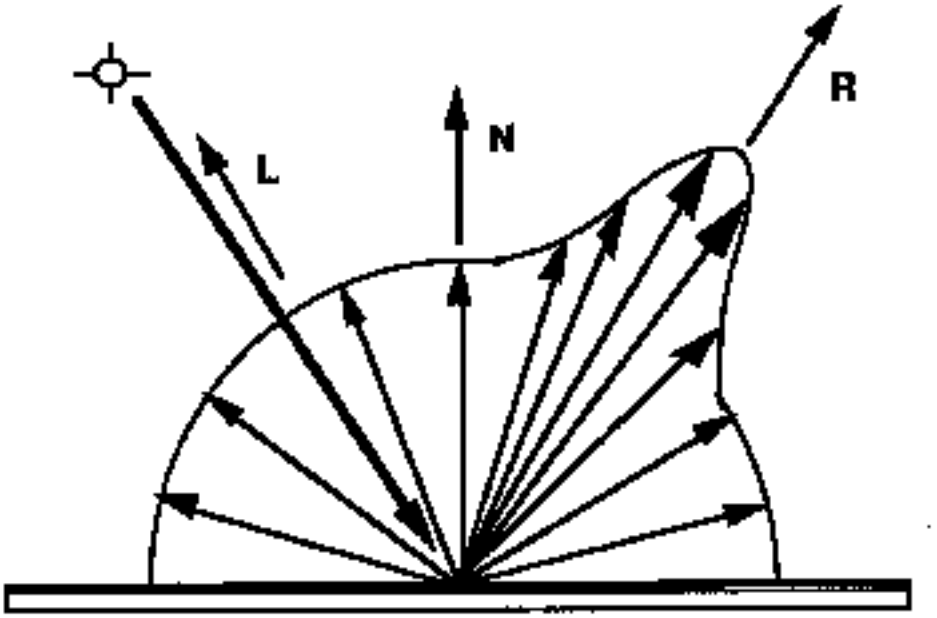


Figure 3.2-4. Réflexion diffuse et spéculaire

C'est là que Phong a introduit une expression qui est proportionnelle à la somme pour chaque source du produit scalaire des vecteurs R et O, où R est la direction du rayon réfléchi et O la direction de l'observateur et n est un exposant qui contrôle le reflet. On a donc:

$$I_s = I_i \, k_s \, (R \cdot O)^n$$

avec I_i l'intensité de la i-ième source de lumière et k_s la constante de réflexion spéculaire.

L'exposant n dépend de la surface; typiquement, il varie entre 1 et 200 et est infini pour un parfait réflecteur. Il faut noter que le terme I_s est empirique et contient un produit scalaire à la puissance n, ce qui signifie un $\cos^n a$ où a est l'angle entre la direction formée par les vecteurs L' et N. De grandes valeurs de n correspondent à des surfaces métalliques tandis que de petites valeurs de n

correspondent à des surfaces non-métalliques (du papier par exemple). Plus l'exposant n est grand, plus le cosinus diminue et le reflet est moins étalé.

James Blinn a montré qu'on pouvait optimiser la méthode en remplaçant le produit scalaire R·O par le produit N·L' où L' est un vecteur dirigé vers le point milieu entre l'observateur et la source de lumière et N est le vecteur normal à la surface au point considéré:

$$I_s = I_i \, k_s \, (N \cdot L')^n$$

Essayons maintenant d'expliquer cette formule empirique.

1. du point de vue mathématique:

$$L' = \frac{L + O}{2}$$

du point de vue géométrique (Figure 3.2-5):

L+O est la diagonale du parallélogramme complété selon L et O. L' est la moitié de la diagonale.

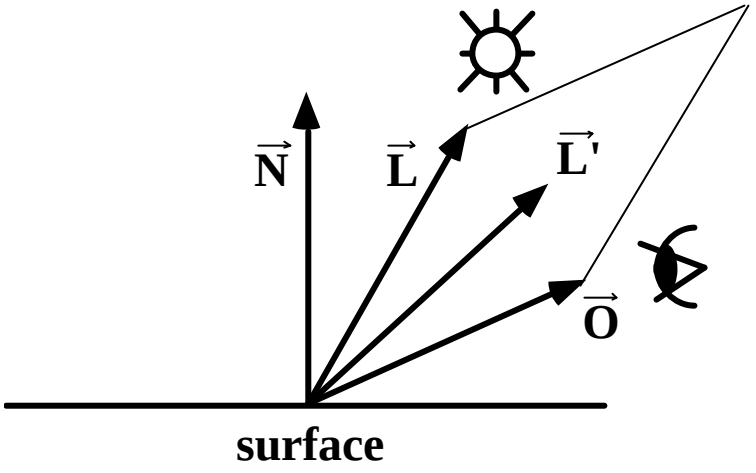


Figure 3.2-5. Les paramètres de la formule de réflexion spéculaire

Quelques cas pour comprendre la formule empirique de réflexion spéculaire

1. Plus la direction de l'observateur est symétrique à la direction de la lumière par rapport à la normale (voir Figure 3.2-6), plus l'intensité spéculaire est importante.

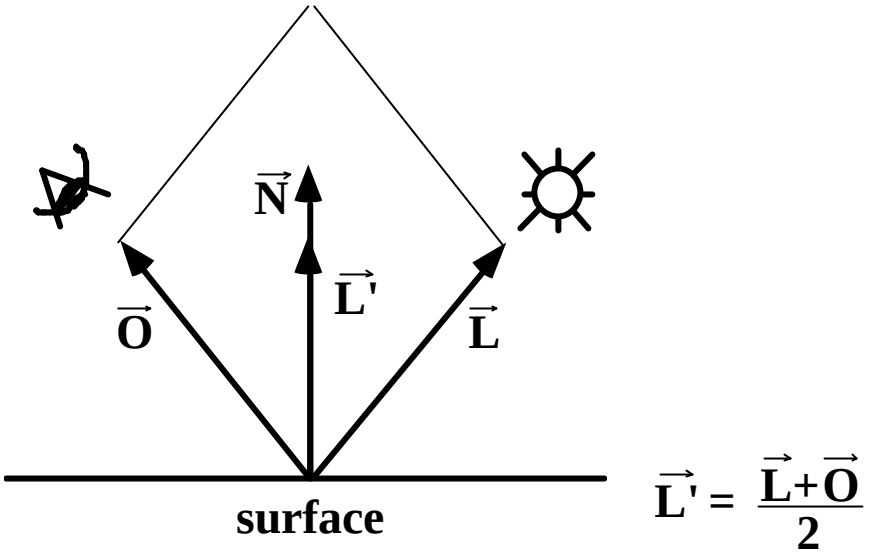


Figure 3.2-6. Cas de symétrie

L’ est parallèle à N donc:

$$N.L' = |N||L'| \cos 0 = |L'|$$

C'est la valeur maximale d'intensité

2. L'intensité spéculaire est moins importante quand la lumière est tangentielle à la surface (Figure 3.2-7) et plus importante quand la lumière est perpendiculaire à la surface (Figure 3.2-8).

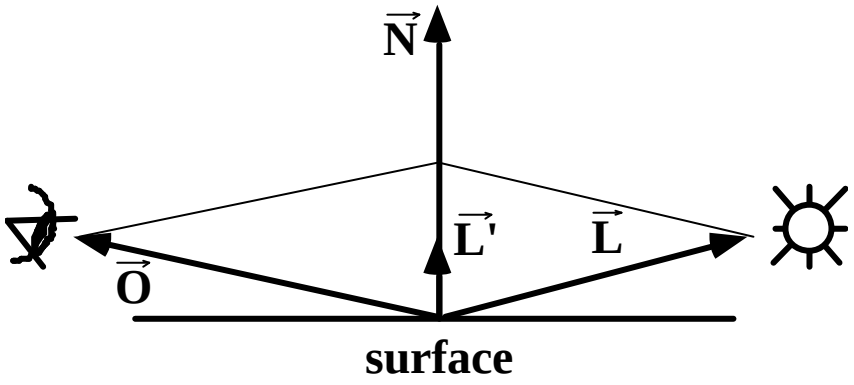


Figure 3.2-7. Cas où la lumière est presque tangente à la surface

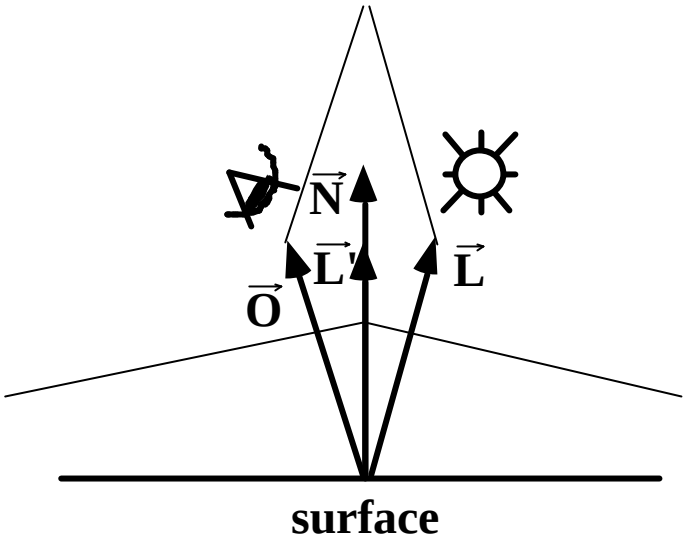


Figure 3.2-8. Cas où la lumière est presque perpendiculaire à la surface

3. Si l'angle entre N et L' est proche de 90^o (voir Figure 3.2-9), le cosinus est proche de 0 et l'intensité spéculaire est quasiment nulle (pas de reflet).

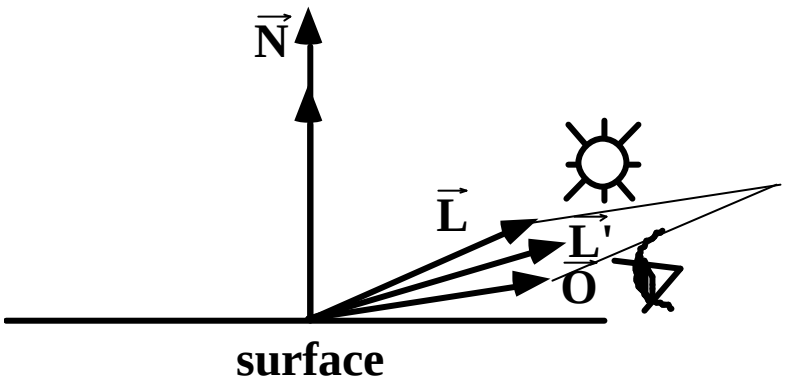


Figure 3.2-9. Cas d'un angle entre N et L' proche de 90^o

Finalement, si on considère s sources de lumière, on peut énoncer l'illumination totale selon Phong avec la formule suivante:

$$I = I_a + k_d \sum_{i=1}^{s} I_i N \cdot L_i + k_s \sum_{i=1}^{s} I_i (N \cdot L_i')^n$$

Il y a deux produits scalaires dans cette équation et Phong a développé une méthode assez efficace pour leur calcul incrémental le long d'une ligne de balayage.

Il faut aussi noter que la formule ci-dessus ne tient pas compte de la couleur. Dans la réalité, une telle formule est valable pour chaque couleur. Ainsi, en RGB, on aura une intensité pour le rouge, le vert et le bleu.

3.2.2.2 Le modèle théorique de Torrance et Sparrow

Bien que le modèle de Phong soit très réaliste, la composante de réflexion spéculaire n'est pas exacte et ceci a un effet très discernable surtout pour les objets non métalliques. La raison de cette inexactitude du modèle est due principalement au fait que l'intensité du reflet ne change pas avec la direction de la source, ce qui est incorrect. Ce défaut est évidemment amplifié dans une séquence d'animation.

Il existe donc des modèles physiques plus réalistes dont celui de Cook et Torrance. Dans ce modèle, la surface a dessiner est supposée formée d'une collection de très petits miroirs qui sont placés comme des facettes au hasard sur toute la surface (voir Figure 3.2-10).

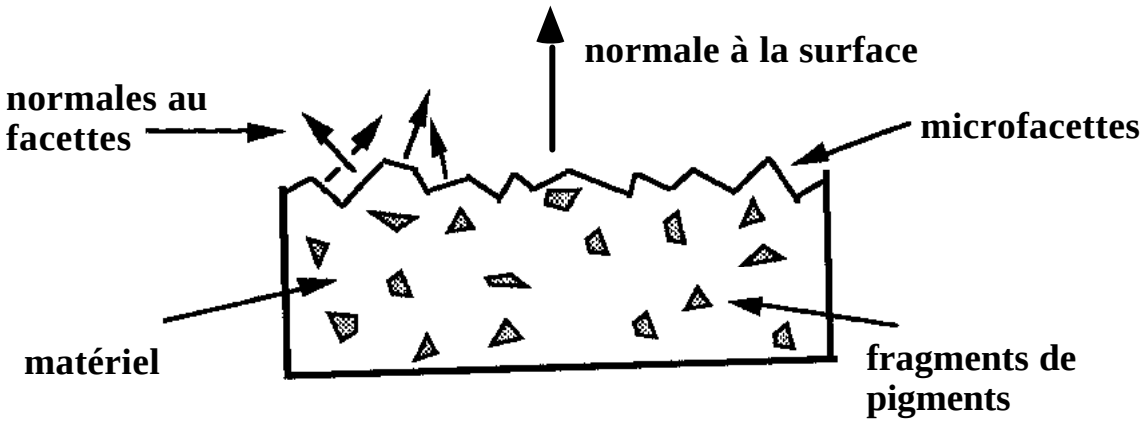


Figure 3.2-10. Microfacettes

Cela signifie que la matière (plastique en particulier) est un composé qui, même si la normale moyenne à la surface est bien perpendiculaire, est formé de microfacettes ayant des normales dans des directions très variables. C'est cette microstructure qui est responsable des reflets. La composante spéculaire est alors considérée comme l'ensemble des réflexions venant de toutes les facettes orientées dans la direction de L'_j. Le montant de la réflexion spéculaire R_s est calculé comme:

$$R_s = \frac{1}{p}\frac{\mathbf{DGF}}{(\mathbf{N}\cdot\mathbf{L})(\mathbf{N}\cdot\mathbf{O})}$$

où N est le vecteur unitaire normal à la surface, O la direction de l'oeil, et L celle de la lumière. D est la fonction de distribution des microfacettes de la surface, G le montant par lequel les facettes s'ombragent et se masquent mutuellement et F le facteur de Fresnel qui tient compte du fait que suivant l'angle de la lumière, la couleur de la réflexion spéculaire est plus dépendante de la couleur de la source de lumière que de la couleur de la surface éclairée. Les grandeurs D, G et F sont calculées ainsi:

$$D = \frac{1}{m^2 \cos^4 \alpha} e^{-\frac{tg^2\alpha}{m^2}}$$

où: α est l'angle entre H et N

H est la direction de reflet maximum calculée comme:

$$H = \frac{L + O}{|L + O|}$$

m est la racine carrée de la pente moyenne des microfacettes

La Figure 3.2-11 montre des éclairages de différentes matières en tenant compte d'un tel modèle physique.

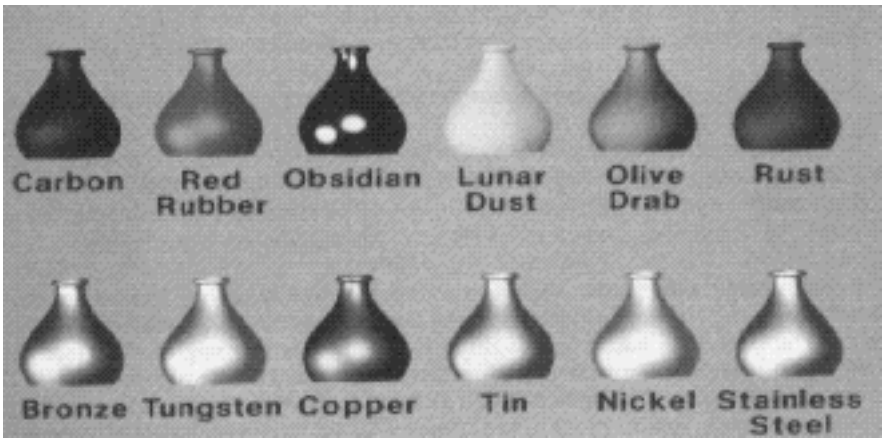


Figure 3.2-11. Illumination de différentes matières

3.2.3 La répartition de la lumière sur les surfaces

3.2.3.1 Le principe de la répartition de lumière

Le calcul de la répartition de l'illumination sur une surface est très complexe et peut s'avérer extrèmement coûteux en termes de temps-machine. Pour chaque type de description d'objets (par facettes, par élément de surfaces bicubiques, par surface algébrique), cette répartition peut être calculée en utilisant les modèles d'illumination en un point vus à la section précédente. Cependant, ces modèles de donnent pas le moyen réel de calculer l'illumination en chaque point de la surface, mais seulement l'intensité de lumière en des points spécifiques. En fait les techniques de répartition de lumière sont différentes selon les types de modélisation d'objets. Pour des objets basés sur des facettes polygonales, les trois méthodes les plus répandues sont les méthodes de Lambert (répartition constante), de Gouraud et de Phong.

3.2.3.2 Le modèle constant

Ce modèle n'implique qu'un seul calcul d'intensité pour chaque polygone, généralement au centre (Figure 3.2-12).

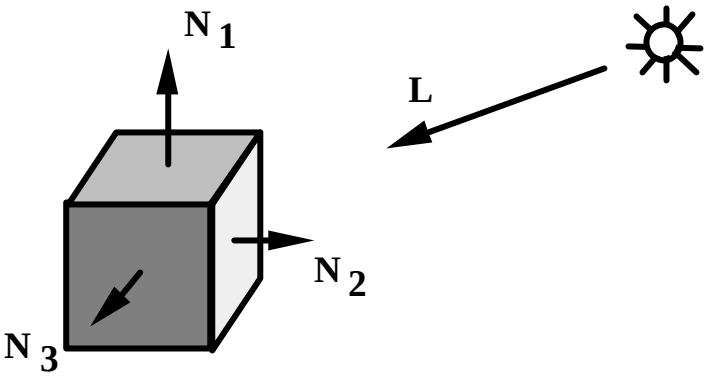


Figure 3.2-12. Le modèle constant

Il requiert pourtant les hypothèses suivantes:

1. la source de lumière est à l'infini (lumière directionnelle)

2. l'observateur est à l'infini (caméra parallèle)

3. les objets à représenter sont des polyèdres (p.e. cube, tétraèdre, parallélépipède).

On peut remarquer que les deux premières hypothèses sont requises pour assurer que les produits scalaires $N{\cdot}L_j$ et $N{\cdot}L_j'$ sont constants dans les calculs d'intensité. La troisième hypothèse est là car chaque facette polygonale d'un objet va avoir nécessairement une intensité différente de celle d'à côté, ce qui donne de bons résultats pour un cube, mais pas pour une sphère. La Figure 3.2-13 montre un exemple en fil de fer et la Figure 3.2-14 le même exemple avec une illumination utilisant le modèle constant.

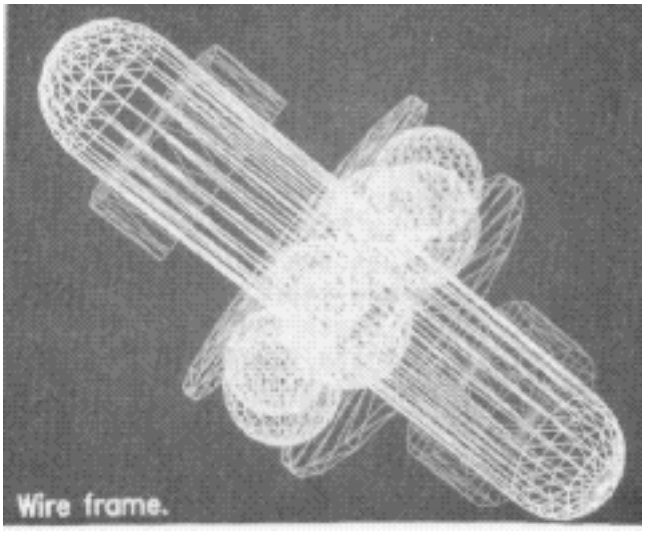


Figure 3.2-13. Objet en fil de fer

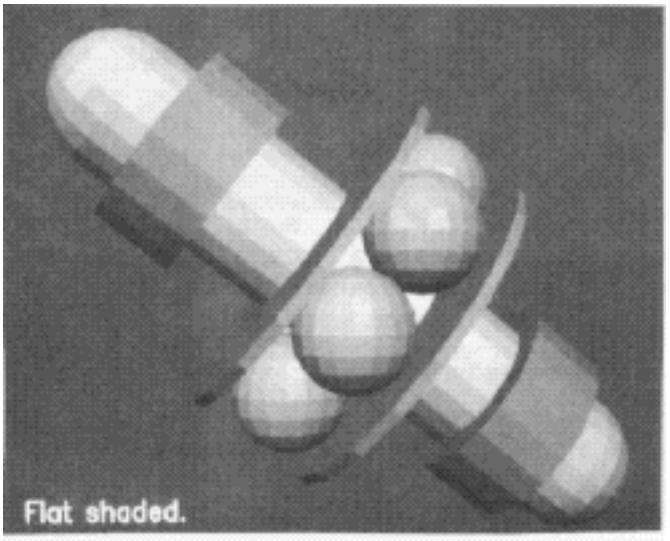


Figure 3.2-14. Objet illuminé selon le modèle constant

3.2.3.3 Le modèle de Gouraud

Gouraud (1971) a introduit une méthode de répartition de lumière qui élimine les discontinuités du modèle de Lambert. Cependant, on observe un assez fort effet de Mach. Cet effet bien connu des physiciens est responsable d'une apparence de bande plus claire ou plus foncée dans les environs d'une flexion de la surface illuminée, c'est à dire lorsque la dérivée de la fonction de répartition change. Le principe du modèle de Gouraud est le suivant:

1. Pour chaque sommet commun aux différents polygones, la normale à chaque polygone est calculée comme un vecteur perpendiculaire au plan de ce polygone.

2. Pour chaque sommet, une unique normale est calculée comme la moyenne des normale obtenues précédemment (voir Figure 3.2-15): $N_t = \sum_{i=1}^{m} \frac{N_i}{m}$

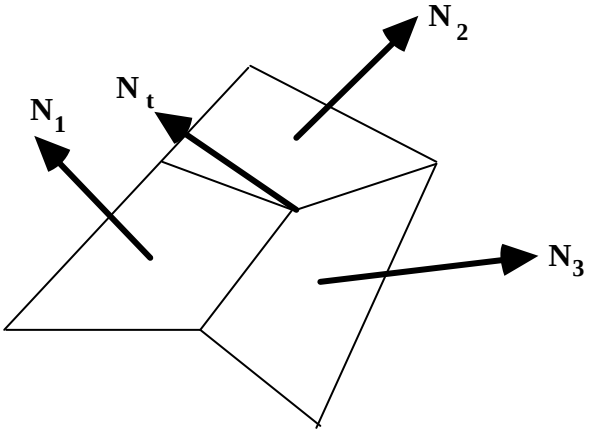


Figure 3.2-15. Calcul d'une normale au sommet

3. Les intensités aux sommets sont calculées en utilisant les normales aux sommets et un des modèles vus à la section précédente.

4. Comme chaque polygone a une illumination différente à chaque sommet, l'illumination en un point intérieur dun polygone est trouvé par une interpolation linéaire des intensités des sommets le long de chaque arête et entre les arêtes le long de chaque ligne de balayage (voir Figure 3.2-16). On suppose généralement que le modèle de Gouraud s'applique à un algorithme du type Watkins (1970).

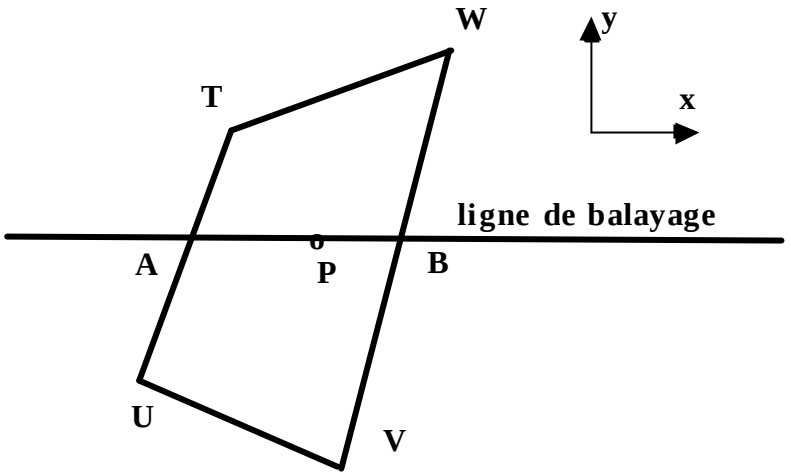


Figure 3.2-16. Interpolation des intensités selon Gouraud

La Figure 3.2-17 nous montre notre objet illuminé avec le modèle de Gouraud.

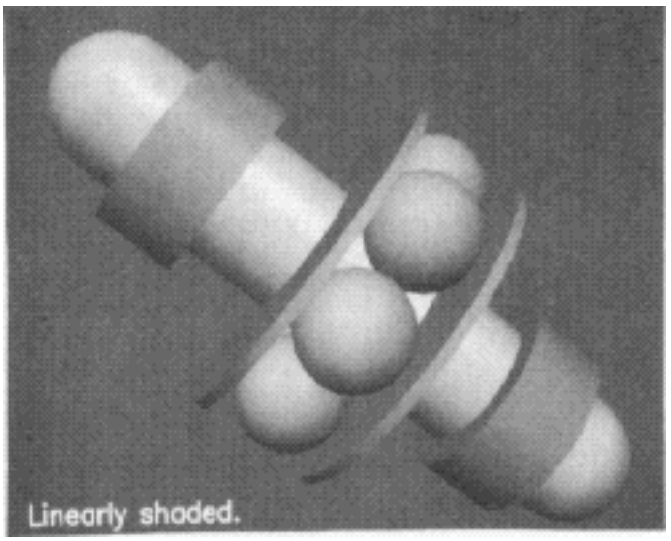


Figure 3.2-17. Objet illuminé selon le modèle de Gouraud

3.2.3.4 Le modèle de Phong

Bui-Tuong Phong (1975) a proposé une interpolation du vecteur normal à la surface à la place de l'interpolation des intensités (comme montré à la Figure 3.2-18). Avec cette approche, l'illumination en un point est recalculée à chaque pixel à partir de la normale interpolée. Avec cette méthode de Phong, on a une bien meilleure approximation de la courbure de la surface et les reflets dus à la

simulation de la réflexion spéculaire sont bien mieux rendus. Cependant, la méthode est beaucoup plus coûteuse, puisqu'il faut calculer les trois composantes de la normale, normaliser celle-ci avant d'évaluer la fonction d'illumination. Il faut aussi signaler que l'effet de Mach est réduit, mais peut encore être observé, car l'algorithme ne garantit pas la continuité de la première dérivée de la fonction de répartition de lumière.

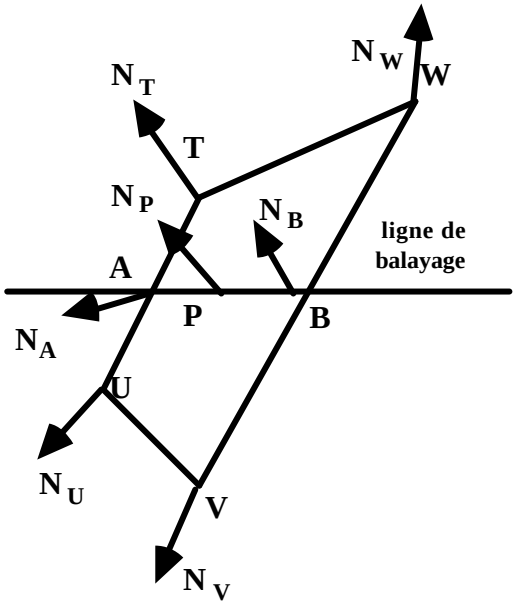


Figure 3.2-18. Interpolation des normales selon Phong

3.3 L'ombre portée

3.3.1 Le rôle de l'ombre portée

Les sources de lumière jouent un rôle fondamental dans le réalisme des images. Cependant, il faut remarquer qu'à moins de placer une source unique de lumière à l'emplacement de l'observateur, il est indispensable de tenir compte de l'ombre portée. Malheureusement, le calcul de l'ombre portée est très coûteux en termes de temps-machine. C'est pour cette raison que les réalisateurs de films d'animation choisissent souvent une illumination très diffuse (ciel couvert, par exemple) pour limiter l'effet de manque d'ombre portée. En fait, la plupart des algorithmes de production d'ombres portées ont de sévères limitations sauf le lancer de rayons (voir Section 3.4) qui est une technique elle-même coûteuse en temps CPU. On peut classer les techniques de production d'ombres portées en quatre catégories:

1. Techniques basées sur un algorithme de coupage de polygones; les deux principaux algorithmes sont ceux de Nishita et Nakamae (1974) et de Atherton et al. (1978).

2. La technique des volumes d'ombres introduite par Crow (1977).

3. Les techniques basées sur un algorithme de mémoire de profondeur (z-buffer); les deux principaux algorithmes sont ceux de Williams (1978) et de Brotman et Badler(1984).

4. La technique de lancer de rayons (voir Section 3.4)

3.3.2 La technique des volumes d'ombre

Un volume d'ombre est la région de l'espace où un objet intercepte la lumière. Ce volume est théoriquement illimité, mais dans les faits on le réduit généralement à son intersection avec le volume de vue comme le montre la Figure 3.3-1. Les polygones qui bordent le volume d'ombre sont alors additionnés à la liste des polygones d'affichage comme les projections des polygones formant l'objet. Pendant le processus d'affichage, les polygones d'ombre sont considérés comme invisibles et lorsqu'ils sont traversés, œci produit une transition. En inspectant correctement ces transitions, on peut générer les bonnes ombres portées.

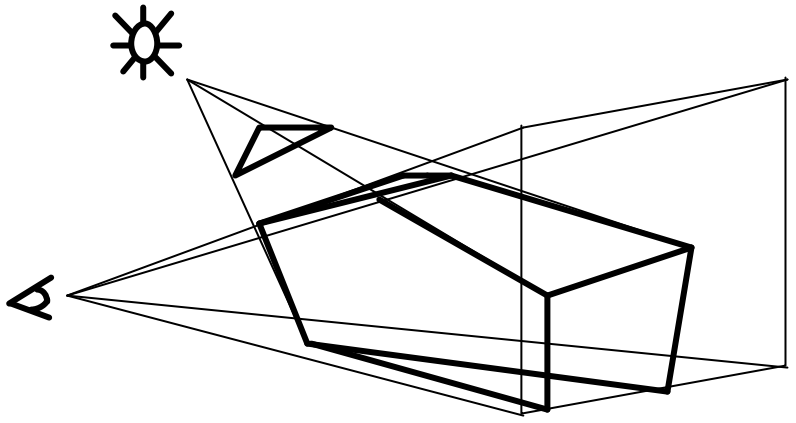


Figure 3.3-1. Volume d'ombre

3.3.3 Un algorithme basé sur le coupage de polygones dans l'espace-objet

Cet algorithme proposé par Atherton et al. se découpe en trois phases:

1. On place temporairement l'observateur à chacune des sources de lumière pour trouver les descriptions d'ombre.

2. En utilisant un algorithme de lignes cachées dû à Weiler et Atherton, les polygones éclairés sont détectés. Ces polygones sont ceux qui ne sont pas dans l'ombre et ils sont déterminés en considérant les surfaces cachées à partir de la source de lumière.

3. Les polygones éclairés sont ajoutés aux polygones originaux.

La Figure 3.3-2 nous montre le principe.

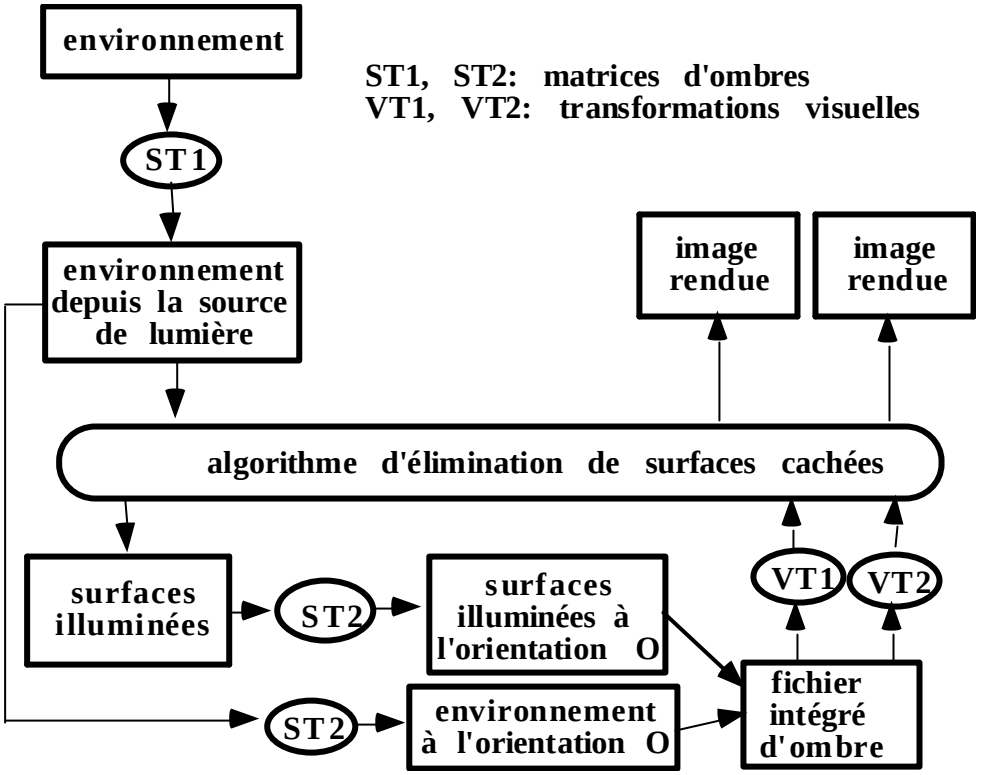


Figure 3.3-2. Principe de l'algorithme d'Atherton

3.3.4 Ombres par z-buffer

L'algorithme introduit par Williams (1978) fonctionne ainsi:

1. Une vue de la scène est construite depuis le point de vue de la source de lumière. Les valeurs z sont calculées et stockées dans un z-buffer d'ombre séparé (**shadow buffer**); les valeurs de couleur sont ignorées

2. Une vue de la scène est construite depuis le point de vue de la caméra. La profondeur en chaque pixel est comparée avec celle dans le z-buffer de l'observateur. Si la surface est visible, une transformation linéaire est utilisée pour faire correspondre les points <X,Y,Z> dans le système de l'observateur en coordonnées X',Y',Z' dans le système de la source de lumière. La valeur Z' est comparée avec la valeur Z en <X',Y'> dans le z-buffer d'ombre. Deux cas sont alors possibles:

    a) Le point n'est pas visible de la source de lumière, donc il est dans l'ombre (noir)

    b) Le point est visible de la source de lumière. Il est rendu selon le modèle de lumière en <X,Y>

Il faut encore remarquer que ces algorithmes produisent des ombres uniformes. Il est possible de produires des ombres plus floues, donc plus naturelles et des pénombres avec des méthodes beaucoup plus complexes. La Figure 3.3-3 nous montre un exemple.

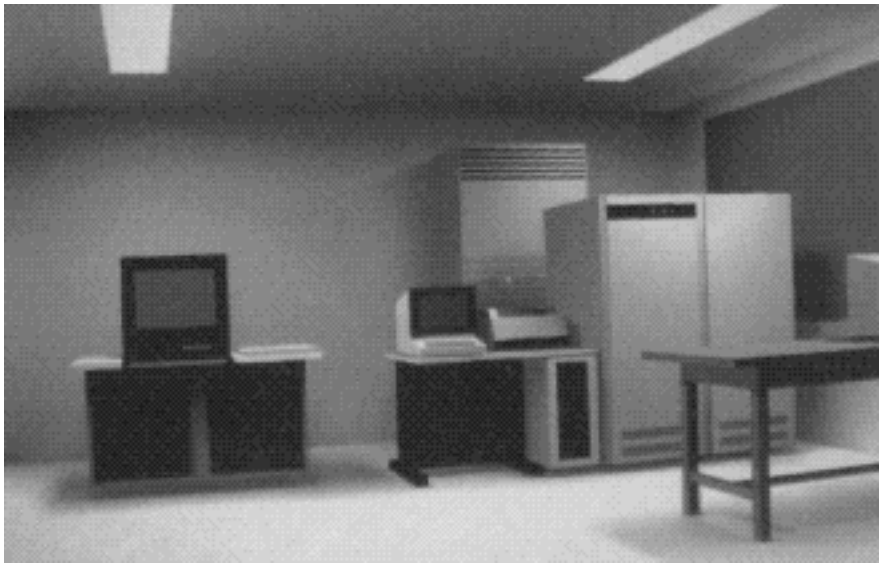


Figure 3.3-3. Exemple de pénombres

3.4 Le lancer de rayons

3.4.1 Le principe du lancer de rayons

Le lancer de rayons (en anglais ray tracing) est une ancienne technique basée sur la simulation numérique d'optique géométrique. Intuitivement, on peut considérer une méthode dans laquelle les rayons lumineux seraient tracés de la source de lumière, suivant leur chemin jusqu'à l'observateur. Cette approche serait catastrophique et même impossible car seuls quelques rayons arrivent à l'observateur. Pour cette raison, on préfère renverser la direction de propagation des rayons, en traçant les rayons à partir des objets, comme montré à la Figure 3.4-1.

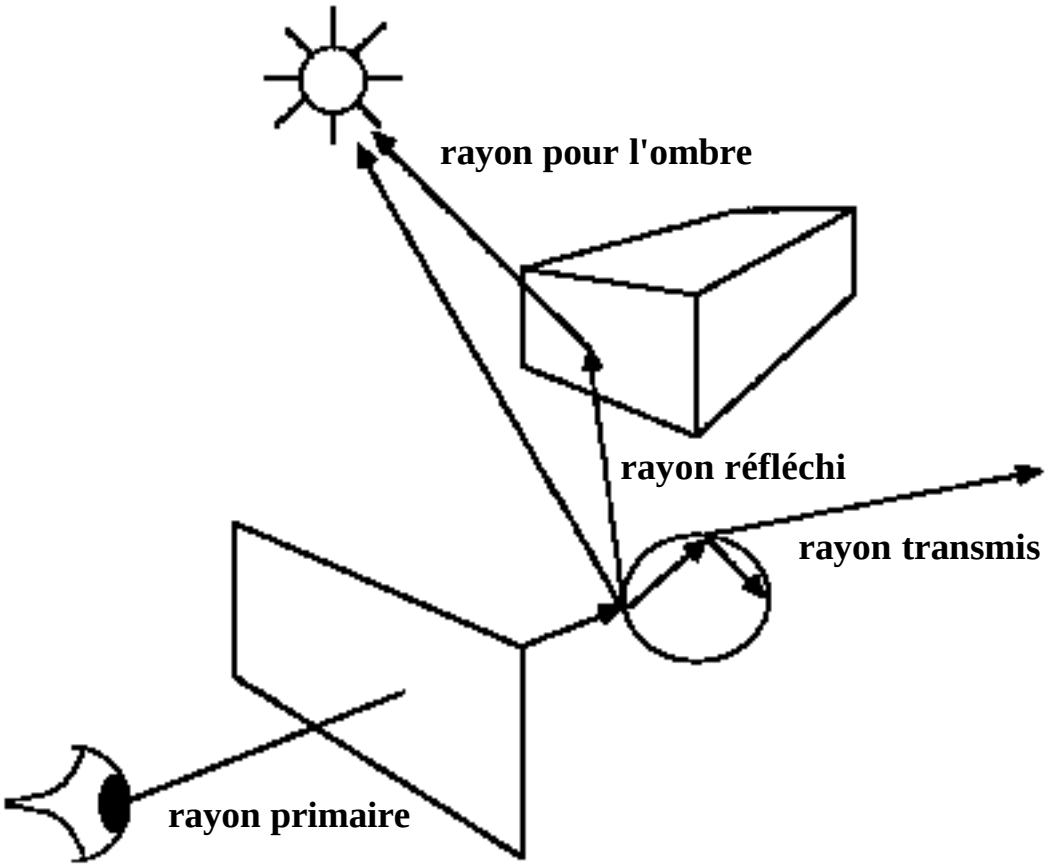


Figure 3.4-1. Principe du lancer de rayon

Un algorithme de lancer de rayons (ray tracing) revient à tirer des rayons à partir de l'observateur pour chaque pixel, calculer les intersections de ces rayons avec les objets de la scène et obtenir les informations photométriques nécessaires pour colorier le pixel courant.

A chaque surface touchée par un rayon, de nouveaux rayons peuvent être engendrés. Un rayon pour l'ombre est ainsi tiré à partir du point d'intersection vers chaque source de lumière pour déterminer si la source éclaire directement l'objet. Si oui, la contribution à l'intensité du rayon est calculée en se basant sur les propriétés de réflexion diffuse et spéculaire et l'intensité de la source. Des rayons réfléchis et transmis peuvent être aussi engendrés. Pour ces nouveaux rayons, il faut évidemment appliquer récursivement le même processus pour déterminer les intersections de ces rayons avec d'autres surfaces. Pour chaque pixel, un arbre d'intersection doit être construit (voir

Figure 3.4-2). Le processus récursif s'arrête dans les cas suivants:

- quand un rayon quitte la scène
- quand un rayon heurte une surface ni spéculaire, ni transparente
- quand la contribution du rayon devient négligeable
- quand on atteint une profondeur limite de la récursivité

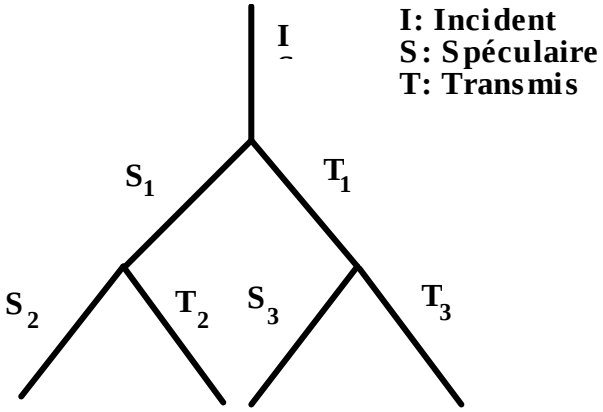


Figure 3.4-2. Arbre d'intersection

Lorque l'arbre a été créé, il est traversé en appliquant une équation à chaque noeud pour calculer l'intensité. Avec le processus récursif, l'intensité pour le noeud courant est obtenue quand tous les sous-noeuds ont été évalués. L'équation appliquée à chaque noeud pour calculer l'intensité est différente de celles vues à la Section 3.2.2.1. On utilise généralement celle de Whitted (1980) ou celle de Hall et Greenberg (1983) qui est très complexe. Nous ne présentons que la première basée sur les lois de la réflexion et la réfraction (loi de Snell-Descartes). Dans le modèle de Whitted, l'illumination est est donnée par l'équation:

$$I= I_a + I_d + I_s + I_t$$

où la lumière ambiante et la lumière diffuse sont calculées comme dans la formule de Phong. La lumière spéculaire est obtenue par l'équation:

$$I_s=k_sS$$

où S est la lumière spéculaire incidente (de direction R) et k_s une constante.

On a en plus une lumière transmise donnée par l'équation:

$$I_t = k_tT$$

où T est l'intensité du rayon transmis (de direction P) et k_t est une constante.

Les directions des rayons sont obtenues à partir des équations de la réflexion et de la réfraction (Snell-Descartes) où on considère l'angle d'incidence i, l'angle de réfraction r et l'indice de réfraction n du premier matériau relativement au second. On a:

1. Le rayon incident, la normale au point incident et le rayon réfracté sont dans un même plan.

2. Le rayon incident et le rayon réfléchi forment un même angle avec la normale:

$$r=i$$

On détermine ainsi la direction R du rayon réfléchi:

$$V' = \frac{V}{|V.N|}$$

$$R = V' + 2N$$

3. Le sinus de l'angle d'incidence est dans un rapport constant avec le sinus de l'angle de réfraction, ce qui s'énonce par:

$$\sin i = n \sin r$$

On détermine ainsi la direction P du rayon réfracté:

$$P = \frac{N + V'}{\sqrt{n^2 |V'|^2 - |V' + N|^2}} - N$$

La Figure 3.4-3 nous montre la situation du point de vue géométrique.

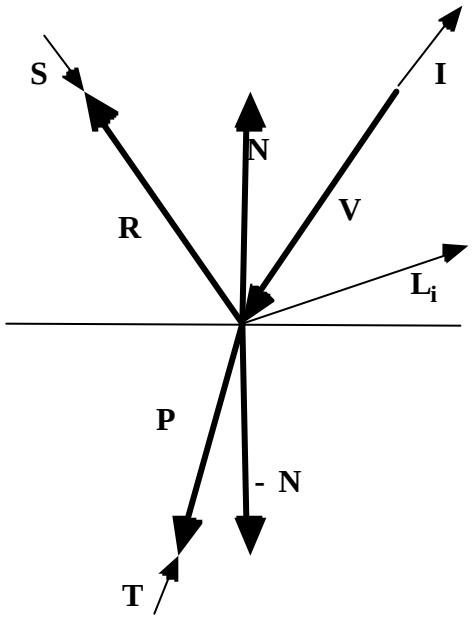


Figure 3.4-3. La réfraction

L'algorithme du lancer de rayons est extrêmement puissant et permet de traiter pratiquement tous les aspects du réalisme. Malheureusement, les calculs d'intersections sont très coûteux en temps de calculs vu le nombre de rayons lancés et seules des machines puissantes peuvent permettre de tels calculs dans un temps raisonnable. Il faut donc trouver des méthodes d'optimisation. Il y a plusieurs stratégies possibles: augmenter la vitesse de traitement de ces intersections, trouver des moyens de tester plus rapidement s'il y a possibilité d'intersection, diminuer le nombre de rayons lancés ou utiliser des rayons plus complexes que des droites (cônes, par exemple). La Figure 3.4-4 nous montre une classification de ces techniques. Dans la Section 3.4.2, on présente un certain nombre de méthodes pour calculer le plus rapidement possible les intersections avec différents types d'objets. La Section 3.4.3 est particulièrement dédiée au cas des solides CSG qui permet de calculer très rapidement une image formée uniquement de tels solides. Dans la Section 3.4.4, on verra comment tester plus rapidement les possibilités d'intersection et par conséquent diminuer le nombre de calculs

d'intersections. On ne traitera pas des cas où l'on diminue le nombre de rayons lancés, on peut seulement mentionner qu'il s'agit de méthodes où l'on génère des rayons secondaires que lorsqu'il contribuent significativement à la qualité de l'image, on a un niveau de récursion adaptatif. L'utilisation de rayons plus complexes qu'une droite est assez limitée et ne sera pas présentée ici.

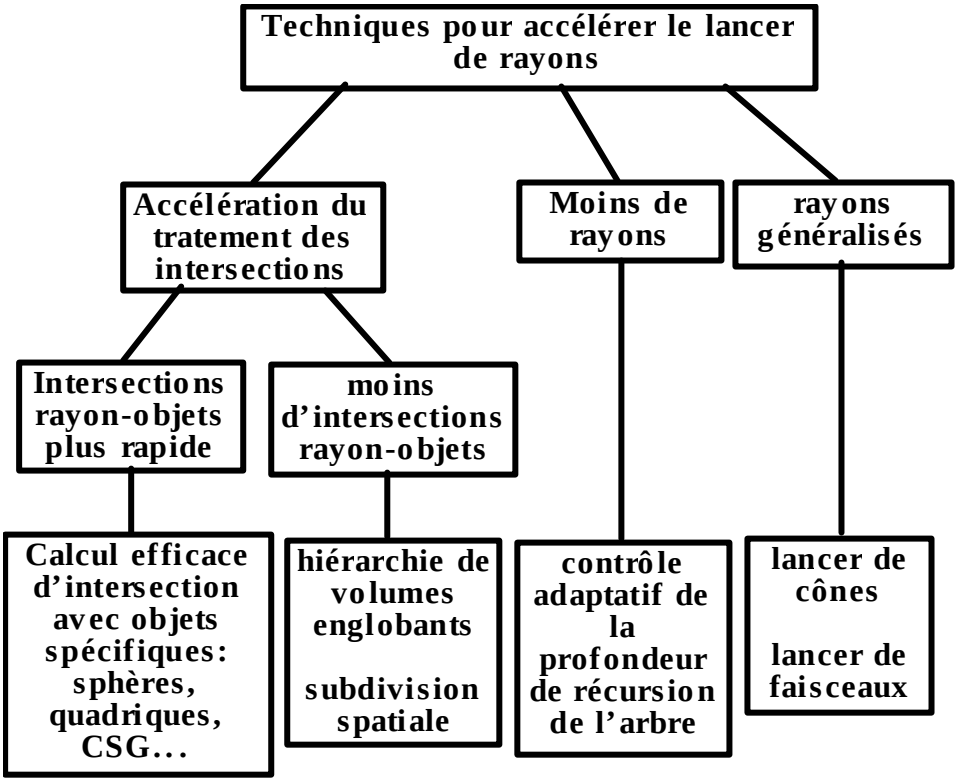


Figure 3.4-4. Une classification des techniques d'accélération

3.4.2 Le calcul des intersections pour certaines surfaces

3.4.2.1 Equation du rayon

Dans ces méthodes, on supposera que le rayon est toujours donné par l'équation:

Equation du rayon: $R(t) = R_0 + Vt$

avec: $R_0=<x_0,y_0,z_0>$: origine du rayon
$V=<x_V,y_V,z_V>$: vecteur unitaire de direction du rayon

L'équation du rayon en coordonnés cartésiennes est donnée par:

$$<x,y,z> = <x_0+x_Vt, y_0+y_Vt, z_0+z_Vt>$$

3.4.2.2 Lancer de rayons pour les sphères

C'est le cas le plus simple et les sphères représentées par lancer de rayons sont très communes. En effet, il suffit de tester la distance minimale d du rayon tracé au centre de la sphère $C=<x_C,y_C,z_C>$. Si cette distance est inférieure au rayon de la sphère, il y a intersection à calculer. En partant de

l'équation du rayon en coordonnés cartésiennes, la distance minimale d entre le centre de la sphère et le rayon est obtenu par l'équation:

$$d^2 = (x-x_C)^2+(y-y_C)^2+(z-z_C)^2 =$$
$$(x_0+x_Vt-x_C)^2 + (y_0+y_Vt-y_C)^2 + (z_0+z_Vt-z_C)^2$$

On obtient aisément le point d'intersection en remplaçant dans l'équation de la sphère le <x,y,z> tiré de l'équation du rayon. La Figure 3.4-5 nous montre un exemple.

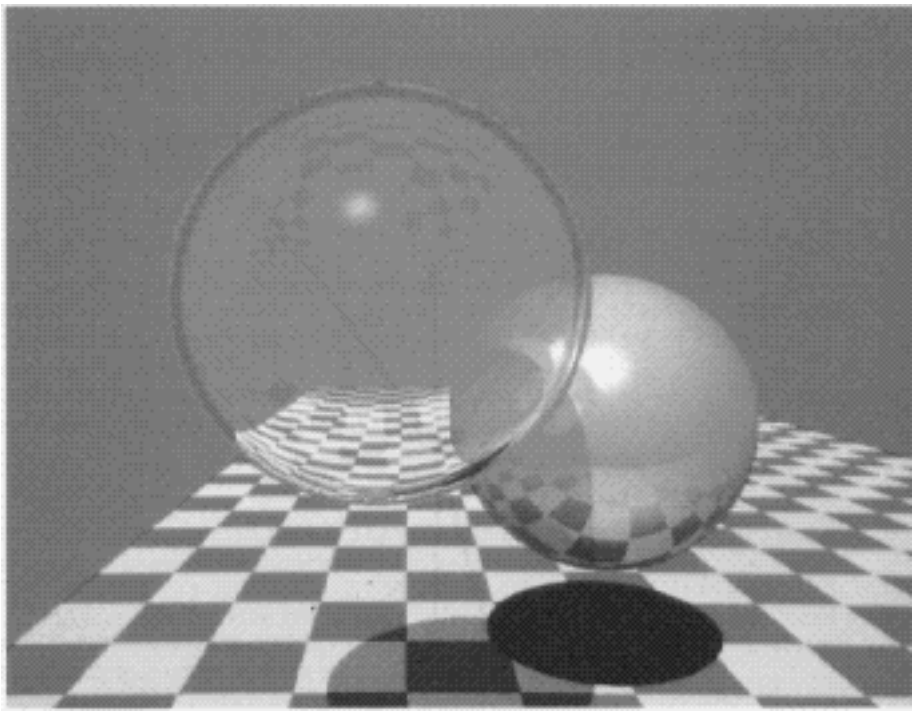


Figure 3.4-5. Exemple d'image par lancer de rayons

3.4.2.3 Lancer de rayon pour des plans

Le plan étant donné par son équation:

$$n \cdot R + D = 0$$

où n est le vecteur normal et R=<x,y,z> , on calcule l'intersection du rayon et du plan en substituant dans l'équation du plan R par R(t) de l'équation du rayon:

$$n \cdot R(t) + D = 0$$

On obtient:
$$t = \frac{-(n \cdot R_0 + D)}{n \cdot V}$$

qui nous permet de trouver le point d'intersection.

3.4.2.4 Lancer de rayons pour des surfaces à facettes polygonales

Pour de telles surfaces, le point d'intersection entre le rayon tracé et le plan de chaque polygone est calculé en premier, puis l'algorithme contrôle si le point est à l'intérieur du polygone selon un des algorithmes de la Section 3.1.2.3.

3.4.2.5 Lancer de rayons pour des surfaces algébriques

Une surface algébrique est définie par son équation:

$$P(x,y,z) = \sum_{ijk} a_{ijk} x^i y^j z^k = 0$$

En substituant l'équation cartésienne du rayon, nous obtenons une équation polynomiale en t à résoudre par des méthodes numériques.

3.4.2.6 Lancer de rayons pour des surfaces implicites

Une surface implicite est définie par:

$$F(x,y,z) = 0$$

on la résoudra donc comme une surface algébrique.

3.4.2.7 Lancer de rayons pour une surface libre

La méthode récursive de Whitted (1980) génère des sphères englobantes pour chaque élément de surface. Si la sphère a une intersection avec le rayon tracé, alors l'élément de surface est subdivisé en sous-éléments et des sphères englobantes sont produites pour chaque sous-élément. Le processus continue ainsi jusqu'à ce que la sphère intersectée soit inférieure à une dimension préétablie ou qu'il n'y ait plus d'intersection.

3.4.2.8 Lancer de rayon pour des surfaces quadriques

La Figure 3.4-6 nous montre les différents cas de surfaces quadriques.

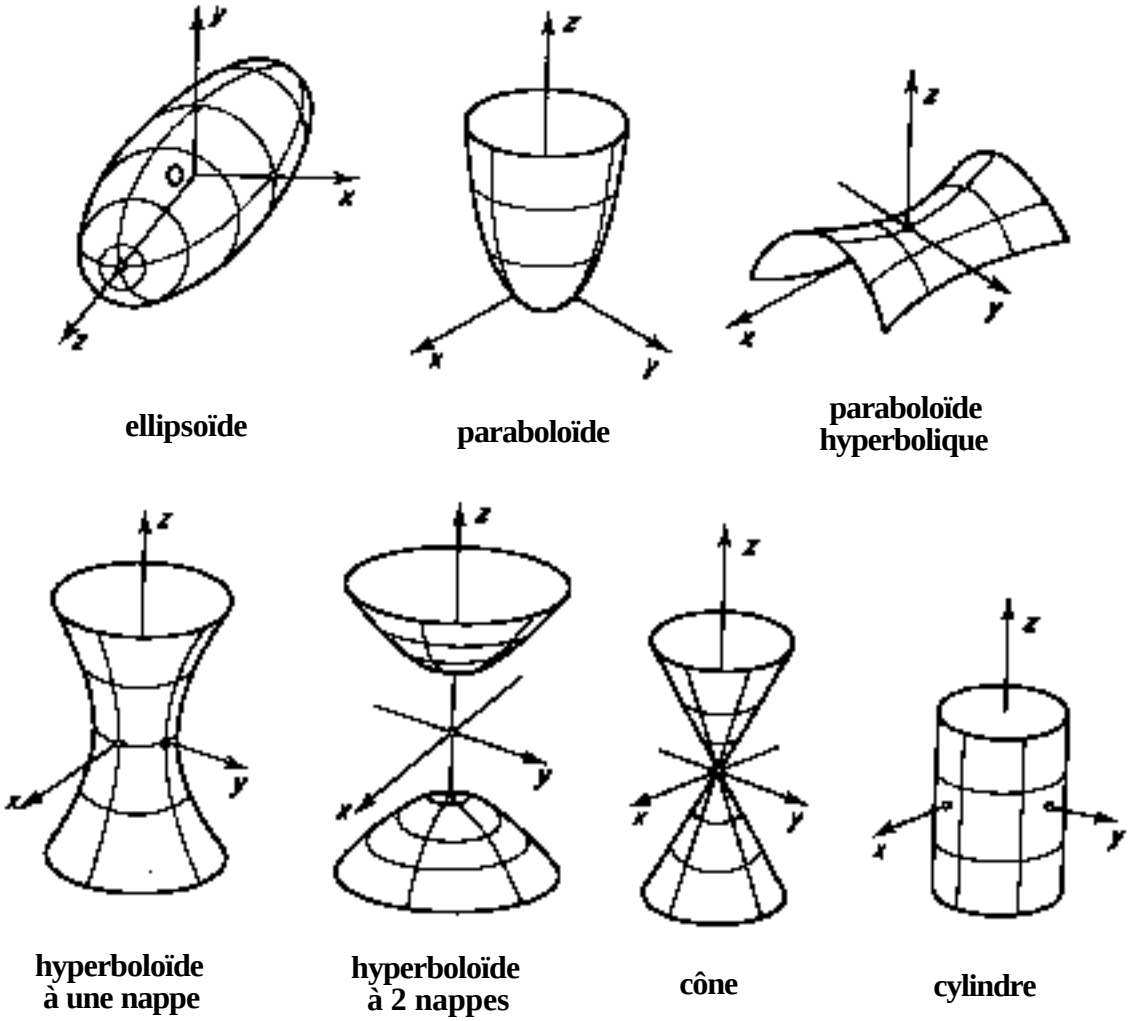


Figure 3.4-6. Les quadriques

Ces quadriques peuvent être représentées par une équation canonique implicite générale:

$$x^2 + y^2 + az^2 + bz + c = 0$$

sphère: $x^2 + y^2 + z^2 - 1 = 0$

cylindre infini: $x^2 + y^2 - 1 = 0$

cône infini: $x^2 + y^2 - z^2 = 0$

paraboloïde: $x^2 + y^2 + z = 0$

hyperboloïde à 2 nappes: $x^2 + y^2 - z^2 + 1 = 0$

hyperboloïde à une nappe: $x^2 + y^2 - z^2 - 1 = 0$

en remplaçant dans chaque équation de la quadrique <x,y,z> par leur valeur prise dans l'équation du rayon, on obtient facilement la valeur de t et par conséquent l'intersection.

3.4.3 Lancer de rayons pour les solides CSG

Le premier algorithme de lancer de rayons a été introduit en 1971 par la maison de production MAGI pour des objets définis sous la forme d'arbres CSG. Comme expliqué dans la section 2.3.5, le modèle CSG est représenté de façon interne comme un arbre binaire. Chaque feuille est un objet primitif et chaque noeud un objet composé obtenu en appliquant l'un des opérateurs ($\cup$,$\cap$,-) aux noeuds des sous-objets définis par les branches gauche et droite du noeud; la racine correspond évidemment à l'objet entier. La Figure 3.4-7 nous montre un exemple.

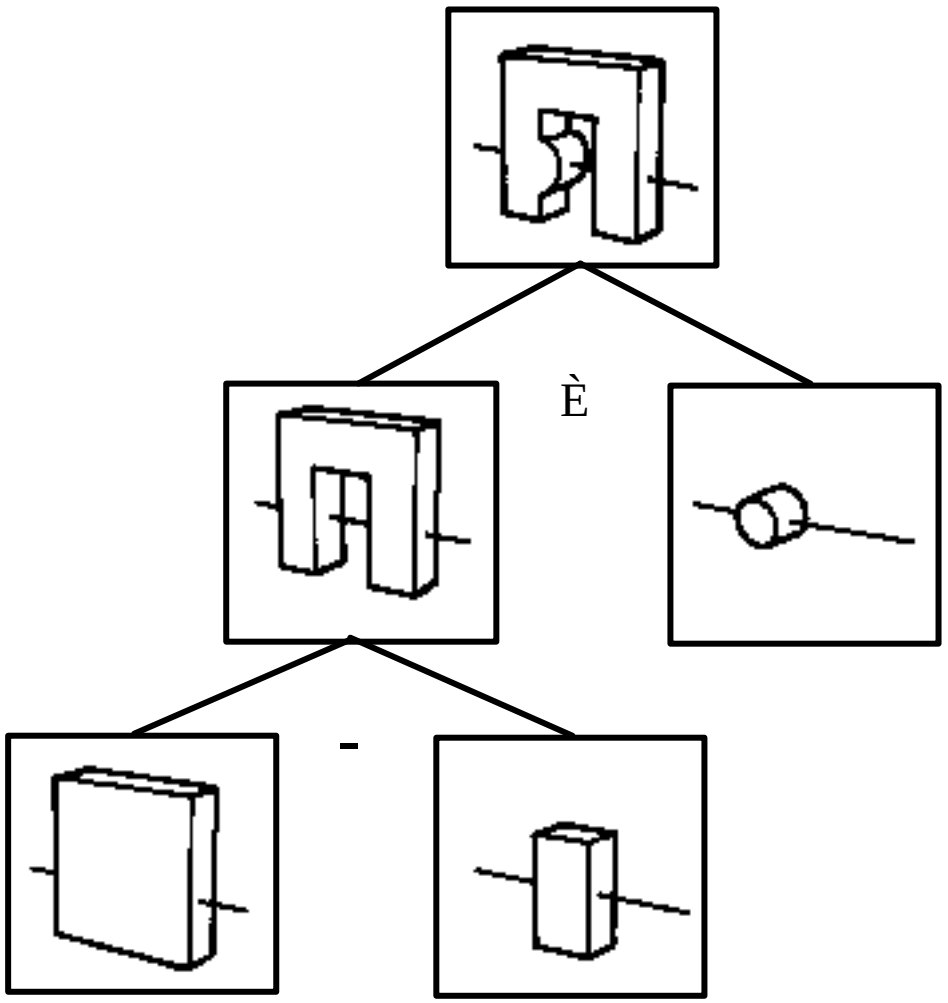


Figure 3.4-7. Un arbre CSG

Le lancer de rayons a un avantage certain sur les techniques de balayage et le z-buffer: il n'y a pas besoin d'évaluer la frontière de la représentation CSG. L'algorithme de base fonctionne ainsi: soit un rayon et un arbre binaire représentant une scène, le rayon est classifié par rapport au solide. La classification est basée sur l'information décrivant l'intersection entre le rayon et le solide. Elle désigne quelles parties du rayon sont à l'intérieur et quelles parties sont à l'extérieur du solide. Le processus commence à la racine de l'arbre CSG, descend récursivement vers le bas, classifie le rayon par rapport aux primitives et retourne l'arbre combinant les classifications des sous-arbres gauche et droite. Pour des primitives non-convexes, la classification est complexe, mais pour des primitives convexes, comme les cylindres, les cônes, les boîtes et les sphères, il n'y a que 4 cas possibles d'intersection rayon-solide:

1. Le rayon manque la primitive
2. Le rayon touche la primitive
3. Le rayon entre et sort de la primitive en deux points différents
4. Le rayon est situé dans la face d'une primitive

Pour classifier les rayons, on peut utiliser les diagrammes de Roth, il s'agit d'une ligne qui va de -8 à +8 et où chaque point représente une valeur du paramètre t du rayon. Chaque point sur la ligne est soit à l'intérieur, soit à l'extérieur. Les transitions de l'intérieur vers l'extérieur et l'inverse se trouveront là où il y a intersection entre le rayon et les primitives formant l'objet. La Figure 3.4-8 nous montre un exemple. Pour obtenir le diagramme de Roth final d'un objet, il faut combiner les diagrammes des primitives en descendant récursivement l'arbre. La Figure 3.4-9 explique les règles de combinaisons des classifications.

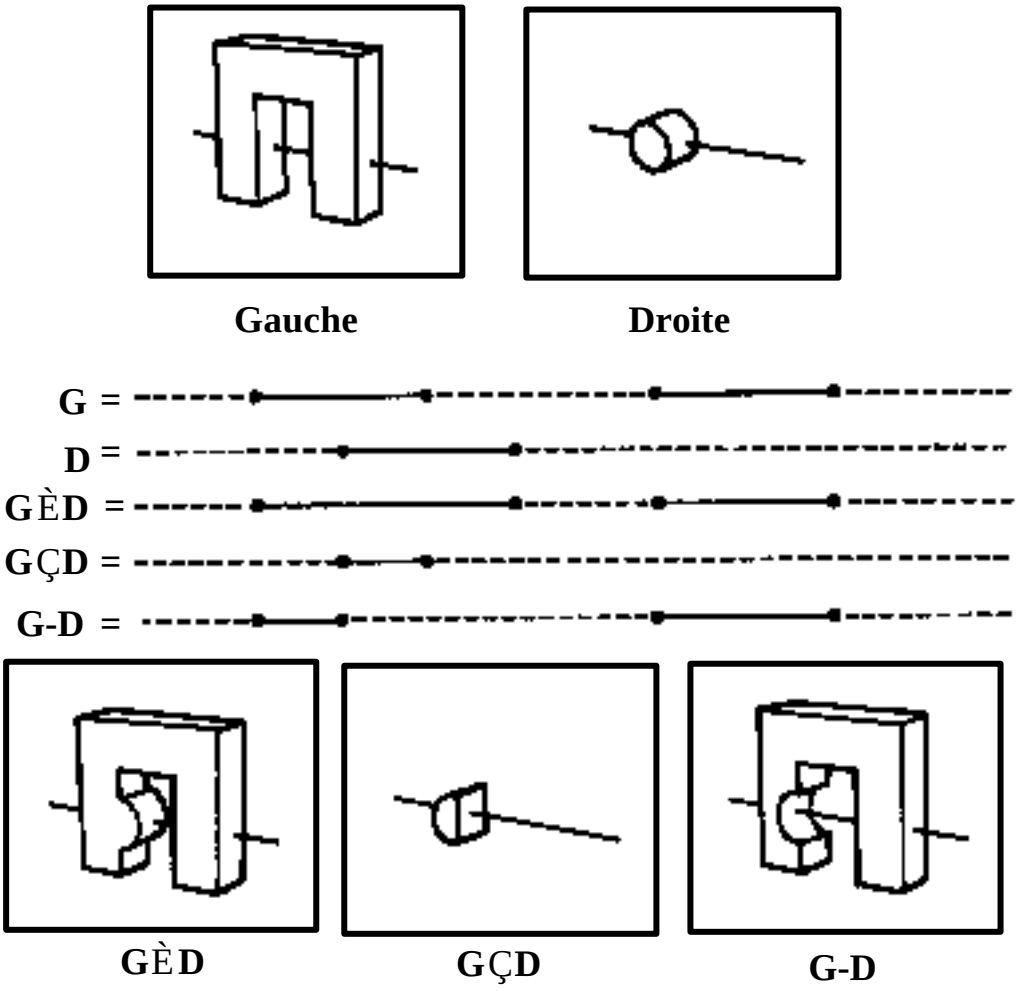


Figure 3.4-8. Exemple de diagramme de Roth

opérateur	gauche	droite	composé
È	**int**	**int**	**int**
	int	**ext**	**int**
	ext	**int**	**int**
	ext	**ext**	**ext**
Ç	**int**	**int**	**int**
	int	**ext**	**ext**
	ext	**ext**	**ext**
	ext	**ext**	**ext**
-	**int**	**int**	**ext**
	int	**ext**	**int**
	ext	**int**	**ext**
	ext	**ext**	**ext**

Figure 3.4-9. Règles de combinaison de classifications pour les diagrammes de Roth

3.4.4 Optimisation par volumes englobants

3.4.4.1 Hiérarchie de volumes englobants

Un des moyens les plus efficaces d'optimiser le lancer de rayons et de réduire le nombre d'intersections à calculer. Pour cela, on peut englober les objets de volumes plus simples et tester l'intersection avec ces volumes. S'il n'y a pas d'intersection avec un volume englobant, on est certain qu'il n'y en a pas avec les objets situés dans le volume et par conséquant, on va éviter de chercher les intersections des rayons avec ces objets.

On peut être encore plus efficace et utiliser une hiérarchie de volumes englobants organisés en arborescence. Chaque niveau de la hiérarchie consiste en volumes englobant les volumes des niveaux inférieurs, les feuilles de l'arbre étant constituées des objets eux-mêmes. Ainsi, comme les volumes englobants des descendants d'un noeud se trouvent entièrement dans le volume englobant du parent, il n'y a besoin de tester les intersections avec les volumes englobants des enfants que lorsqu'il y a intersection avec le volume englobant du parent.

L'algorithme général d'intersection entre le rayon et une collection d'objets organisée en une hiérarchie de volumes englobants s'exprime alors comme une procédure récursive:

```
procedure intersection (rayon, noeud)
        si le noeud est une feuille alors
                calculer l'intersection entre le rayon et l'objet
        sinon
                s'il y a intersection entre le rayon et le volume englobant alors
                        pour chaque enfant du noeud
                                intersection (rayon, enfant)
```

3.4.4.2 Sélection de volumes englobants

Le problème est maintenant de trouver les meilleurs types de volumes englobants. Comme nous l'avons déjà vu à la Section 3.4.2, l'intersection d'un rayon avec une sphère est particulièrement simple à déterminer et la sphère est donc une candidate tout trouvée pour un volume englobant. Cependant, elle a le défaut de ne pas correspondre à des objects longs et minces, comme le montre la Figure 3.4-10.

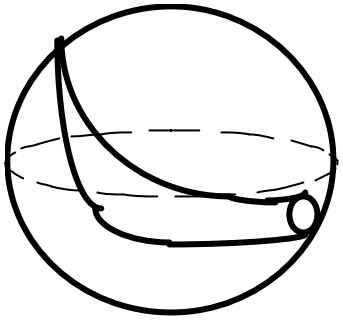


Figure 3.4-10. Sphère englobante pour un objet du type-banane

En fait, la sélection de bons volumes englobants n'est pas simple, en effet, ils doivent satisfaire deux critères qui peuvent être contradictoires:

1. Ils doivent être le plus proche possible de l'objet afin de minimiser la région où le test d'intersection s'avère positif avec le volume englobant, mais négatif avec l'objet.

2. Ils doivent permettre un test très efficace d'intersection

On utilise souvent des boîtes englobantes qui sont en général meilleures pour le premier critère que la sphère, mais moins bon pour le second.

Une boite est un parallélépipède rectangle; un test en 3 dimensions coûte relativement cher. D'autre part, il est assez simple de transformer le test de boite englobante tridimensionnelle en un test de signe en deux dimensions. En utilisant des translations et des rotations, on peut faire coïncider le rayon avec l'axe z. Les mêmes transformations sont alors appliquées à l'objet et à la boite englobante. Il y a intersection entre le rayon et la boite quand la condition suivante est remplie dans le système de coordonnées obtenus après translations et rotations:

$$x_{min}\, x_{max} < 0 \text{ and } y_{min}\, y_{max} < 0$$

où x_{min},x_{max},y_{min} and y_{max} sont les limites de la boite englobante transformée, comme le montre la Figure 3.4-11.

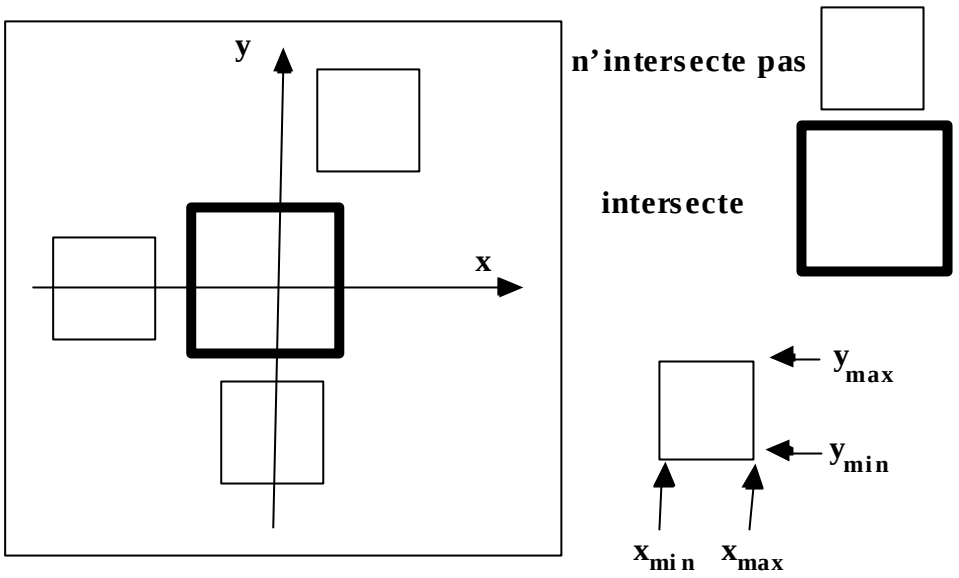


Figure 3.4-11.Test de boites englobantes en deux dimensions

La Figure 3.4-12 nous montre un exemple en 2D avec une hiérarchie de boîtes englobantes. Les 6 petites sphères sont entourées chacune d'une boite, tandis que chaque paire {3,4}, {5,6} et {7,8} sont entourées de boites et que finalement les 6 sont entourées d'une plus grande boîte. Le rayon primaire va donc tester au premier niveau pour l'objet 1, l'objet 2 et la boite englobant les 6 sphères. Comme le rayon traverse la boite de l'objet 2, un test d'intersection avec cet objet est nécessaire. Le rayon traverse aussi la boite englobant les 6 sphères, on va donc tester au second niveau les intersections avec les 3 boites englobant les paires. Seules les boites des paires {5,6} et {7,8} sont traversées et donc seules les boites des objets 5, 6, 7 et 8 sont testées. Finalement, comme seules les boîtes des objets 6 et 7 sont traversées, on ne testera que les intersections avec ces 2 objets.

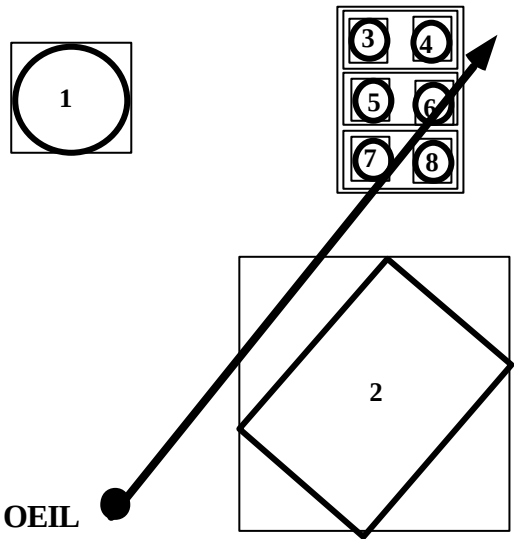


Figure 3.4-12. Boites englobantes

Kajiya (1986) a proposé un nouveau type de volume englobant qui peut s'approcher aussi près que possible de l'enveloppe convexe des objets. Les objets sont englobés avec des volumes construits par des plans arbitraires décrit par l'équation cartésienne du plan:

$$Ax + By + Cz = D$$

où N = <A,B,C> est le vecteur normal au plan et D la distance du plan à l'origine.

Kajiya définit un "**slab"** comme la région entre deux plans parallèles p_1 et p_2 caractérisés par la paire de valeurs de D: $\{D_1 , D_2\}$; la normale N est évidemment la même pour les deux plans et appelée normale à la paire de plans. Différents slabs englobant peuvent être définis simplement en changeant N. La

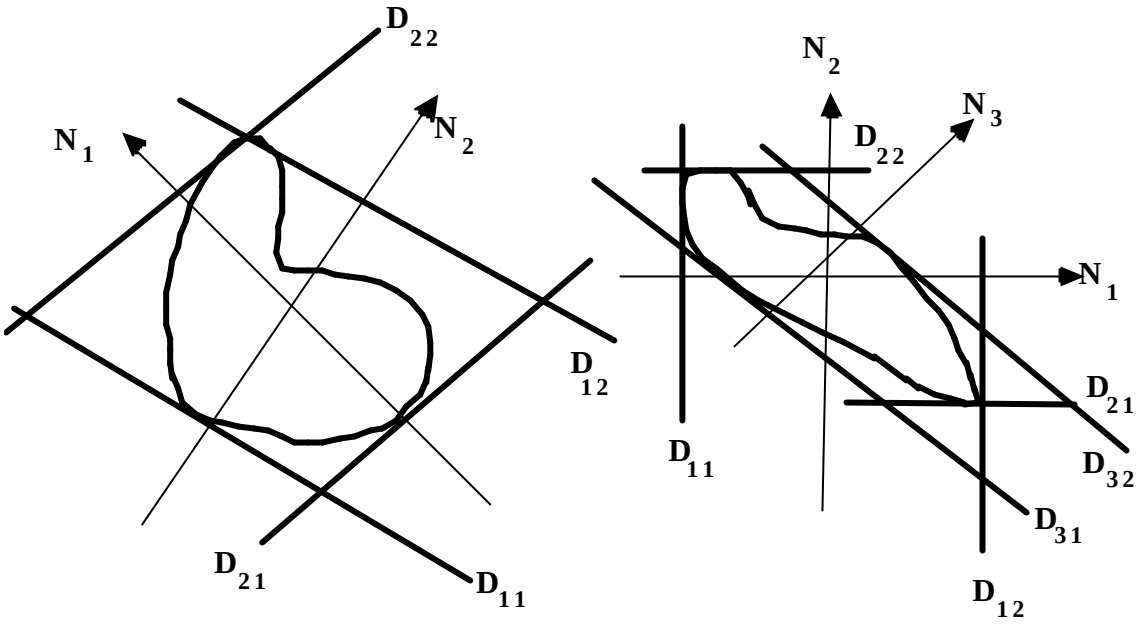
Figure 3.4-13 montre une représentation 2D des slabs.

Le volume englobant est obtenu par l'intersection d'un ensemble de slabs englobant (au minimum 3), ce qui implique un ensemble de normales à des paires de plans, par exemple:

{<1,0,0>, <0,1,0>, <0,0,1>} parallélépipède rectangle

$\{\frac{1}{\sqrt{3}}<1,1,1>, \frac{1}{\sqrt{3}}<-1,1,1>, \frac{1}{\sqrt{3}}<-1,-1,1>, \frac{1}{\sqrt{3}}<1,-1,1>\}$ parallélépipède à 8 côtés

Comme les volumes englobants existent dans le monde virtuel, une transformation est nécessaire (matrice 3x3 M et translation T). Cette transformation est associée à chaque objet dans la base de données.



N_i normale à la i-ième paire de plans

D_{i1} D_{i2} distances pour la i-ième paire de plans

Figure 3.4-13. Exemples de slabs en deux dimensions

Pour des polyèdres, les volumes englobants sont déterminés en calculant le volume englobant les sommets. On distingue trois étapes:

1. Transformation des sommets P_k: $P_k' = P_k M + T$

2. Projection de chaque sommet transformé sur chaque normale N_i: $d_{ik} = P_k' N_i$

3. Calcul de la paire $\{D_1 , D_2 \} = \{\min\{d_{ik}\}, \max\{d_{ik}\}\}$

Pour une surface implicite, la méthode de Kajiya consiste a appliquer la transformation M à chaque point P =<x,y,z> satisfaisant l'équation implicite F(x,y,z) = 0 et à projeter le point P sur la normale N_i.

On obtient une équation qui peut être résolue par la méthode des multiplicateurs de Lagrange. Pour un objet composé de deux objets, le volume englobant du composé est obtenu en prenant le plus petit des D_1 et le plus grand des D_2.

L'intersection entre le rayon et le volume est calculée comme l'intersection des intervalles obtenus comme résultats de chaque intersection entre le rayon et un slab. Ces intersections rayon-slab sont elles mêmes obtenues en substituant l'équation du rayon dans celles des plans bordant le slab.

Notons que Kajiya a aussi introduit une hiérarchie d'objets en définissant comme volume englobant d'un noeud dans la hiérarchie la combinaison des volumes englobantsde tous les descendants de ce noeud. Avec une telle approche, il est possible d'écarter d'emblée tout le sous-arbre d'un noeud dont le volume englobant n'a pas d'intersection avec le rayon.

Une alternative consiste à décomposer l'espace en une ensemble de volumes disjoints contenant une liste des objets inclus dans ce volume. Le nombre total d'objets dans chaque volume doit être gardé petit, ce qui implique une subdivision fine. La recherche d'une intersection entre un objet et le rayon se fait alors en parcourant la subdivision le long du rayon. Deux approches sont intéressantes: la décomposition spatiale selon un octree et la méthode d'analyse différentielle.

3.4.4.3 La décomposition de l'espace à l'aide d'un octree

Glassner (1984) a décrit un algorithme utilisant la structure d'octree (voir Section 2.3.4), qui constitue un très bon cadre pour subdiviser l'espace en compartiments. Une boîte est d'abord choisie pour englober l'environnement. Puis cette boite est subdivisée en 8 volumes (voxels) selon le principe de l'octree. A chaque fois, les objets situés dans le voxel sont placés dans une liste associée. Pour cela, on teste s'il y a une intersection de l'objet avec chacun des 6 plans bordant le voxel. S'il y en a une, l'objet est mémorisé pour ce voxel, sinon on regarde simplement pour un point de l'objet, s'il est à l'intérieur ou l'extérieur du voxel; dans le premier cas, on garde l'objet, dans le second on l'ignore.

L'algorithme d'intersection du rayon avec les objets stockés dans la structure d'octree peut s'écrire ainsi:

```
Q := l'origine du rayon R0
répéter
    trouver le voxel contenant Q
    pour chaque objet associé avec le voxel
        trouver l'intersection du rayon avec l'objet
    s'il n'y a aucune intersection alors
        Q := un point dans le prochain voxel traversé par le rayon
jusqu'à ce qu'une intersection est trouvée ou que Q est en dehors de la boite englobante de l'environnement
```

L'algorithme de déplacement d'un voxel au prochain se fait en suivant le rayon et en trouvant un point garanti d'être dans le prochain voxel. Pour résoudre ce problème, on détermine le point de sortie P_s du rayon du voxel courant (on obtient P_s en trouvant la plus grande valeur t_m du paramètre t de l'équation du rayon telle que le point obtenu soit dans le voxel). Il s'agit ensuite de s'assurer qu'on ne va pas trop loin (en dehors du prochain voxel); pour cela, on prend la longueur L du côté du plus petit voxel. Le point est alors trouvé en se déplaçant de la distance $\frac{L}{2}$ à partir du point P_s. Le problème avec une telle approche est que la détermination de t_m requiert des calculs d'intersections du rayon avec les six plans qui bordent le voxel.

La Figure 3.4-14 nous donne un exemple 2D. Le rayon primaire parcoure les voxels de l'octree, testant tout d'abord l'objet 2, puis l'objet 1, puis les objets 2, 7 et 8. Il n'y a pas besoin de test supplémentaire, car on est certain que l'objet 7 correspond à l'intersection la plus proche.

3.4.4.4 La méthode d'analyse différentielle

Dans cette méthode introduite par Fujimoto et al. (1986), l'espace est subdivisé en cellules volumiques de même taille (voxels), sur le même principe que l'énumération spatiale vue à la Section 2.3.3. Le volume le long du tracé du rayon est trouvé par un algorithme incrémental, correspondant à une extension DDA (Digital Difference Analyzer) de l'algorithme 2D de tracé d'une droite sur un écran raster. La discrétisation de la droite partant du point $P_1=<x_1,y_1>$ pour aller au point $P_2=<x_2,y_2>$ se fait en résolvant l'équation différentielle:

$$\frac{dy}{dx} = c \qquad \text{ou} \qquad \frac{?y}{?x} = \frac{y_2 - y_1}{x_2 - x_1}$$

dont la solution est:

$$y_{i+1} = y_i + ?y = y_i + ?x \frac{y_2 - y_1}{x_2 - x_1}$$

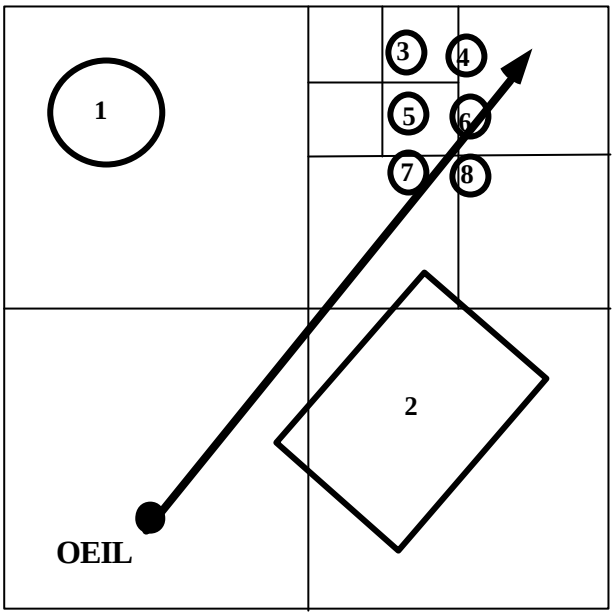


Figure 3.4-14. Subdivision spatiale par octree

Dans un tel algorithme, on distingue deux axes, l'axe dominant qui correspond à la direction de propagation et l'axe passif qui correspond à la direction perpendiculaire.

L'algorithme de génération de la droite peut être vu comme un outil très efficace d'identification des pixels traversés par la droite à discrétiser en les considérant tous (voir Figure 3.4-15). L'extension en 3D (appelée 3DDA) se fait en utilisant deux DDAs synchronisés et travaillant dans des plans perpendiculaires s'intersectant selon l'axe dominant. L'algorithme s'applique en parcourant le rayon et en identifiant directement les trois indices (numéros selon chaque direction de l'espace) de chaque voxel traversé. Les calculs sont donc réduit au minimum.

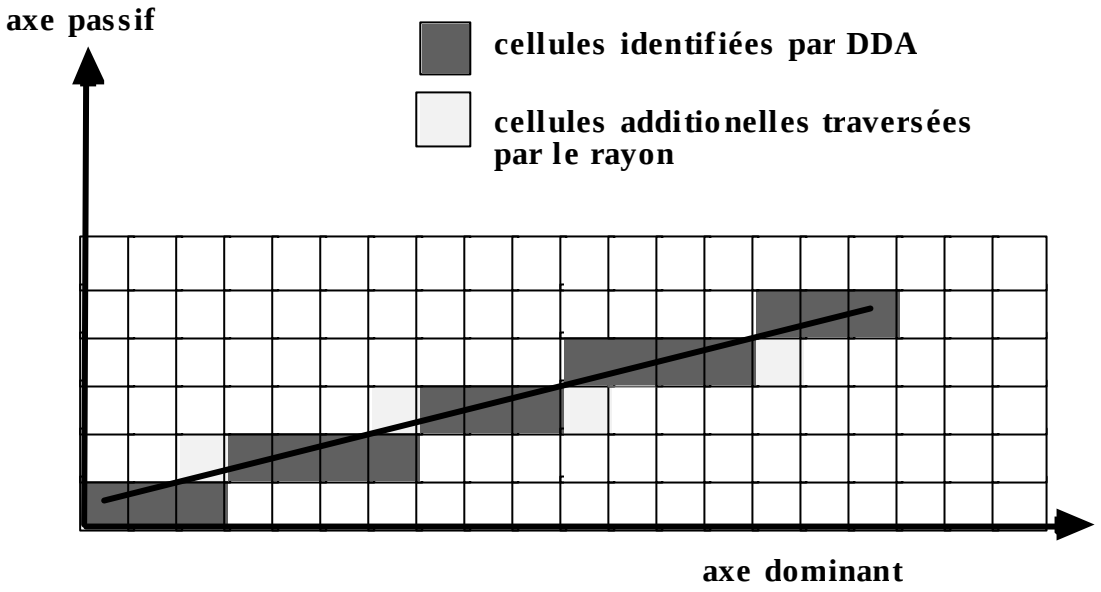


Figure 3.4-15. Principe de l'algorithme DDA

La procédure d'intersection entre un rayon et une collection d'objets organisée en subdivision uniforme en voxels peut s'écrire:

```
calculer i, j, k pour le voxel contenant l'origine du rayon
initialiser l'algorithme 3DDA sur la base de la direction et d el'origine du rayon
repeter
    pour chaque objet associé avec le voxel V[i,j,k]
        calculer l'intersection de l'objet et du rayon
    si aucune intersection n'est trouvée alors
        utiliser l'algorithme 3DDDA pour déterminer de nouvelles valeurs i,j,k
jusqu'à ce que l'intersection est trouvée ou i,j,k sont hors limites (en dehors du tableau de voxels)
```

La Figure 3.4-16 nous donne un exemple 2D. Le rayon primaire parcoure les voxels de l'octree, testant tout d'abord l'objet 2, puis l'objet 1, puis les objets 2, 7 et 8. Il n'y a pas besoin de test supplémentaire, car on est certain que l'objet 7 correspond à l'intersection la plus proche.

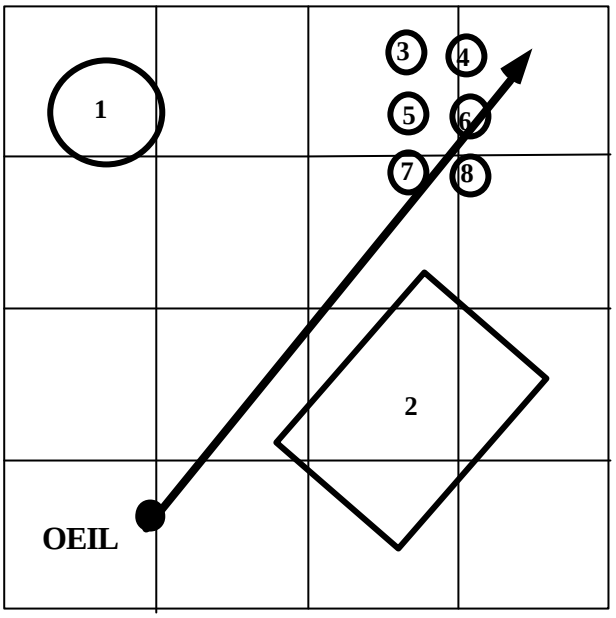


Figure 3.4-16. Enumération spatiale

3.5 La radiosité

3.5.1 Le concept de radiosité

La méthode est très différente du lancer de rayons car elle est basée sur des considérations d'équilibre énergétique et elle permet de modéliser les interréflexions entre surfaces diffuses. La radiosité a été introduite à Cornell University (Cohen and Greenberg 1985-1986). Il faut noter que la méthode est très coûteuse en termes de calcul, elle a cependant l'avantage d'être indépendante de la position de l'observateur. On peut donc calculer une illumination à l'avance et ensuite modifier la position de l'observateur, ce qui permet par exemple de créer des environnements virtuels avec une telle méthode de s'y promener avec un casque de réalité virtuelle (voir Partie 5).

Pour exprimer nos équations de radiosité, nous allons faire l'hypothèse que les processus d'émission et de réflexion sont basés sur des réflecteurs parfaits, donc la direction des rayons est perdue après toute réflexion. On commencera par définir les termes suivants:

radiosité (B): c'est la quantité de base qu'on cherche à calculer pour chaque surface; c'est une énergie par unité de surface et de temps.

réflectivité ρ: c'est la fraction de lumière réfléchie sur une surface (c'est un nombre entre 0 et 1 sans unités); on peut noter que l'absorption vaut 1- ρ

facteur de forme (F): c'est la fraction de la lumière quittant une surface pour arriver à l'autre (c'est une valeur entre 0 et 1 et sans unités)

Emission (E): c'est l'énergie émise par la surface, comme pour une source de lumière, elle s'exprime par unité de surface et de temps.

Nous pouvons maintenant calculer la radiosité d'un élément de surface dA_i (voir Figure 3.5-1) L'intensité de cet élément va dépendre de toute lumière qu'il émet directement plus la lumière qui est réfléchie. Cette lumière réfléchie dépend de la lumière quittant toute autre surface dans l'environnement. Une fraction de la lumière quittant toute autre surface peut arriver à l'élément de surface en question et être réfléchie à son tour dans l'environnement. La fraction dépend du facteur de forme entre les surfaces et de la réflectivité de l'élément de surface.

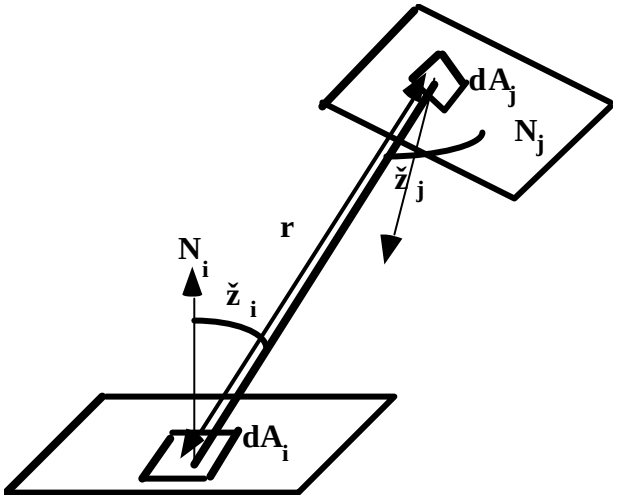


Figure 3.5-1. Interprétation géométrique

Ceci nous amène à l'équation suivante:

$$B_{dA_i} dA_i = E_{dA_i} + \rho_{dA_i} \int B_{dA_j} F_{dA_j - dA_i} dA_j$$

où:

B_{dA_i} est la radiosité de l'élément de surface dA_i

E_{dA_i} est l'émission de l'élément de surface dA_i

ρ_{dA_i} est la réflectivité de l'élément de surface dA_i

$F_{dA_j - dA_i}$ est le facteur de forme de dA_j à dA_i, donc la fraction d'énergie quittant dA_j pour arriver en dA_i

Pour résoudre une telle équation, il faut discrétiser l'environnement en le subdivisant en régions discrètes pour lesquelles on assume que la radiosité et l'émission sont constantes dans la région. On aura donc la nouvelle équation:

$$B_{A_i} A_i = E_{A_i} A_i + \rho_{A_i} \sum_j B_{A_j} F_{A_j - A_i} A_j$$

En considérant que la fraction d'énergie émise par un élément et reçue par un autre est identique à la fraction d'énergie émise dans l'autre sens, on peut exprimer les facteurs de forme entre les surfaces i et j simplement comme le rapport des surfaces:

$$F_{A_i - A_j} = F_{A_j - A_i} \frac{A_j}{A_i}$$

On en déduit l'équation fondamentale pour calculer la radiosité:

$$B_{A_i} = E_{A_i} + \rho_{A_i} \sum_j B_{A_j} F_{A_i - A_j}$$

Si l'environnement est divisé en N régions, il en résultera un système linéaire de N équations qu'on peut mettre sous forme matricielle et résoudre avec la méthode de Gauss-Seidel.

3.5.2 La méthode de l'hémicube pour calculer les facteurs de forme

Cohen and Greenberg (1985) ont décrit une méthode pour approximer les facteurs de forme. Cette méthode est basée sur le fait qu'on peut observer que (voir Figure 3.5-2) lorsqu'on projette 2 régions de l'environnement sur n'importe quelle surface et qu'elles occupent la même zone et emplacement, alors elles ont le même facteur de forme.

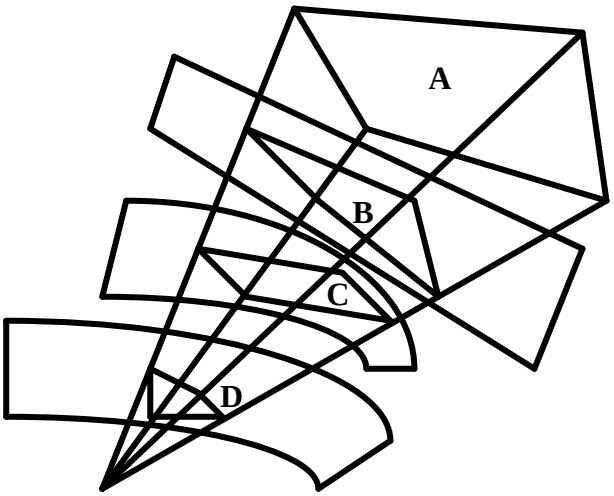


Figure 3.5-2. Aires avec les mêmes facteurs de forme

Par exemple, chaque région est projetée sur un cube imaginaire situé au centre de la région de façon à calculer les surfaces cachées. Seule la moitié du cube (**hémicube**) au-dessus de la région est de l'environnement. La contribution de chaque pixel de la surface du cube au facteur de forme varie et dépend de la position et de l'orientation du pixel. Un élément de facteur de forme $?F_q$ est calculé pour chaque pixel q sur l'hémicube. Par exemple, considérons le sommet de l'hémicube montré à la Figure 3.5-3.

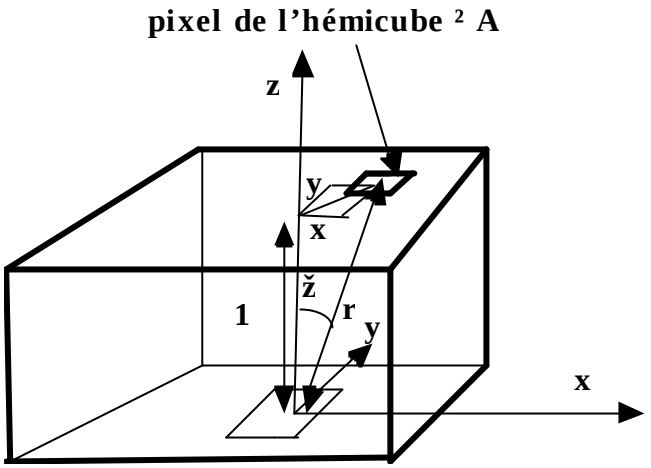


Figure 3.5-3. Sommet de l'hémicube

Comme r = $\sqrt{x^2 + y^2 + 1}$ et cos ? $_i$ = cos ? $_j$ = $\frac{1}{r}$, l'élément de facteur de forme est donné par:

$$?F_q = \frac{\cos\Omega_i \cos\Omega_j}{\pi r^2} \; ?A = \frac{\Delta A}{(x^2 + y^2 + 1)^2}$$

Le processus est répété pour chaque région de l'environnement en positionnant l'hémicube au centre de la région. Finalement, le facteur de forme est approximé par:

$$F_{A_i - A_j} = \sum_{q=1}^{N} \Delta F_q$$

où N est le nombre de pixels de l'hémicube couverts par la projection de la région sur l'hémicube.

Les facteurs de forme sont stockés dans une matrice, qui après simplification est multipliée par les réflectivités pour chaque composante de couleur (Rouge, Vert, Bleu) et résolue en utilisant la méthode itérative de Gauss-Seidel afin d'obtenir les radiosités pour chaque région.

3.6 La texture

3.6.1 Définition de la texture

Les objets synthétisés par ordinateur ont très souvent une apparence artificielle; il est en effet difficile d'empêcher cet aspect lisse et "plastique" des objets. Un exemple typique est la représentation de la peau humaine qui paraît toujours trop parfaite et donne aux humains synthétisés une allure de mannequin de cire ou de marionnette de bois. Pour réduire cet aspect lisse, on peut utiliser des méthodes de texture. Une texture est définie comme une microstructure de la surface à représenter et différentes techniques sont possibles.

1. Texture par application d'une image ("texture mapping")

2. Texture par perturbation de la lumière.

3. Texture solide

3.6.2 Application d'une image bidimensionnelle

C'est la méthode la plus simple et la plus courante, elle peut être appliquée en temps réel sur des stations graphiques comme les Silicon Graphics Onyx ou Impact.

Catmull (1975) a montré que des photographies ou n'importe quel dessin peut être appliqué sur une surface libre. Cependant cette technique ne fonctionne pas bien quand le nombre de points de la surface à afficher est inférieur au nombre d'éléments dans l'image à appliquer. Catmull suggère alors de faire des correspondances région par région plutôt que point par point, en subdivisant simultanément la surface et l'image. En fait, cette application d'image revient à une transformation entre un système de coordonnée (espace de texture) et un autre système de coordonnées (espace tridimensionnel classique). Si les deux espaces sont exprimés en coordonnées paramétriques, on peut définir la fonction de correspondance:

espace de texture: (t,u) espace 3D : (v,w)

v=F(t,u) et w=G(t,u) ou t=H(v,w) et u=I(v,w)

Un cas commun est l'application d'images polygonales bidimensionnelles sur des surfaces polygonales 3D définies avec m x n faces quadrilatérales. La technique revient à découper l'image plane selon une grille puis à faire correspondre à chaque case son équivalent sur la surface. On peut alors transformer la partie d'image contenue dans la case plane selon la forme du polygone équivalent.

3.6.3 Texture par perturbation de la lumière

Nous avons vu que la lumière en chaque point d'un objet est dépendante de l'angle que forme l'objet en ce point avec la direction de la lumière; plus précisément le calcul de la lumière en un point dépend de la normale (perpendiculaire) en ce point. En modifiant judicieusement cette normale selon une technique due à Blinn (1978), on peut créer des zones de lumière et d'ombre qui simulent une texture. Pour modifier la normale, Blinn utilise une fonction mathématique F(u,v) qui mesure le déplacement de la surface irrégulière par rapport à la surface idéale. Un nouveau point P' sur la surface perturbée est obtenu par l'équation suivante:

$$P' = P + FU$$

où P est le point original et U un vecteur normal unitaire. La Figure 3.6-1 montre une section de la surface originale (v est supposé constant), la fonction F(u) et la section de la surface perturbée correspondante.

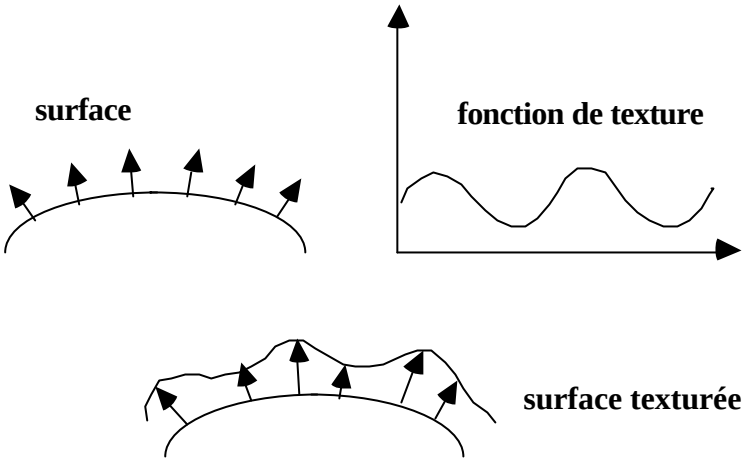


Figure 3.6-1. Principe de la texture par perturbation de la normale

Comment calculer cette nouvelle normale ?

Soit P=<x,y,z>, x=x(u,v), y=y(u,v), et z=z(u,v), le vecteur normal à la surface en P est donné par:

$$P'_u = \frac{\partial P}{\partial u} = < \frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u} >$$

$$P'_v = \frac{\partial P}{\partial v} = < \frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v} >$$

où P'_u et P'_v définissent un plan tangent à la surface en P.

On peut alors calculer le produit vectoriel qui nous donne le vecteur normal à la surface:

$$N = P'_u \times P'_v$$

d'où:

$$Q = P + F \frac{N}{|N|} = P + F\, n$$

La nouvelle normale M est obtenue comme:

$$M = Q'_u \times Q'_v = (P'_u + F'_u\, n + F\, n'_u) \times (P'_v + F'_v\, n + F\, n'_v)$$

En considérant F comme négligeable, on obtient

$$M = P'_u \times P'_v + F'_u\, (n \times P'_v) + F'_v\, (P'_u \times n) + F'_u\, F'_v\, (n \times n)$$

Le dernier terme vaut 0 et le premier terme est N, on a alors:

$$M = N + D$$

avec $$D = F'_u\, (n \times P'_v) - F'_v\, (n \times P'_u)$$

Interprétation géométrique

Une partie des 2 vecteurs ($N \times P'_u$, $N \times P'_v$) dans le plan tangent à la surface est additionnée à la normale originale N. Cette partie est proportionnelle aux dérivées de F par rapport à u et v (voir Figure 3.6-2).

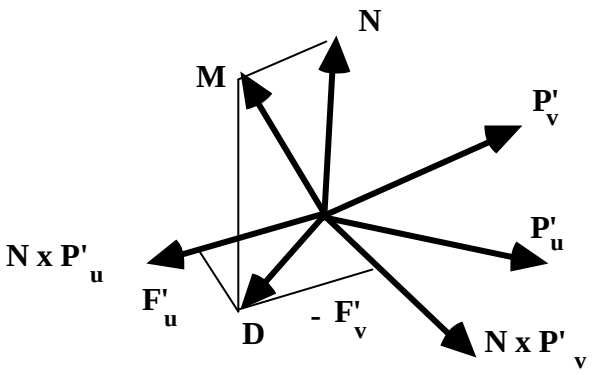


Figure 3.6-2. Interprétation géométrique

Le vecteur M peut être aussi vu comme provenant de la rotation du vecteur original N autour d'un axe dans le plan tangent à la surface (voir Figure 3.6-3).

Le vecteur directeur de l'axe est obtenu par le calcul du produit vectoriel:

$$N \times M = (F'_u (N \times (n \times P'_v)) - F'_v (N \times (n \times P'_u))$$

$$= |N| (F'_v \ P'_u - F'_u \ P'_v) = |N| A$$

A est un vecteur perpendiculaire au vecteur $< F'_u, F'_v >$.

L'angle de rotation est: $q = \arctan \frac{|D|}{|N|}$

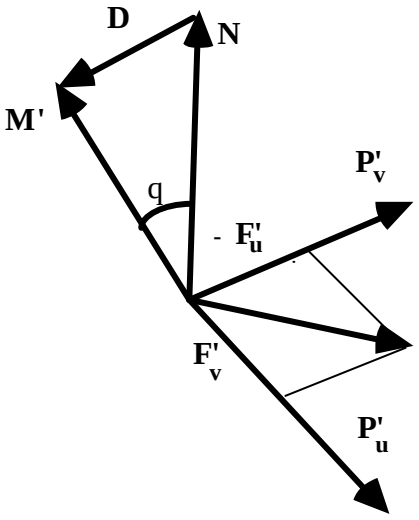


Figure 3.6-3. Interprétation rotationnelle

Le choix de la fonction F est très important. Blinn a proposé différentes techniques:

- F est défini analytiquement comme un polynôme a deux variables ou une série de Fourier; cette approche est très coûteuse.
- F est défini par une table à deux entrées; les résultats ne sont pas excellents et cela nécessite beaucoup de mémoire.
- F est défini par une table à deux entrées plus petite et une interpolation (par B-spline, par exemple) est opérée pour trouver les valeurs intermédiaires.

3.6.4 Texture solide

La texture solide proposée par différents auteurs (Perlin 1985; Peachey 1985) a l'avantage de pouvoir s'appliquer à des objets modélisés arbitrairement (surfaces libres, facettes). C'est une généralisation de la méthode de Blinn avec des fonctions tridimensionnelles F(x,y,z). A chaque point de la surface correspond un point dans un espace de texture; on peut d'ailleurs aussi faire correspondre un point dans un espace de couleurs ou de transparence. La méthode assure la continuité de la texture sur tout l'objet puisqu'elle est indépendante de la forme de l'objet. Pour appliquer de tels espaces 3D, il faut appliquer des transformations inverses aux coordonnées des objets.

N'importe quelle fonction T(x,y,z) peut être théoriquement appliquée comme fonction de texture. Cependant, dans la pratique le choix de la fonction est assez délicat. Par exemple, la fonction suivante permet de simuler des motifs réguliers avec une faible perturbation aléatoire:

$$T(x,y,z)=Fact*\sin[2p\ (Ax + B/10[C*random(x,y,z)+1+\cos(2p\ Dy)])]$$

La Figure 3.6-4 montre des exemples de textures solides appliquées à un vase.

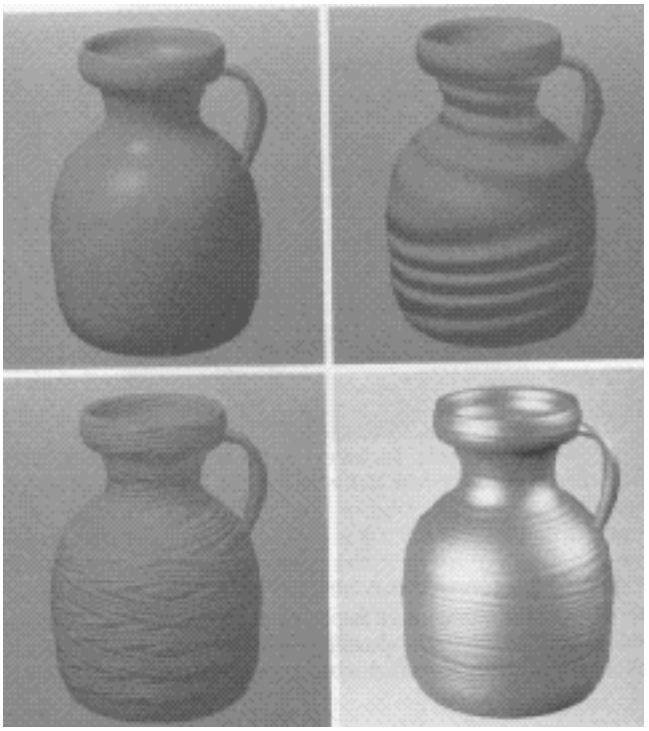


Figure 3.6-4. Exemples de textures solides

3.7 La simulation des objets et des phénomènes naturels

3.7.1 La représentation de la nature

Il n'est pas surprenant que les objets les plus faciles à construire par ordinateur soient justement ceux que l'humain a inventé: les maisons, les voitures, les avions. Par contre, les objets de la nature et les phénomènes naturels en général ont donné des casse-têtes aux chercheurs. Les objets rigides (à facettes ou à surfaces libres) ne permettent généralement pas de modéliser les phénomènes naturels. Il faut plutôt utiliser des objets de nature floue. Des objets flous, tels que définis par Reeves (1983, 1985) sont des objets qui ne sont pas délimités par une surface bien définie, bien nette. Leur forme est irrégulière, mal définie et change au cours du temps. Les objets flous que l'on cherche le plus souvent à représenter sont les nuages, le feu, la fumée, l'eau. Il est toujours possible d'utiliser des formes géométriques pour simuler ces objets flous. Ainsi, par exemple, Schachter (1983) a décrit la simulation de nuages (cumulus) et de fumée à l'aide d'une concaténation d'ellipsoïdes. Les résultats sont très loin de la réalité mais la technique a l'avantage de ne requérir que très peu de temps d'ordinateur, d'où son usage dans le cadre des simulateurs de vols aériens.

L'équipe de Csuri (1979) a proposé un modèle pour représenter un nuage de fumée. Le nuage est tout d'abord généré par une approximation mathématique. Ensuite, une technique de lancer de rayons permet de construire un tableau des intensités de lumière. La technique décrite par Marshall et al. (1980) est une des premières à introduire le concept de particules repris par Reeves. Blinn (1982) a produit des images de l'anneau de Saturne en utilisant des fonctions de réflexion de lumière permettant de simuler les nuages et les surfaces poussiéreuses. La technique consiste à simuler la lumière transmise et réfléchie par des couches de particules. Nelson Max (1981) a modélisé des vagues de l'océan dans son film "Carla's Island" en utilisant des collections de fronts d'ondes linéaires

basés sur les équations à dérivées partielles. James Kajiya (1983) a défini un modèle de formation de nuages sur des densités de volume. Mais les nuages les plus impressionnants sont certainement ceux de Gardner (1984) basés sur des fonctions de texture appliquées à des ellipsoïdes. Finalement, les modèles de feu et de vagues de Ken Perlin (1985) sont basés sur une fonction linéaire du temps appliquée à une fonction de texture aléatoire. Dans les sections suivantes, on verra tout d'abord les systèmes de particules qui permettent de représenter le feu, la fumée, les feux d'artifice, puis on verra comment créer des arbres, enfin on montrera comment représenter des cheveux et de la fourrure.

3.7.2 Les systèmes de particules

3.7.2.1 Principe des systèmes de particules

La technique de base a été mise au point par Reeves (1983). Il s'agit de faire évoluer des particules dans un système donné. Les particules naissent, meurent, se déplacent, changent de couleur, de taille. Reeves décrit comment calculer chaque image selon les 4 étapes suivantes:

1. Générer de nouvelles particules au moyen de processus stochastiques et leur assigner des attributs.

2. Supprimer les particules dont le temps de vie est dépassé.

3. Déplacer et transformer les particules restantes sur la base de leurs attributs dynamiques.

4. Appliquer des algorithmes de traitement réaliste de l'affichage des particules.

Le nombre de particules Np_f générées à une image donnée peut être déterminé par une équation inpliquant soit le nombre moyen de particules MP_f générées à cette image et sa variance VP_f:

$$NP_f = MP_f + RANDOM \cdot VP_f$$

où le nombre moyen généré par unité d'écran MP_{sf} et sa variance VP_{sf} par unité d'écran:

$$NP_{sf} = (MP_{sf} + RANDOM \cdot VP_{sf}) \times S$$

où RANDOM est une fonction qui retourne un nombre aléatoire entre -1 et 1 et S est la surface de l'écran.

Le nombre moyen de particules générées par image MP_f peut varier au cours du temps comme une fonction du numéro d'image f:

$$MP_f = MP_{f0} + ?MP \cdot (f-f_0)$$

Les particules sont générées selon une certaine forme du système. Cette forme sera différente si on veut générer une boule de feu ou un feu de cheminée. La Figure 3.7-1 nous montre trois types de forme de système. Pour un feu de cheminée on choisira certainement le second type, car les flammes devraient partir du sol pour monter.

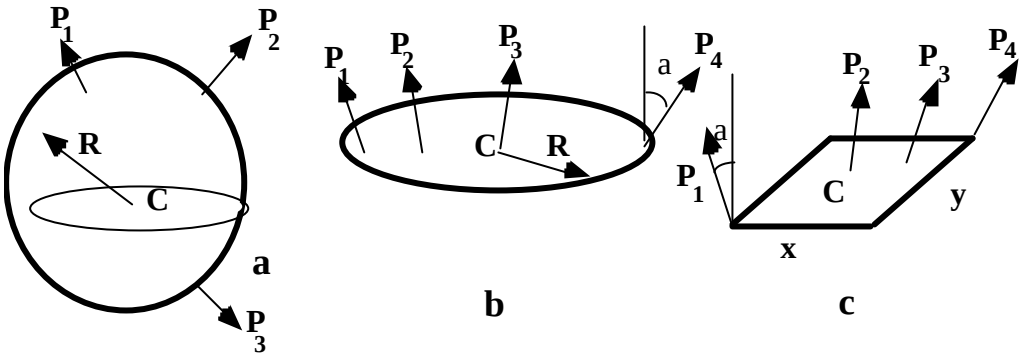


Figure 3.7-1**.** Forme de systèmes de particules. Les P_i sont les particules. **a** Sphère de centre **C** et de rayon R; **b** cercle de centre **C** et de rayon R dans le plan xy; α est l'angle d'éjection; **c** rectangle de centre **C** et d'arêtes xy; α est l'angle d'éjection

3.7.2.2 Attributs des particules et dynamique

Pour chaque nouvelle particule générée, des valeurs sont déterminées pour les attributs suivants: position initiale, vitesse, taille, couleur, transparence, forme (sphère, rectangle) et temps de vie. Les valeurs initiales peuvent aussi varier au cours du temps en donnant une valeur moyenne et un écart-type. Par exemple, la couleur initiale COL est donnée par:

$$COL = M_{COL} + RANDOM \cdot V_{COL}$$

où M_{COL} est la couleur moyenne et V_{COL} la variation maximale.

Des attributs dynamiques sont aussi définis par des taux de changements des attributs initiaux: changement de couleur, de forme, de taille, de vitesse. Ceci permet de contrôler le mouvement et les transformations. Une particule est morte dès que son temps de vie atteint zéro, ce qui s'accomplit en décrémentant le temps de vie courant à chaque image.

Les systèmes de particules ont été utilisés pour produire la séquence "Genesis Demo" dans le film de Lucasfilm Ltd *Star Trek II: The Wrath of Khan.* Des feux d'artifice ont été aussi modélisés en utilisant des systèmes de particules.

3.7.2.3 L'apparence réaliste des particules

L'apparence réaliste des particules pourrait être difficile à rendre à cause des problèmes de transparence et d'ombre portée. Cependant dans le cas d'explosions ou de feu, l'algorithme est simple car chaque particule peut être considérée comme une source de lumière ponctuelle. Chaque pixel va alors "gagner" de l'intensité lorsqu'il est couvert par une particule. Le montant d'intensité dépend d'attributs tels que la couleur et la transparence. La taille et la forme des particules influent bien entendu sur le nombre de pixels recouverts.

La modélisation des nuages par des systèmes de particules est donc beaucoup plus complexe, étant donné que les particules ne peuvent être représentées comme des sources de lumière, mais doivent être considérées comme des objets individuels reflétant la lumière.

Il faut aussi signaler que, dans le travail de Reeves, les systèmes de particules ne peuvent être mélangées avec d'autres types d'objets, comme des objets basés sur des facettes. En effet, la séquence Genesis Demo a été produite en utilisant un processus de composition d'images. Un

algorithme permettant l'intersection d'objets basés sur des facettes avec des systèmes de particules a été développé par Magnenat-Thalmann et al. (1986). Cet algorithme a été intégré au système MIRANIM, pour permettre un contrôle interactif des systèmes de particules et de leur évolution. L'algorithme de réalisme des particules est basé sur la technique du A-buffer décrite à la Section 3.1.4.3. Les particules sont tout d'abord calculées dans le système de coordonnées de l'oeil, comme pour des polygones. Chaque particule est considérée comme une surface translucide; ce qui signifie que l'on traite la particule comme un polygone transparent. Dans la liste des pixels transparents, nous trouvons des pixels provenant de polygones et des pixels dus aux particules, triés selon la profondeur. On distingue les particules des polygones par une information spéciale. Avant d'afficher la ligne de balayage, les particules de la ligne courante sont insérées dans une liste de particules actives, puisqu'elles deviennent actives pour la nouvelle ligne. Après le calcul des intensités pour la ligne courante, les particules devenues inactives sont retirées de la liste. La Figure 3.7-2 nous montre un exemple.

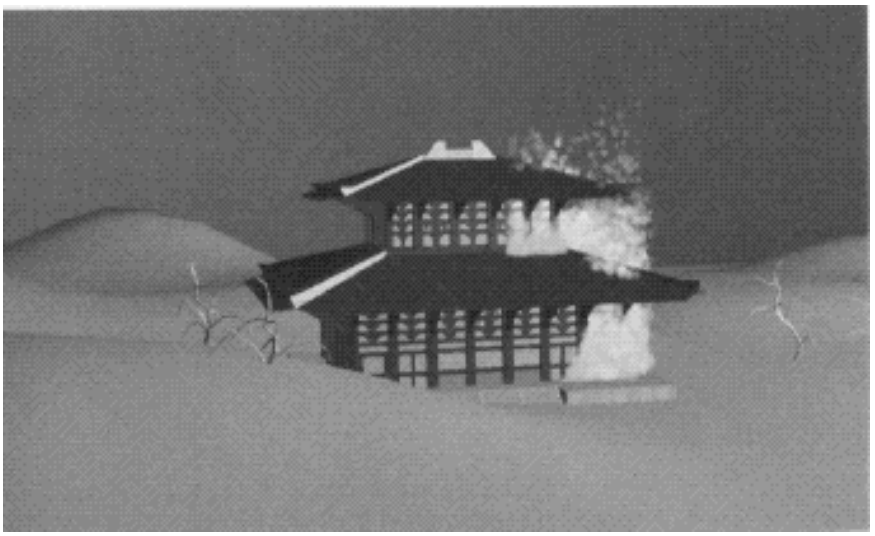


Figure 3.7-2. Pagode en feu avec systèmes de particules

3.7.3 La représentation des arbres

3.7.3.1 Arbres fractals

Pour représenter des arbres, on peut utiliser les fractales et définir un initiateur et un générateur. En effet, si on considère qu'un arbre est formé de branches et que celles-ci ne sont que des sous-arbres, semblables à l'arbre lui-même, il est facile de générer un arbre aussi complexe que l'on veut, comme le montre la Figure 3.7-3.

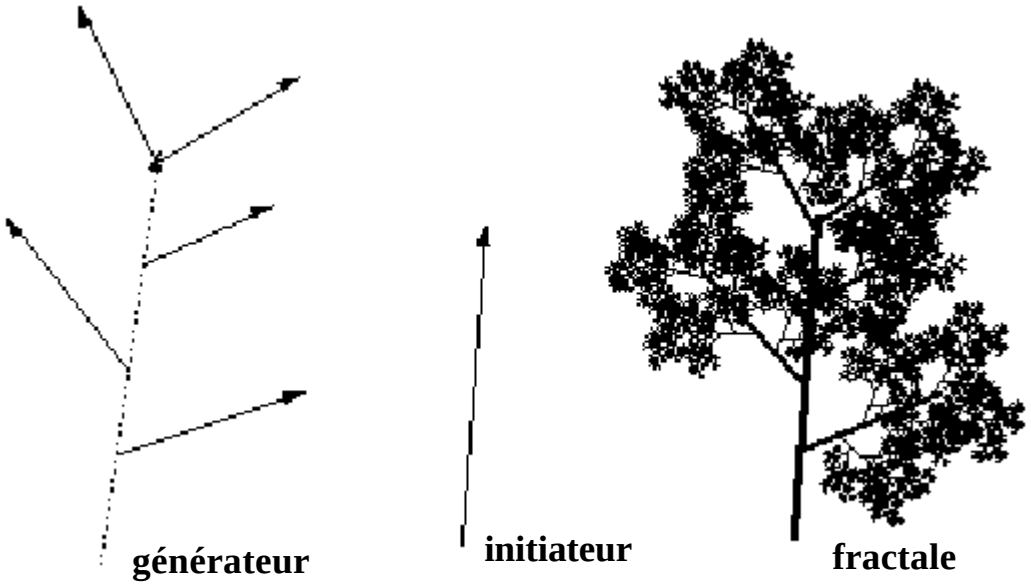


Figure 3.7-3. Arbre fractal

3.7.3.2 0L-systèmes arborescents

En fait, l'approche la plus adéquate est la définition d'arbres sous forme de grammaires et plus particulièrement sous forme de L-systèmes, comme on a vu à la Section 2.6. Pour représenter des structures hiérarchiques, on va définir des L-systèmes indépendant du contexte (0L-systèmes) dans lesquels le mécanisme de réécriture s'applique directement à des arbres axiaux.

Un arbre axial est un arbre avec racine tel qu'à chaque noeud on distingue au plus un segment droit qui part. Tous les autres arcs sont appelés des segments latéraux. Une suite de segments est considérée comme un axe si:

- le premier segment part de la racine de l'arbre ou est un segment latéral en un noeud
- chaque segment suivant est un segment droit
- le dernier segment n'est suivi par aucun segment droit dans l'arbre

Un axe avec tous ses descendants constitue une branche qui est elle-même un sous-arbre axial.

Un 0L-système arborescent G est défini par trois composantes:

- un ensemble d'arcs étiquetés V
- un arbre initial ω avec étiquettes dans V
- un ensemble de productions P

Un arbre axial T_2 est directement dérivé d'un arbre T_1 (noté $T_1 \Rightarrow T_2$) si T_2 s'obtient à partir de T_1 en remplaçant simultanément chaque arc de T_1 par son successeur selon les productions P. Un arbre T est généré par G en une dérivation de longueur n s'il existe une séquence d'arbres T_0, T_1, ... T_n telle que $T_0 = \omega$, $T_n = T$ et $T_0 \Rightarrow T_1 \Rightarrow \ldots \Rightarrow T_n$.

La Figure 3.7-4 nous montre une production p d'arbre et le mécanisme de remplacement d'un arc par son successeur selon la production p.

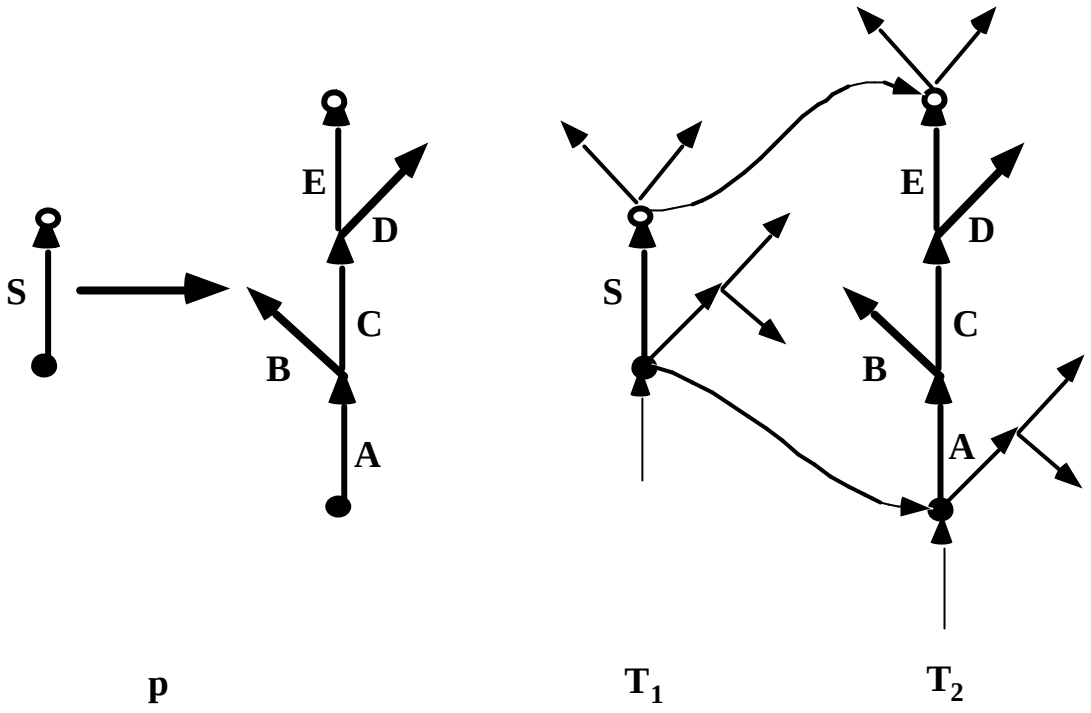


Figure 3.7-4. Production p d'arbre et mécanisme de remplacement

La Figure 3.7-5 donne un exemple très complet d'arbre 3D produit avec un L-système.

Figure 3.7-5. Exemple d'arbre produit avec les productions: w: A
p1: A → [&FL! A] /////'[&FL!A]///////'[&FL!A]
p2: F → S ///// F p3: S → FL

p4: L → [‘’’^^{-f+f+f-|-f+f+f}]

3.7.3.3 L-systèmes dépendant du temps

Pour représenter l'évolution dans le temps (p.e. croissance des plantes ou animation), on doit introduire les tD0L-systèmes qui dépendent du temps. Un tD0L-système est un triplet G = (V, ω, P) où V est l'alphabet, ω est le mot initial et P l'ensemble fini des productions avec $\omega \in (V \times R)^+$, $P \subset (V \times R) \times (V \times R)^*$ et R l'ensemble des nombres réels positifs.

Lorsqu'on écrit la production $(a, \beta) \rightarrow (b_1, \alpha_1) \ldots (b_n, \alpha_n)$, le paramètre β représente l'âge terminal de la lettre a et chacun des paramètre α_i est l'âge initial assigné à la lettre b_i par production P. On a évidemment que le temps de vie $\beta - \alpha$ doit être positif.

On définira alors la fonction de dérivation comme:

$$D: (((V \times R)^+ \times R) \rightarrow (V \times R)^*$$

avec les axiomes suivants:

1) $D(((a_1, \tau_1) \ldots (a_n, \tau_n)), t) = D((a_1, \tau_1), t) \ldots D((a_n, \tau_n), t)$ où t est le temps global qui synchronise le développement entier et τ_i sont les valeurs des âges locaux spécifiques aux lettres a_i. L'axiome spécifie l'indépendance de toutes les lettres.

2) $D((a, \tau), t) = (a, \tau+t)$ si $\tau+t = \beta$, cela signifie que chaque lettre vieillit jusqu'à ce qu'elle atteigne son âge terminal.

3) $D((a, \tau), t) = D((b_1, \alpha_1)\ldots)(b_n, \alpha_n), t-(\beta-\tau))$ si $\tau+t > \beta$, cela signifie que lorsque l'âge terminal de la lettre est atteint, une subdivision a lieu et de nouvelles lettres sont produites avec des valeurs d'âge initial spécifiées par la production correspondante.

A titre d'exemple, nous pouvons reprendre l'exemple biologique du paragraphe 2.6.2 et introduire une durée de vie de 0 à 1, on aura:

ω: $(a_r, 0)$
p_1: $(a_r, 1) \rightarrow (a_l, 0)(b_r, 0)$
p_1: $(a_l, 1) \rightarrow (b_l, 0)(a_r, 0)$
p_1: $(b_r, 1) \rightarrow (a_r, 0)$
p_1: $(b_l, 1) \rightarrow (a_l, 0)$

qui correspond à l'arbre de dérivation de la Figure 3.7-6 où on a utilisé des arcs triangulaires pour représenter le processus de vieillissement. Ainsi à l'âge 2.75, on a trois filaments b_l, a_r et a_r dont l'âge est de 0.75.

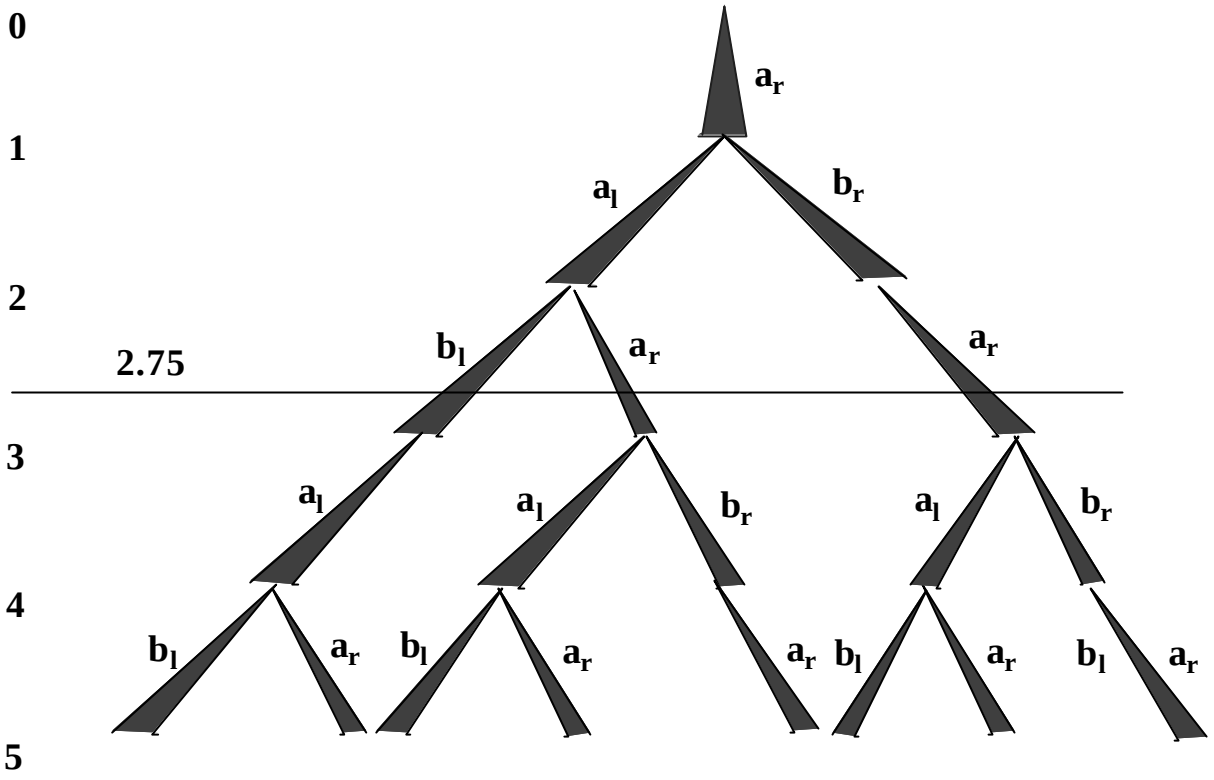


Figure 3.7-6. Arbre de dérivation avec développement continu

3.7.4 La représentation des cheveux et fourrure

3.7.4.1 Le problème des cheveux et quelques solutions proposées

La représentation réaliste des cheveux et de la fourrure en général est un des problèmes les plus complexes à résoudre dans le domaine de la synthèse d'images. Pour simplifier la description, nous considérerons le cas spécifique d'une chevelure. Il faut évidemment considérer que le terme cheveu signifiera pour nous un poil et chevelure sera synonyme de coiffure, de fourrure, barbe etc ... Les difficultés résultent du nombre très grand de cheveux (une chevelure normale a en moyenne 100,000 à 150,000 cheveux), du détail nécessaire pour chaque cheveu individuel, de l'interaction très complexe de la lumière et de l'ombre avec les cheveux et de la finesse des cheveux comparativement à la taille des pixels d'un écran.

Au niveau de la modélisation, le problème est celui de spécifier des centaines de milliers d'éléments individuels: leur géométrie, leur distribution, leur forme, leur direction. Au niveau du rendu, c'est surtout le problème d'aliaising qui constitue une difficulté. En effet, de nombreux cheveux, réfléchissant la lumière et produisant de l'ombre les uns sur les autres, contribuent à la couleur finale de chaque pixel et pour une tête de la taille de l'écran, le diamètre d'un cheveu est plus petit que la largeur d'une pixel.

On peut bien sûr utiliser des méthodes simples pour représenter une chevelure. Par exemple, Csuri et al. (1979) ont modélisé des cheveux en les représentant comme des triangles très minces et en utilisant un algorithme de z-buffer. De meilleures résultats ont été obtenus par Gavin Miller (1988) qui a modélisé les cheveux à l'aide de longues pyramides. Même avec un nombre de cheveux limité, ces méthodes restent coûteuse et le résultat n'est pas très réaliste, car il est impossible de simuler la

finesse des cheveux humains, par contre ces méthodes permettent de créer des fourrures d'animaux ou des champs d'herbes.

Une autre approche consiste à ne pas simuler les cheveux individuellement, mais à les représenter par des densités de volumes ou des textures 3D. Un ours en peluche très réaliste a ainsi été créé par Kajiya and Kay (1989) en utilisant un concept de texel, sorte de modèle intermédiaire entre une texture et une géométrie.

Watanabe and Suenaga (1989) ont modélisé des cheveux humains en connectant des prismes triangulaires et en utilisant un algorithme de z-buffer avec le modèle de Gouraud pour l'illumination. La méthode est rapide et permet de représenter une chevelure complète, mais le problème d'aliaising n'a pas été résolu et les cheveux ne sont pas vraiment assez fins, du moins pour des cheveux occidentaux.

Nous allons décrire une méthode que nous avons introduite (LeBlanc et al. 1991) et qui s'approche un peu de la méthode des systèmes de particules vue à la Section 3.7.2.

3.7.4.2 Modélisation

Pour créer des images réalistes avec des coiffures ou des objets en fourrure, il est tout d'abord nécessaire de définir un fichier de courbes représentant les cheveux individuels avec leur géométrie et l'aspect matériel. Une base de données de cheveux sous forme de segments va être créée et elle sera ensuite utilisée pour le traitement du rendu. Pour cette étape de création, on doit utiliser un modeleur spécialisé comme STYLER (Daldegan et al. 1993) et l'élaboration d'un fichier de caractéristiques suit deux étapes principales:

- la création des courbes de cheveux ou de poils
- la définition de la coiffure

La création de courbes ne présente évidemment aucune difficulté. Pour créer un objet avec une chevelure ou une fourrure, le programme STYLER va partir d'une structure polygonale représentant l'objet et affecter à chaque polygone de la surface l'information du type de cheveu, de la courbe de base ou du segment de cheveu, son orientation et certains paramètres stochastiques.

Les courbes de cheveux vont être distribuées sur la surface polygonale (des triangles en fait), conformément à la coiffure choisie par le concepteur. Il va être possible de modifier un certain nombre de paramètres (par rapport à une valeur standard), p.e. la couleur, la taille (longueur moyenne), l'orientation des cheveux, leur déviation par rapport à une direction donnée, la densité.

Une fois qu'on a défini le format de cheveux et appliqué à un triangle, la coiffure complète peut être spécifiée en indiquant la densité de cheveux sur la surface. Au départ les cheveux ont tous la même longueur et la même orientation. Pour rendre les coiffures plus naturelles, tous les paramètres peuvent être modifiés interactivement. Une longueur, orientation et position aléatoires peuvent être assignées à chaque triangle. Une collection de coiffures différentes peut être générée à partir du même ensemble de cheveux construits avec des cylindres courbés.

3.7.4.3 Le rendu des cheveux

Le rendu des cheveux est fait en modifiant un algorithme de rendu, par exemple de lancer de rayons. Un algorithme de shadow buffer (voir Section 3.3.4) a été ainsi ajouté. Une telle méthode fonctionne

très bien pour calculer les ombres des objets en général et aussi pour créer l'ombre portée des cheveux.

Le processus de rendu se fait en plusieurs étapes: l'ombre de la scène est calculée pour chaque source de lumière *i*, de même que les ombres des cheveux pour chacune des sources. Les ombres des cheveux sont calculées pour la surface des objets et pour chaque cheveu individuel. Finalement, la coiffure elle-même est fondue dans la scène, en utilisant tous les shadow buffers. Le résultat est une image avec une coiffure réaliste avec effets de lumière et d'ombre. La Figure 3.7-7 nous montre le principe.

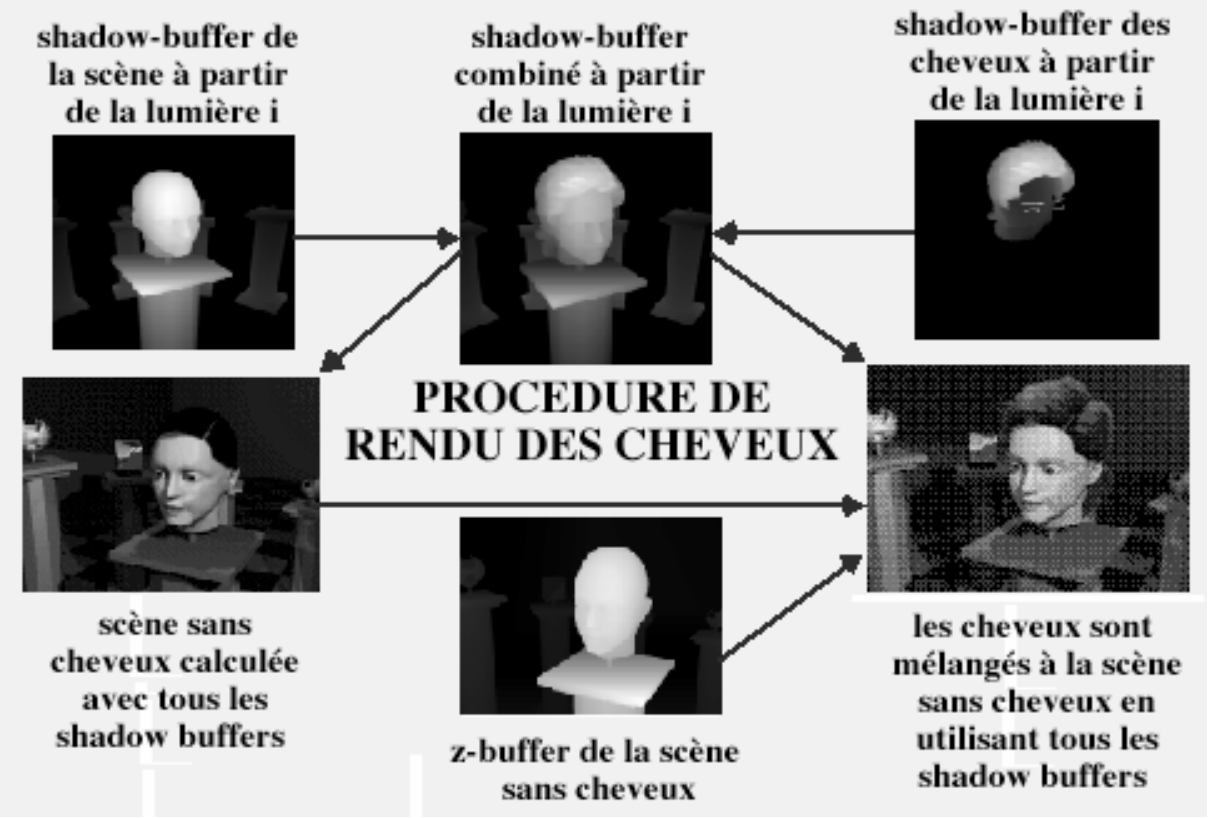


Figure 3.7-7.Principe de rendu des cheveux

De manière plus détaillée, le processus complet (pipeline) de rendu peut se décrire ainsi:

- Nous prenons le modèle de description de la scène et le projetons sur chaque source de lumière, créant ainsi un shadow buffer de la scène pour chaque source de lumière.

- Nous prenons le modèle de cheveux et le projetons sur chaque source de lumière, créant ainsi un shadow buffer des cheveux pour chaque source de lumière. Ceci est effectué en dessinant chaque segment de cheveux dans le frame buffer associé au z-buffer et en extrayant la répartition finale des profondeurs (depth map).

- Nous composons ensuite les depth maps pour le shadow buffer de la scène et le shadow buffer des cheveux, ce qui nous donne un unique shadow buffer composé pour chaque source de lumière.

- Nous générons l'image de la scène et son z-buffer, en utilisant le modèle de description de la scène et les shadow buffers composés comme données d'entrée pour le programme de rendu de la scène. Il en résulte une scène complètement rendue avec les ombres des cheveux mais pas les cheveux.

- Nous fondons les segments de cheveux dans l'image de la scène, en utilisant le z-buffer de la scène pour déterminer la visibilité et les shadow buffers composés pour déterminer les ombres, résultant en l'image finale avec cheveux et ombres complètes. Pour ce processus de fusion, chaque cheveu est décomposé en segments 3D de droite. L'intensité H de chaque extrémité du segment est determinée en utilisant l'équation suivante:

$$H = L_A K_A + \Sigma S_i L_i (KD\sin(\theta) + KS \cos^n(\phi + \theta - \pi))$$

  où L_i est l'intensité de la lumière, K_A est le coefficient de réflectance ambiant et L_A est la puissance de lumière ambiante incidente par unité de surface. S_i est un coefficient d'ombrage que l'on obtient en filtrant le pixel à la valeur de profondeur du cheveu par rapport au shadow buffer de la i-ième source de lumière.

Le programme de lancer de rayons a deux rôles de base dans le processus de rendu. Le premier est de construire les shadow buffers pour chaque source de lumière, et le second est de calculer le rendu des objets sans cheveux mais avec les ombres complètes (venant des cheveux et des objets sans fourrure).

Il y a en fait deux moyens de créer des shadow buffers. La première méthode est de commencer par calculer le rendu des objets avec l'algorithme de z-buffer, puis de relire le z-buffer. La seconde est d'utiliser l'algorithme de lancer de rayons et de calculer le rendu à partir de la lumière, en demandant de garder une image z-buffer de la scène. Une fois que les shadow buffers ont été créés, que les objets sans fourrure ont été rendus avec les ombres complètes, et qu'un z-buffer a été calculé à partir de la position de la caméra, les cheveux peuvent être ajoutés à la scène.

La Figure 3.7-8 montre un exemple d'une actrice de synthèse avec une coiffure typique tandis que la Figure 3.7-9 montre une variété de moustaches et barbes. Ces différentes combinaisons ont été produites en changent des paramètres comme la densité ou les facteurs d'échelle et d'orientation.

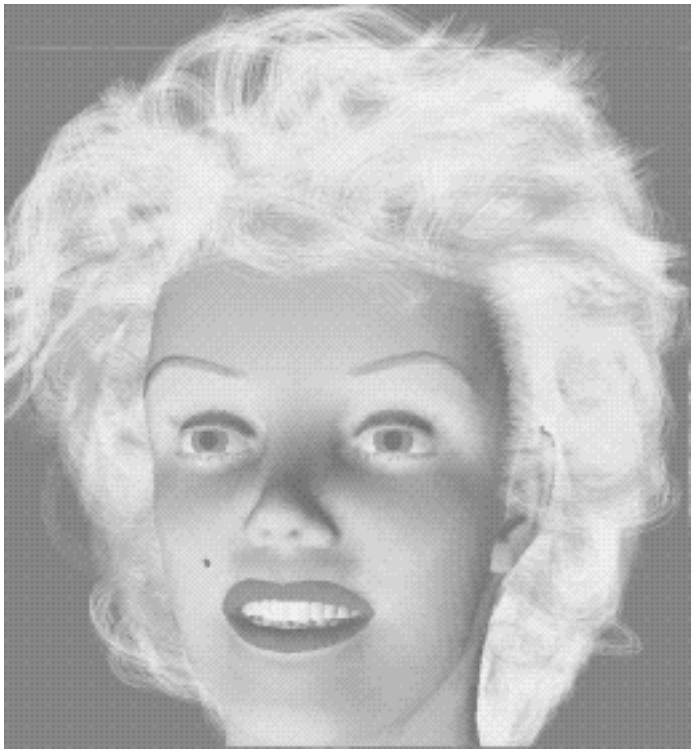


Figure 3.7-8. Actrice de synthèse avec coiffure

Figure 3.7-9. Barbes et moustaches