Gestion dynamique de scènes 3D

Gérard Hégron

IRIA

CENTRE DE RENNES

IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France

Tél. (1) 39 63 55 11

Rapports de Recherche

N° 575

GESTION DYNAMIQUE DE SCÈNES 3D

Gérard HEGRON

Octobre 1986

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 - RENNES CEDEX
FRANCE
Tél. : (99) 36.20.00
Télex : UNIRISA 95 0473 F

PUBLICATION INTERNE N° 313

Octobre 1986

28 pages

GESTION DYNAMIQUE DE SCENES 3D

Gérard HEGRON

DYNAMIC MANAGEMENT OF 3D SCENES

RESUME

Lorsque nous simulons le déplacement d'un observateur ou d'un capteur dans une scène tridimensionnelle (3D) qui contient un très grand nombre d'objets, seul l'ensemble des objets situés dans son champ de vision ou d'interaction, appelé base de données locale, doit être pris en compte afin de minimiser le temps de génération de l'image ou des divers traitements effectés à chaque pas. Dans ce rapport nous présentons un algorithme de gestion dynamique de la base de données locale, c'est à dire une méthode qui permet de gérer l'ensemble des objets de l'univers qui entrent et sortent du champ de vision d'un observateur (capteur, caméra) au cours de son déplacement. La méthode se décompose en trois étapes:

- une partition binaire de l'espace est réalisée à partir des faces des boîtes englobantes des objets définies parallèlement aux axes X, Y et Z;
- un graphe de connexité des régions obtenues est créé;
- la gestion dynamique de la base de données locale est réalisée en représentant le champ de vision du capteur par un cube de rayon R et en exploitant le cohérence spatiale et temporelle entre chaque déplacement (utilisation du graphe de connexité et de critères d'inclusion et d'intersection).

Lorsque la base de données de la scène est très grande, la gestion dynamique de la mémoire est effectuée simultanément en appliquant cette méthode en raisonnant non plus sur les boîtes englobantes des objets mais sur celles de sous – scènes disjointes.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L.A. 227)
UNIVERSITÉ DE RENNES 1 I.N.S.A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

ABSTRACT

When simulating a moving observer or sensor in a 3D scene which contains a large number of objects, only the objects lying in the field of view or of interaction, named the local data basis, have to be taken into account in order to decrease image generation or various treatments computing time at each step. This paper presents an algorithm which achieves the dynamic management of the local data basis, that is to say which manages the set of objects which enter or leave the field of view of an observer (camera, sensor) during its displacement. This method consists broadly of the following three steps:

- a binary space partitionning of the 3D space is done from the objects's bounding boxes by mean of planes perpandicular to the X, Y and Z axes;
- a subregion adjacency graph is created;
- the dynamic management of the local data basis is achieved by modelling the bounded bearing volume of the sensor by a cube of R radius and by using the adjacency graph and the inclusion and intersection criteria to exploit spatial and temporal coherences between each displacement.

When the scene data basis is very large, the dynamic management of the memory is done simultaneously by using this method reasoning with the bounding boxes of disjointed sub-scenes.

GESTION DYNAMIQUE DE SCENES 3D

Gérard HEGRON
IRISA/INRIA
Campus Universitaire de Beaulieu
35042 Rennes Cédex
France

1 INTRODUCTION

A l'heure actuelle, l'une des principales préoccupations en infographie est la possibilité de pouvoir synthétiser rapidement, voir en temps réel (c'est à dire 25 images par seconde), une séquence d'images réalistes d'un environememt tridimensionnel (3D) perçu par un observateur en mouvement. Nous rencontrons cette contrainte temporelle dans de nombreuses applications: simulateurs de pilotage (avions, voitures, etc.), systèmes d'animation de scènes 3D pour l'audiovisuelle, simulateurs pour la commande de robots mobiles. Dans la mesure où le nombre d'objets qui composent la scène est très grand, il devient nécessaire de ne prendre en compte que l'ensemble des objets qui sont situés partiellement ou totalement dans le champ de vision de l'observateur (caméra, capteur) afin de réduire le temps de génération de l'image à chaque pas.

Le problème à résoudre est le suivant:

1. extraire l'ensemble des objets 3D, appelé "base de données locale", qui sont situés dans le champ de vision en fonction de la position initiale de la caméra. Pour réduire le nombre de calculs, ceci peut être effectué en utilisant une description simplifiée des objets: volumes englobants, points caractéristiques.
2. gérer dynamiquement la base de données locale en trouvant les objets qui entrent et sortent du champ de vision durant chaque déplacement élémentaire de la caméra. Afin de tirer partie de la cohérence spatiale et temporelle une nouvelle représentation interne de la scène est nécessaire: graphe de connexité spatiale des objets ou de régions de l'espace.

Dans la littérature, de nombreux articles proposent différentes solutions pour améliorer la vitesse de visualisation d'une scène complexe: soit en utilisant une représentation hiérarchique des objets 3D [1,2], soit en subdivisant tout l'espace en sous-régions [3,4]. Mais ces méthodes sont essentiellement utilisables et efficaces pour la génération d'images statiques parce qu'elles n'exploitent pas la cohérence temporelle mais la cohérence spatiale uniquement. Par conséquent

de nouvelles techniques doivent être mises en oeuvre. Dans ce rapport nous présentons un nouvel algorithme qui tient compte des hypothèses suivantes:

- la vision de l'observateur est bornée;
- les objets contenus dans la scène sont immobiles. Si un objet est localement mobile (comme les robots à poste fixe par exemple) on pourra cependant raisonner sur l'espace fermé dans lequel il se meut (applelé espace de travail dans le cas d'un robot). Par contre, si l'objet se déplace à travers tout l'espace il devra être étudié séparément.

La solution que nous proposons ici permet non seulement de gérer dynamiquement la base de données locale, mais également de gérer l'occupation de la mémoire centrale par la base de données de l'univers 3D préalablement décomposé en sous – scènes disjointes.

2 PRESENTATION GENERALE DE LA METHODE

Dans [5] P.RIVES proposait un algorithme où les objets étaient représentés par un ensemble de points caractéristiques, où la scène était modèlisée par un graphe de connexité spatiale entre les points, et où la base de données locale était formée de l'ensemble des objets (points) appartenant à une sphère de rayon R (portée de la vision) centrée sur la position de l'observateur. Etant donnée la position du capteur au temps K, chaque point de l'espace appartenait à l'une des quatre classes suivantes (voir figure 1):

- classe 1: points qui restent dans la base de données locale au temps $k+1$;
- classe 2: points susceptibles de sortir de la base de données locale au temps $k+1$;
- classe 3: points susceptibles d'entrer dans la base de données locale au temps $k+1$;
- classe 4: points qui restent à l'extérieur de la base de données locale au temps $k+1$;

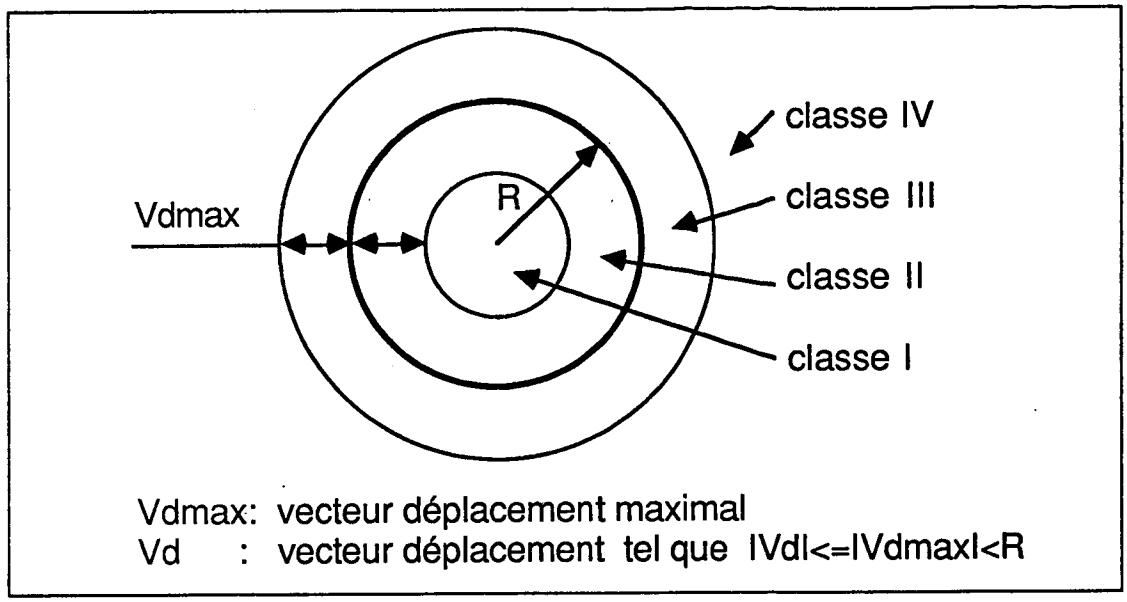

Figure 1. Modèlisation de la base de données locale

La gestion dynamique de la base de données locale était basée sur l'exploitation des propriétés de voisinage des points et sur les techniques de parcours d'un graphe. Entre deux instants K et K+1 l'algorithme ne raisonnait que sur les points qui appartenaient aux classes 2 et 3.

Cet algorithme est performant si le nombre de points caractéristiques n'est pas trop grand par rapport au nombre d'objets; c'est à dire si la taille des objets est petite par rapport à celle du rayon de la sphère de vision. Cette restriction nous a conduit à développer une nouvelle méthode qui se décompose globalement en trois étapes distinctes:

1. une partition binaire de l'espace est réalisée à partir des faces des boîtes englobantes des objets définies parallèlement aux axes X, Y et Z. Les régions de l'espace obtenues sont des parallèlépipèdes rectangles qui sont vides où qui contiennent un objet;
2. un graphe de connexité des régions est créé;
3. la gestion dynamique de la base de données locale est réalisée en représentant le champ de vision du capteur par un cube de rayon R et en exploitant le cohérence spatiale et temporelle (utilisation du graphe de connexité).

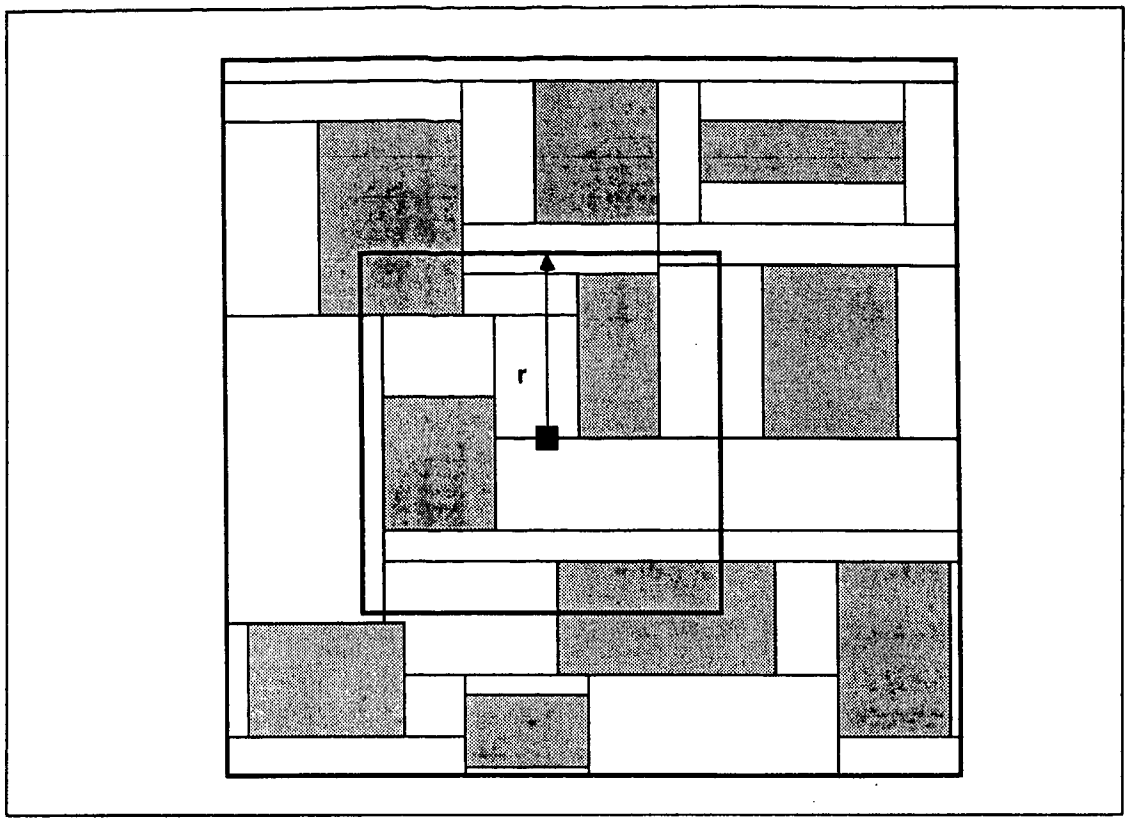

Figure 2. Example d'une scene 2D partitionnée et d'un champ de vision

La valeur R est fonction de la portée du capteur et de l'interaction des objets entre eux: par exemple des objets situés hors du champ du capteur peuvent produire des ombres portées dans la scène locale. Les objets qui sont partiellement ou totalement contenus par le "cube de vision", appartiennent aux régions non vides du graphe qui coupent ce cube (voir figure 2 où les surfaces grisées représentent les boîtes englobantes des objets). Lorsque la caméra effectue une rotation pure la BD locale est invariante.

Les deux premières étapes sont effectuées hors-ligne et les résultats sont exploités lors de la gestion dynamique proprement dite.

3 PARTITION DE L'ESPACE

La scène est divisée en régions par le biais d'une partition binaire de l'espace (BSP:"Binary Space Partitioning") à partir des faces des boîtes englobantes des objets [7]. La procédure exposée par FUCHS dans [6] est appliquée en raisonnant sur l'ensemble F des faces des boîtes.

Le plan passant par une face f_k divise l'espace E en deux demis – espaces (E^+_k pour le côté positif et E^-_k pour le côté négatif), partage l'ensemble des faces de F appartenant à E en deux sous – ensembles F^+_k et F^-_k appartenant respectivement à E^+_k et E^-_k, et découpe la boîte englobante R de l'espace E en deux boîtes R^+_k et R^-_k. Cette procédure générale est appliquée récursivement sur chacun des sous – espaces E^+_k et E^-_k tant que l'ensemble des faces associé n'est pas vide. Les faces à cheval sur le plan de découpage sont divisées en deux parties appartenant aux deux demis – espaces respectifs engendrés.

A l'initialisation de la partition binaire, E représente tout l'espace 3D, R la boîte englobante de toute la scène, et F l'ensemble des faces de toutes les boîtes englobantes des objets contenus dans la scène. La procédure récursive à la forme générale qui suit.

```
Procédure Construire_arbre(F:liste_faces; B:liste_boîtes; R:région):arbre
Début
    Si F est non vide
    alors /* région non terminale */
        Choisir face f_k dans F;
        F+_k:=vide;F-_k:=vide;
        B+_k:=vide;B-_k:=vide;
        Découper_région(R,f_k,R+_k,R-_k);
        Pour chaque face f_i de F faire
        début
            si i<>k alors
            début
                Découper_face(f_i,f_k,f+_i,f-_i);
                Ajouter_face(F+_k,f+_i);
                Ajouter_face(F-_k,f-_i);
            fin
        fin
        Pour chaque boîte b_j de B faire
        début
            Découper_boîte(b_j,f_k,b+_j,b-_j);
            Ajouter_boîte(B+_k,b+_j);
            Ajouter_boîte(B-_k,b-_j);
        fin
        combiner_arbre( construire_arbre(F+_k,B+_k,R+_k),
                        face f_k,
                        construire_arbre(F-_k,B-_k,R-_k));
    sinon /* R région terminale */
```

Attribuer la région R à la feuille de l'arbre;
si B est vide
alors liste_objets(R) := vide;
sinon liste_objets(R) := liste des objets de B;
Fin.

Comme les faces des boîtes sont parallèles aux plans XoY, XoZ ou YoZ les opérations arithmétiques utilisées sont uniquement des tests. Le résultat obtenu est représenté par un arbre binaire appelé arbre-BSP ("BSP-tree") dont les noeuds non-terminaux contiennent une face f_k et dont les feuilles représentent les régions de l'espace. A chaque région sont associés ses dimensions (bornes minimales et maximales suivant x, y et z) et la liste des objets qu'elle contient partiellement ou totalement si elle n'est pas vide. Cette liste d'objets est calculée en maintenant parallèlement à la liste F des faces, lors de la construction de la partition binaire, une liste B des boîtes englobantes des objets que l'on découpe également récursivement en deux sous-listes situées de part et d'autre du plan de découpage.

La figure 3 illustre la partition binaire d'une scène 2D . La figure 4 représente l'arbre-BSP obtenu.

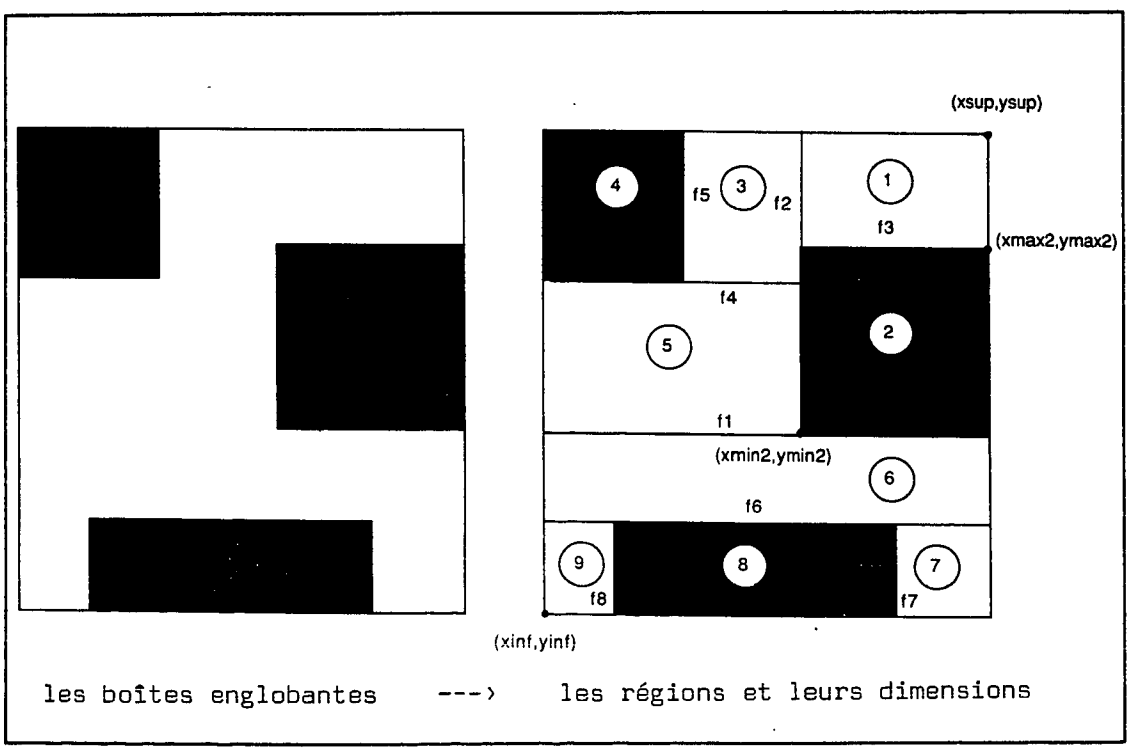

Figure 3. Partition binaire d'une scène 2D

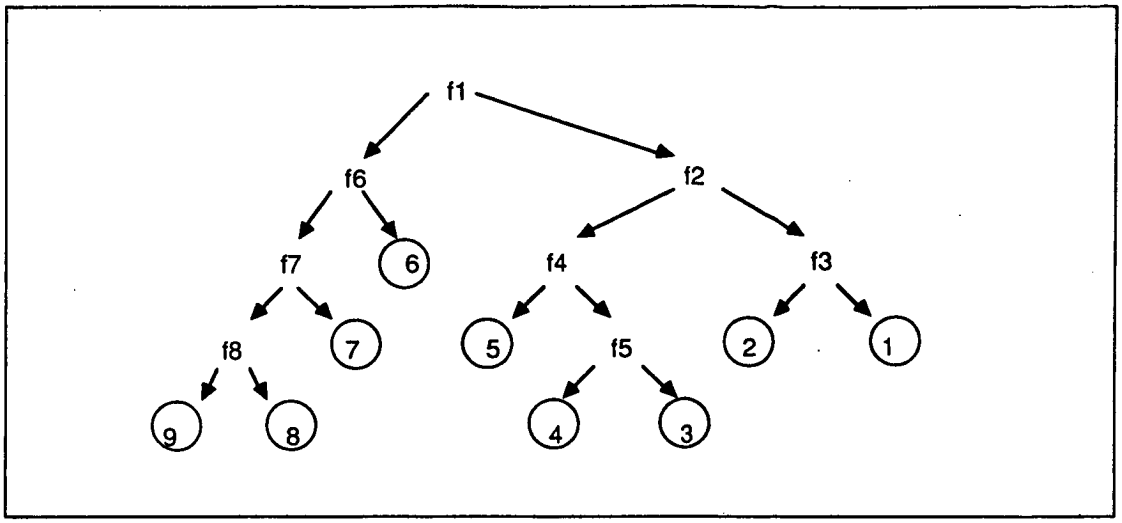

Figure 4. Arbre – BSP

La région à laquelle appartient un point de l'espace est obtenue à l'aide de la procédure qui suit.

Procédure Région_point(p: point; arbre: arbre – BSP): région
Début
    Si racine(arbre) n'est pas une feuille
    alors /* noeud non terminal */
        si p est du côté positif de la face de racine(arbre)
        alors région_point(p;sous_arbre_positif(racine(arbre)));
        sinon région_point(p;sous_arbre_négatif(racine(arbre)));
    sinon /* noeud terminal */
        région_point: = région_de(racine(arbre));
Fin.

4 CREATION DU GRAPHE DE CONNEXITE

Deux régions sont connexes si deux faces d'entre elles se recouvrent (partiellement ou totallement). Deux faces qui se touchent uniquement en un sommet ou sur une arête ne sont pas adjacentes (6 – connexité). Pour chaque région nous définissons six listes de régions adjacentes, une pour chaque face de la boîte (voir figure 5).

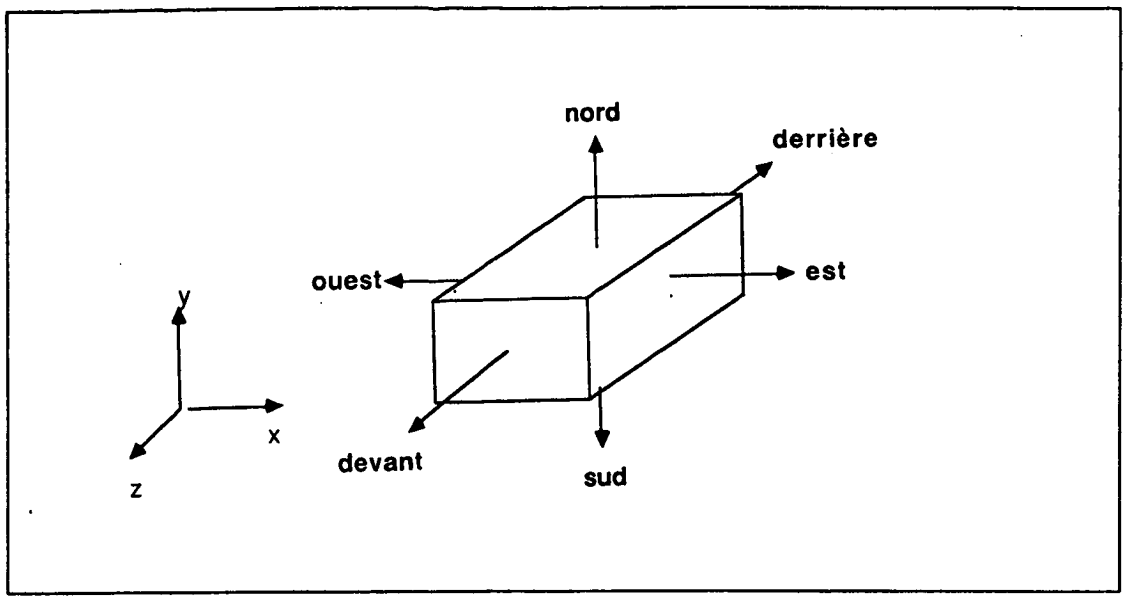

Figure 5. Les 6 directions de connexité

Les deux méthodes proposées

Le graphe peut être généré soit pendant la partition binaire (voir [7]) de l'espace soit après.

Pendant la génération de l'arbre – BSP, la région R obtenue en chaque noeud est divisée en deux sous – régions connexes R^+ et R^- par une face f de F. Dans le graphe la région R est remplacée par R^+ et R^- et ce dernier est mis à jour en fonction de la position de la face f et des anciennes listes des régions reliées à R (voir figure 6). Au début du processus la région R est la boîte englobant toute la scène et le graphe n'est formé que de cette dernière dont les listes des régions adjacentes sont vides.

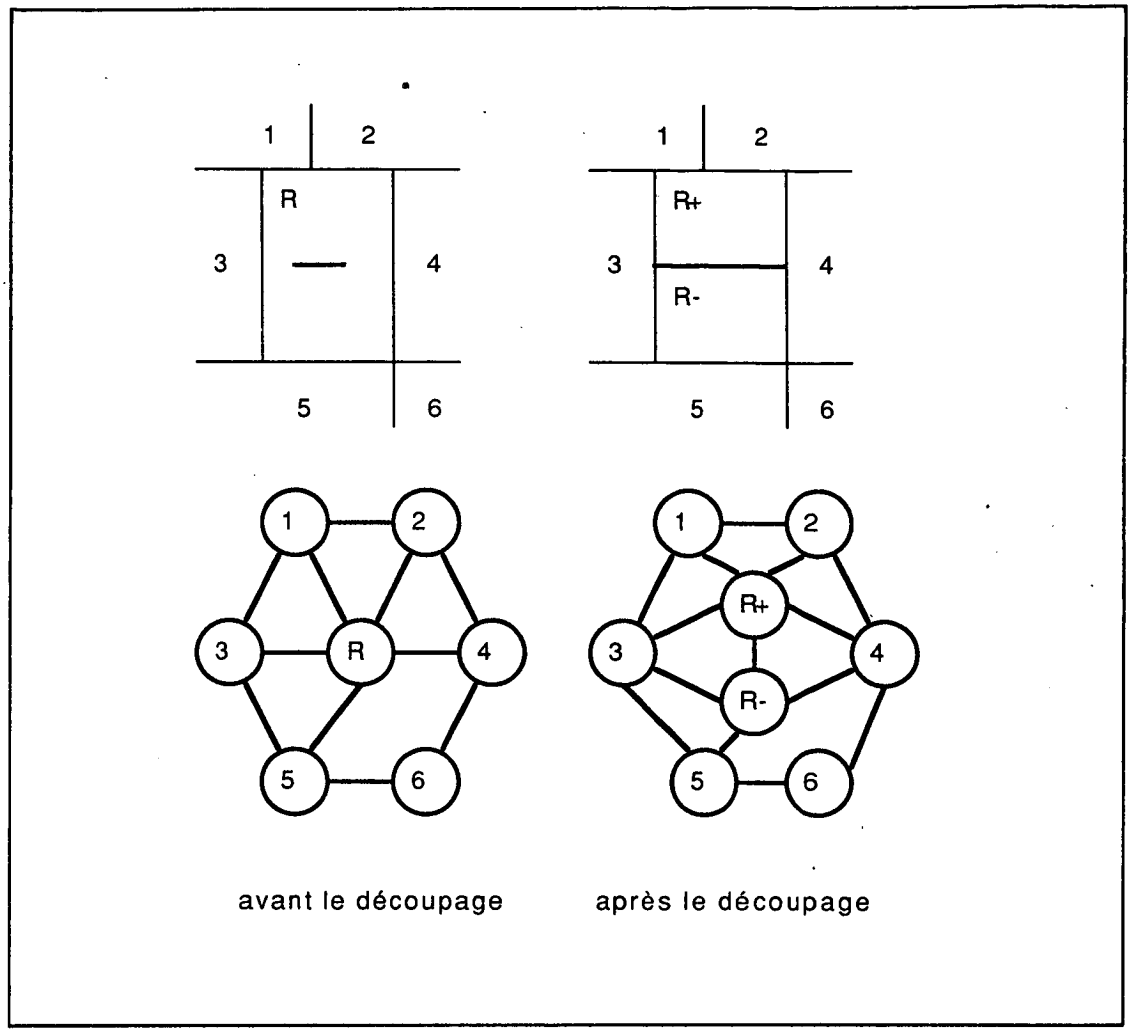

Figure 6. Création du graphe connexité pendant celle de l'arbre – BSP

Après la partition binaire de l'espace, nous raisonnons sur les sous – régions terminales. L'algorithme est fondé sur la propriété suivante: les points de l'espace situés au voisinage des faces d'une région et à l'extérieur de celle – ci appartiennent aux régions connexes. Pour chacune des faces un voisinage est alors défini en créant une face extérieure plus petite que la précédente en ajoutant plus ou moins epsilon sur les extrémités de la face courante. Cette nouvelle face est ensuite découpée par un parcours de l'arbre – BSP et les feuilles de l'arbre dans lesquelles les morceaux de la face terminent leur parcours nous donnent les régions adjacentes (voir figure 7).

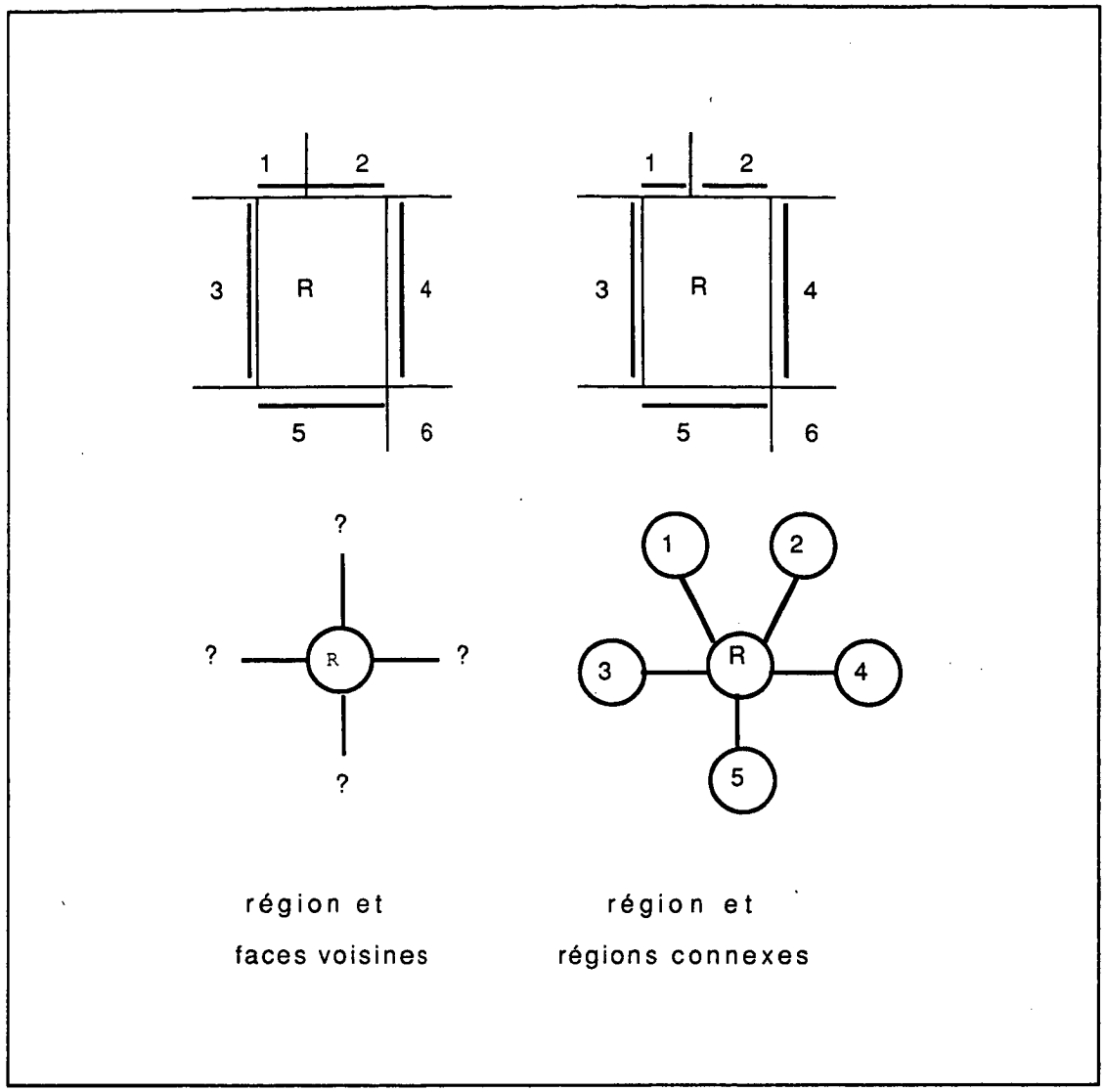

Figure 7. Création du graphe connexité après celle de l'arbre – BSP

Comme la connexité entre deux régions est symétrique la liste des régions terminales est explorée en effectuant un parcours (gauche droite par exemple) de l'arbre – BSP où les sous – arbres (gauches par exemple) complètement explorés lors de ce parcours sont bloqués pour arrêter les morceaux des faces définissant un voisinage lors de leur parcours de l'arbre. Le programme général qui suit réalise la création du graphe selon cette méthode.

Création du graphe de connexité
Debut
Parcours_arbre_BSP(racine(arbre_BSP));
Fin.

```
Procédure Parcours_arbre_BSP(noeud:noeud_arbre_BSP)
Début
    Si noeud non terminal
    alors
        parcours_arbre_BSP(fils_gauche(noeud)); /* côté négatif */
        sous_arbre_gauche(noeud)): = traité;
        parcours_arbre_BSP(fils_droit(noeud)); /* côté positif */
    sinon /* noeud terminal */
        traiter_région_du(noeud);
Fin.

Procédure Traiter_région_du(noeud:noeud_arbre_BSP)
Début
    R: = région_du(noeud);
    Pour chacune des 6 directions dir_i de connexité faire
    début
        créer_face_voisinage(dir_i,fv_i);
        connexité(fv_i,dir_i,arbre_BSP) ;
    fin
Fin.

Procédure Connexité(fv:face_voisinage;dir:direction;arbre:arbre_BSP)
Début
    si racine(arbre) n'est pas une feuille
    alors
        découper_face(fv,face(racine(arbre)),fv+,fv-);
        si sous_arbre_gauche(racine(arbre)) non traité
           alors connexité(fv-,dir,sous_arbre_gauche);
                 connexité(fv+,dir,sous_arbre_droit);
    sinon /* noeud terminal */
        région_de(racine(arbre)) connexe à R suivant dir;
        R connexe à région_de(racine(arbre)) suivant direction opposée à dir;
Fin.
```

Comparaison des méthodes

Soit k la profondeur moyenne de l'arbre – BSP. Le temps passé pour la création du graphe de connexité au cours de la génération de l'arbre – BSP est donné par l'expression:

$C_1 = T_1.2^{k+1}$

où T_1 est le temps moyen passé pour mettre à jour le graphe en chaque noeud de l'arbre.

Le temps passé pour créer le graphe après la construction de l'arbre-BSP est donné par l'équation:

$C_2 = 6.T_2.k2^k$

où T_2 est le temps moyen passé pour découper une face représentant un voisinage en chaque noeud de l'arbre.

D'où $C_2 > C_1$ si $k > (T_1/3T_2)$

A partir des temps d'exécutions prélevés sur un ensemble de neuf scènes (voir figure 8) nous avons évalué que, approximativement ,$T_1 = 60.T_2$.

D'où $C_2 > C_1$ si $k > 20$.

En d'autres termes ceci signifie que la seconde méthode est plus rapide que la précédente tant que le nombre de régions terminales ($N_t = 2^k$) reste inférieur à 10^6 environ.

scènes	nombre de primitives	nombre de régions	temps en seconde C1	temps en seconde C2
1	2	7	0.5	0.2
2	4	33	0.52	0.18
3	6	38	0.43	0.21
4	8	73	1.26	0.48
5	12	115	1.93	0.77
6	12	337	8.70	4.08
7	24	246	6.35	1.99
8	36	380	9.71	3.27
9	72	716	16.74	6.66

Figure 8. Temps CPU de création du graphe de connexité sur VAX 11/750

Réduction du nombre de régions dans le graphe

Le nombre de régions du graphe peut être minimisé à deux niveaux.

Dans un premier temps, lors de la partition binaire de l'espace on choisit parmi l'ensemble F des faces de la région en cours de partitionnement la face f_k qui coupera un minimum de faces de F en deux.

Dans un deuxième temps, le nombre de régions du graphe peut être réduit une fois le graphe de connexité construit. Les boîtes englobantes des objets contiennent un certain nombre de régions terminales. Comme ces régions sont totalement incluses dans la boîte, l'idée est de substituer toutes ces régions par la boîte elle-même. Cette opération n'est effectuée que pour les boîtes englobantes disjointes . L'algorithme consiste à effectuer pour chaque boîte disjointe le traitement suivant:

1. trouver une région du graphe intérieure à la boîte englobante en effectuant avec le centre de la boîte un parcours de l'arbre – BSP: la feuille de l'arbre dans laquelle ce point termine son parcours nous donne cette région;
2. parcours du graphe à partir de cette région intérieure suivant les 6 directions: si la région parcourue n'est pas intérieure à la boîte elle lui est connnexe.

Les procédures utilisées ont la forme générale qui suit.

Prodédure Réduction_graphe(boîte: boîte_englobante)
Début
R_0: = Région_point(centre(boîte),arbre_BSP);
R_0 est intérieure à boîte;
Pour chaque direction dir_i (i = 1 à 6) **faire** Parcours_graphe(R_0,dir_i);
Fin.

Procédure Parcours_graphe(R:région;dir:diretion);
Début
Pour chaque région R_c connexe à R suivant dir **faire**
début
si R_c n'est pas déjà intérieure ou connexe à boîte
alors
si R_c est intérieure à boîte
alors Pour toute direction exeptée celle opposée à dir **faire**
Parcours_graphe(R_c,direction);
sinon R_c est connexe à boîte suivant la direction courante;
Enlever R de la liste des régions connexes à R_c suivant la direction opposée à la direction courante;
Ajouter boîte à cette liste;
Ajouter R_c à la liste des régions connexes à boîte suivant la direction courante;
finsi
finsi
fin;
Fin.

L'ensemble des régions intérieures à la boîte sont substituées par cette dernière dans l'arbre – BSP. Pour déterminer si une région est intérieure à la boîte englobante nous utilisons le critère d'intersection défini dans le paragraphe suivant.

La figure 9 ci-dessous illustre l'opération de réduction du graphe pour une boîte englobante B où cinq régions (10, 11, 12, 13, 14) sont remplacées par B.

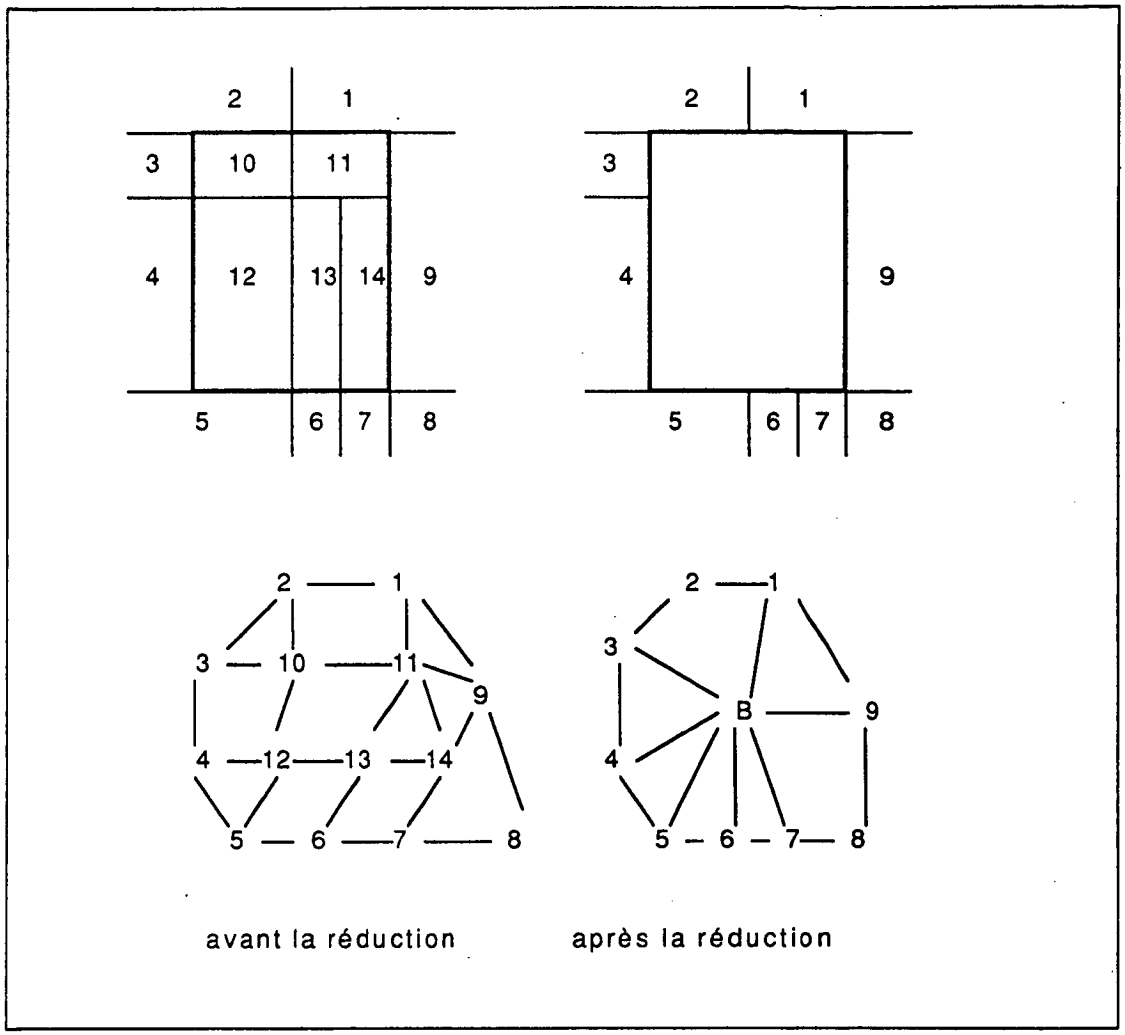

Figure 9. Réduction du graphe

5 GESTION DYNAMIQUE DE LA BASE DE DONNEES LOCALE

Au début du déplacement du capteur les objets qui appartiennent à la base de données locale sont situés dans les régions non vides qui coupent le cube de vision . Puis après chaque déplacement élémentaire nous connaissons les objets qui entrent ou sortent du champ de vue en calculant les régions qui pénètrent ou quittent le cube de vision (voir figure 10).

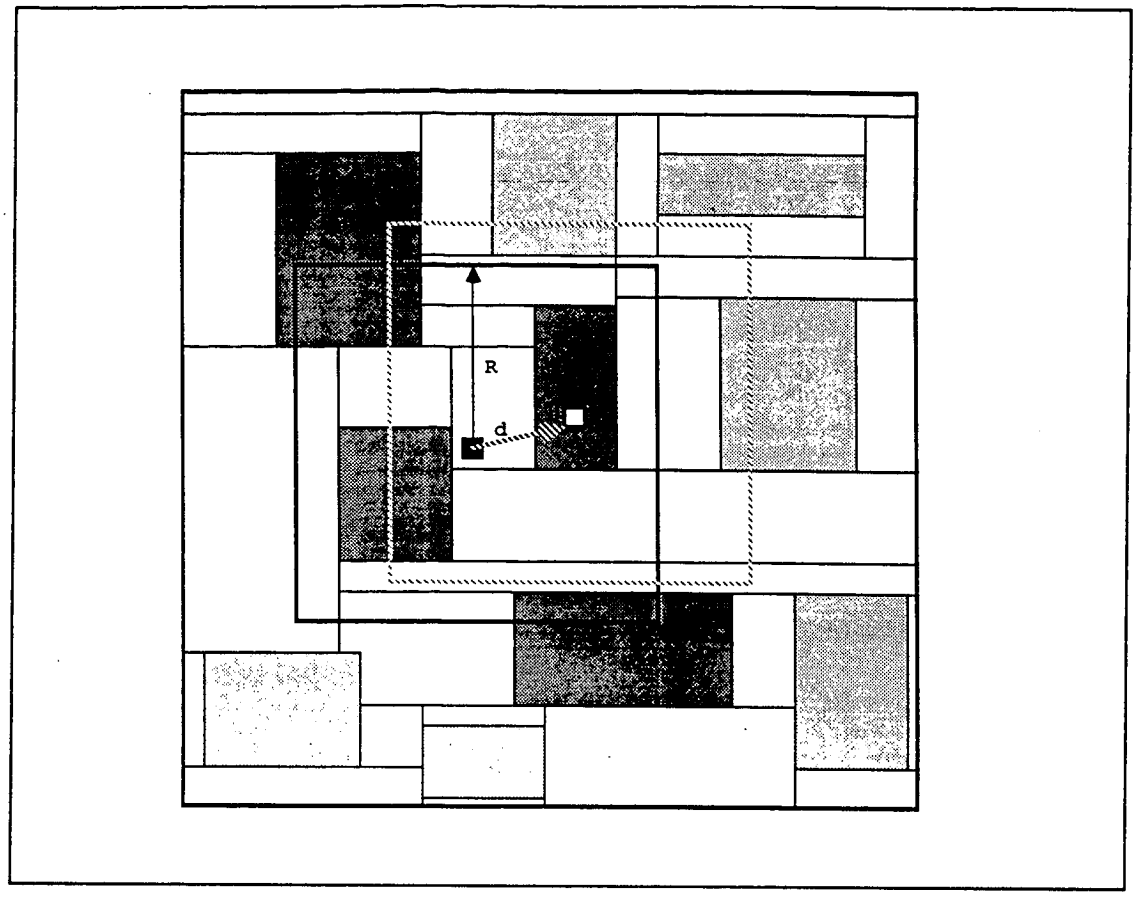

Figure 10. Gestion dynamique de la base de données locale

Pour tirer partie de la cohérence spatiale nous utilisons le graphe de connexité; pour exploiter la cohérence temporelle nous mettons à jour de façon incrémentale un critère d'intersection et un critère d'inclusion. Avant de développer l'algorithme de gestion dynamique, spécifions ces deux critères.

Définitions

Une région R_k de l'espace est définie par ses

- extrémités ($umin^k$, $umax^k$)
- rayons $r^k_u = (umax^k - umin^k)/2$
- centres $c^k_u = umin^k + r^k_u$

suivant les trois axes u = x, y, z.

De même le cube de vision C est caractèrisé par ses centres $(C_u)_{u=x,y,z}$ et ses rayons r qui sont identiques suivant les trois axes.

Le critère d'intersection est défini par

$$Int^k_u = distance(c^k_u, c_u) - (r^k_u + r)$$

si $Int^k_u < 0$ pour tout $u = x,y,z$

alors il y a intersection entre R_k et C

sinon il n'y a pas d'intersection;

Le critère d'inclusion est défini par

$$Inc^k_u = distance(c^k_u, c_u) + r^k_u - r$$

si $Inc^k_u <= 0$ pour tout $u = x,y,z$

alors R_k est inclus dans C

sinon R_k n'est pas inclus dans C;

Si $Int^k_u < 0$ ou si $Inc^k_u <= 0$ pour u = x, y, ou z, nous disons que R_k coupe C ou respectivement est inclus dans C suivant l'axe u.

L'algorithme

La liste des régions qui coupent le cube de vision C est initialisé par les deux étapes suivantes:

- trouver la région R_o à laquelle appartient le centre de C par un parcours de l'arbre – BSP (R_o coupe C);
- calculer les régions R_k qui coupent C en parcourant le graphe de connexité à partir de R_o et en utilisant les critères d'intersection et d'inclusion.

Après chaque déplacement $(d_u)_{u=x,y,z}$ de la caméra la liste des régions et des objets qui sont situés partiellement ou totallement (inclusion) dans C sont mis à jour par le processus qui suit.

Gestion de la BD locale((d_u): déplacement)
Début

Mettre à jour le centre de C: $(c_u := c_u + d_u)$;
Pour chaque région R_k coupant C faire

```
début
    Mettre à jour le critère d'intersection Int^k_u;
    si il existe u tel que Int^k_u > = 0
    alors
        début
            R_k quitte la liste des régions qui coupent C;
            si R_k n'est pas vide alors mettre à jour la BD locale;
        fin
    sinon
        début
            Suivant l'octant dans lequel le vecteur déplacement est situé,
            sélectionner les trois directions suivant lesquelles nous progressons
            dans le graphe à partir de R_k pour trouver les nouvelles régions
            qui coupent C. Soient (dir_u) ces trois directions;
            Pour chaque direction dir_u faire
                si Inc^k_u > = 0 alors rechercher suivant la direction
                dir_u à partir de R_k les nouvelles régions qui coupent C, et
                mettre à jour la BD locale quand ces régions ne sont pas vides;
            fin
            Mettre à jour le critère inclusion Inc^k_u;
        fin;
fin;
Fin.
```

Dans l'algorithme de gestion de la BD locale nous recherchons dans le graphe les nouvelles régions qui pénètrent dans le cube de vision suivant trois directions uniquement. Ces trois directions privilégiées sont les seules selon lesquelles nous pouvons rencontrer de nouvelles régions et sont fonction de l'octant dans lequel est situé le vecteur déplacement. Par exemple si le vecteur déplacement est dans le premier octant ces trois directions seront nord, est et devant (voir figure 11).

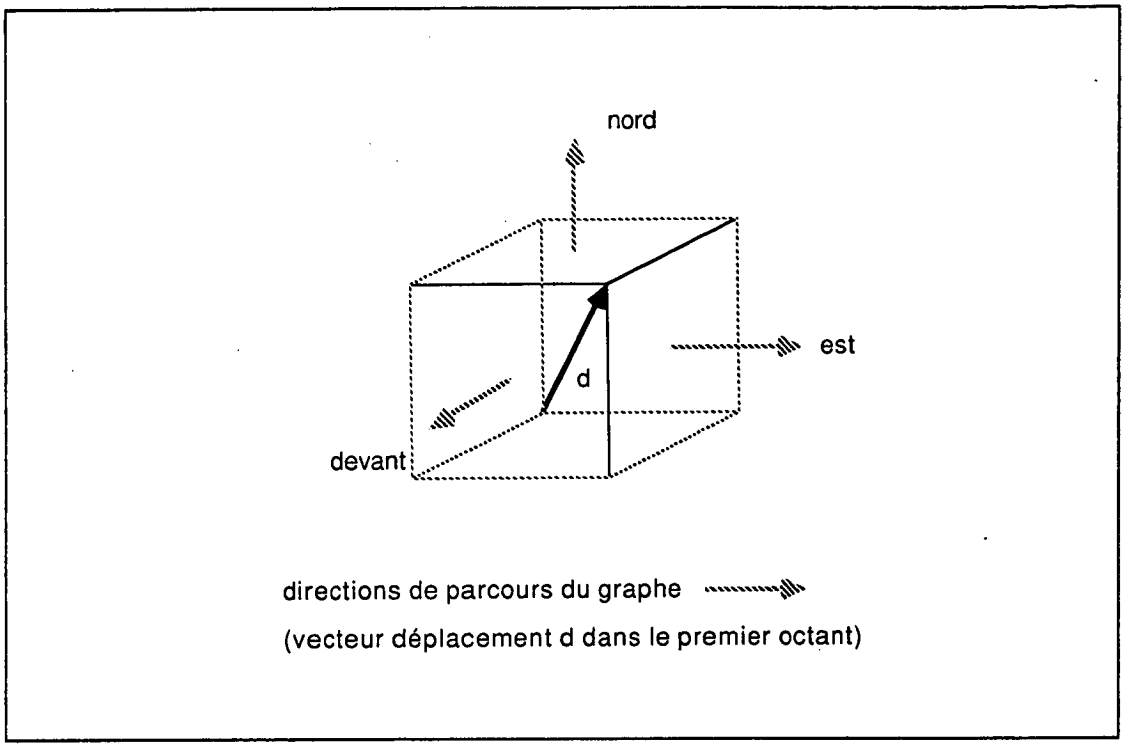

Figure 11. Directions de recherche dans le graphe

Pour minimiser encore la progression dans le graphe, nous utilisons le critère d'inclusion. Si avant le déplacement une région R_k coupant le cube de vision est incluse dans celui-ci suivant l'axe u associé à la direction d'exploration courante (c.a.d $Inc^k_u < 0$) nous ne parcourons pas le graphe à partir de R_k suivant cette direction (voir figure 12). En d'autres termes, nous ne parcourons le graphe qu'à partir des régions à cheval sur la frontière du cube de vision et que dans les directions pour lesquelles le critère d'inclusion n'est pas vérifié.

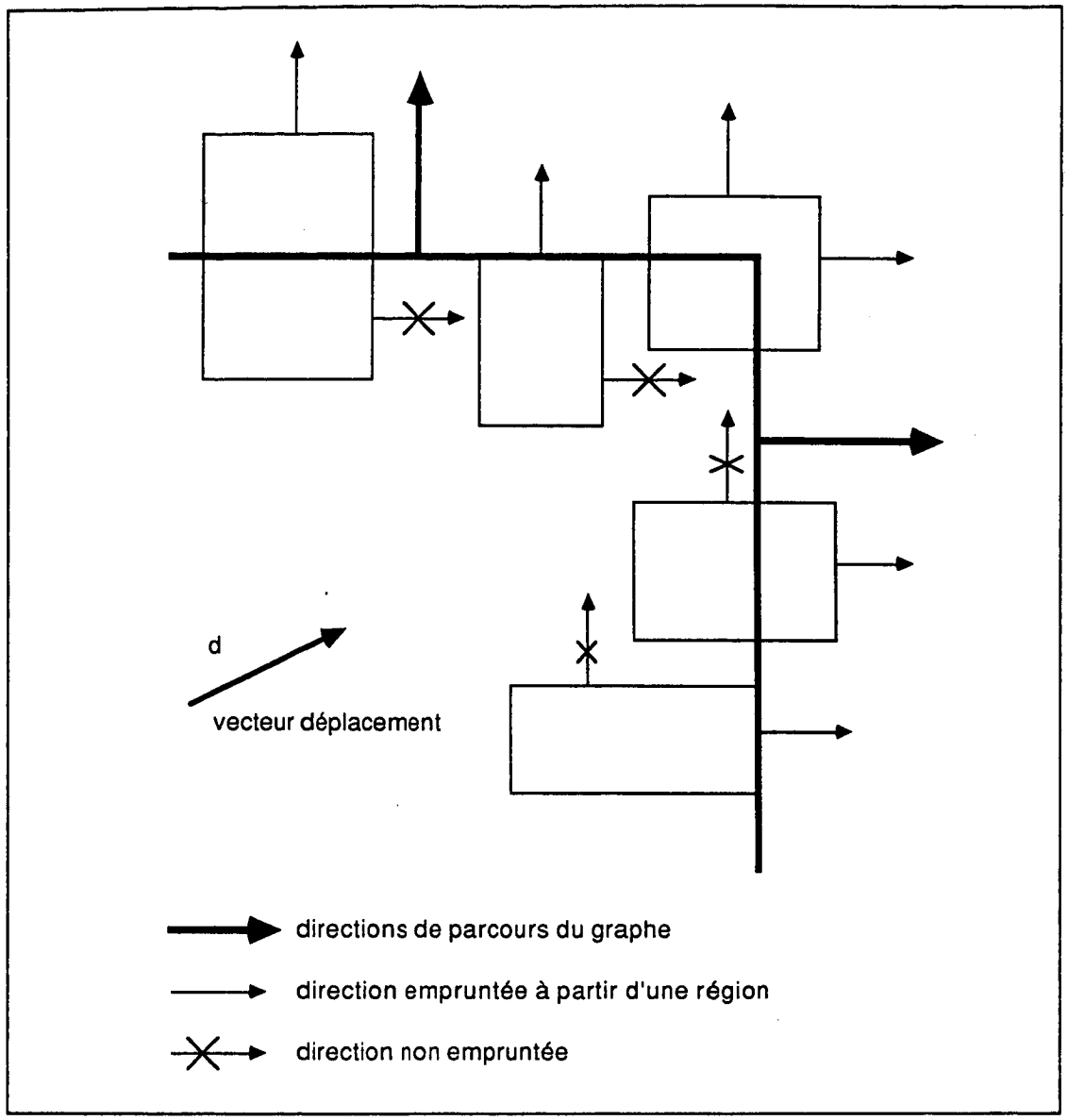

Figure 12. Utilisation du critère d'inclusion

6 GESTION DYNAMIQUE DE LA MEMOIRE

Lorsque la taille de la base de données qui décrit la scène dépasse la capacité de la mémoire centrale du calculateur, il est possible de gérer la mémoire vive en n'y chargeant qu'un sous-ensemble de la BD totale. L'idée repose sur une généralisation de la technique de gestion dynamique de la base de données locale développée plus haut en raisonnant non plus sur les boîtes englobantes des objets de la scène mais sur les boîtes englobantes de sous-scènes.

Supposant que nous disposions d'un ensemble de sous – scènes dont les boîtes englobantes sont disjointes, la méthode de gestion de la mémoire se décompose comme pour la gestion de la BD locale en trois étapes:

1. une partition binaire de l'espace est réalisée à partir des faces des boîtes englobantes des sous – scènes définies parallèlement aux axes X, Y et Z.
2. un graphe de connexité des régions est créé;
3. la gestion dynamique de la mémoire est réalisée en représentant le champ de vision du capteur par un cube de rayon R et en exploitant le cohérence spatiale et temporelle (utilisation du graphe de connexité et des critères d'intersection et d'inclusion). Elle nous permet de connaître les sous – scènes qui entrent et sortent du cube de vision et de ne conserver en mémoire que celles qui interagissent avec ce dernier.

Les algorithmes utilisés sont identiques à ceux mis en oeuvre pour la gestion de la BD locale.

Connaissant à chaque position de la caméra l'ensemble des sous – scènes partiellement ou totalement contenues dans le cube de vision, il est alors possible de gérer dynamiquement pour chacune d'entre elles l'ensemble des objets qui pénètrent ou quittent le cube de vision c'est à dire leurs bases de données locales (voir figure 13).

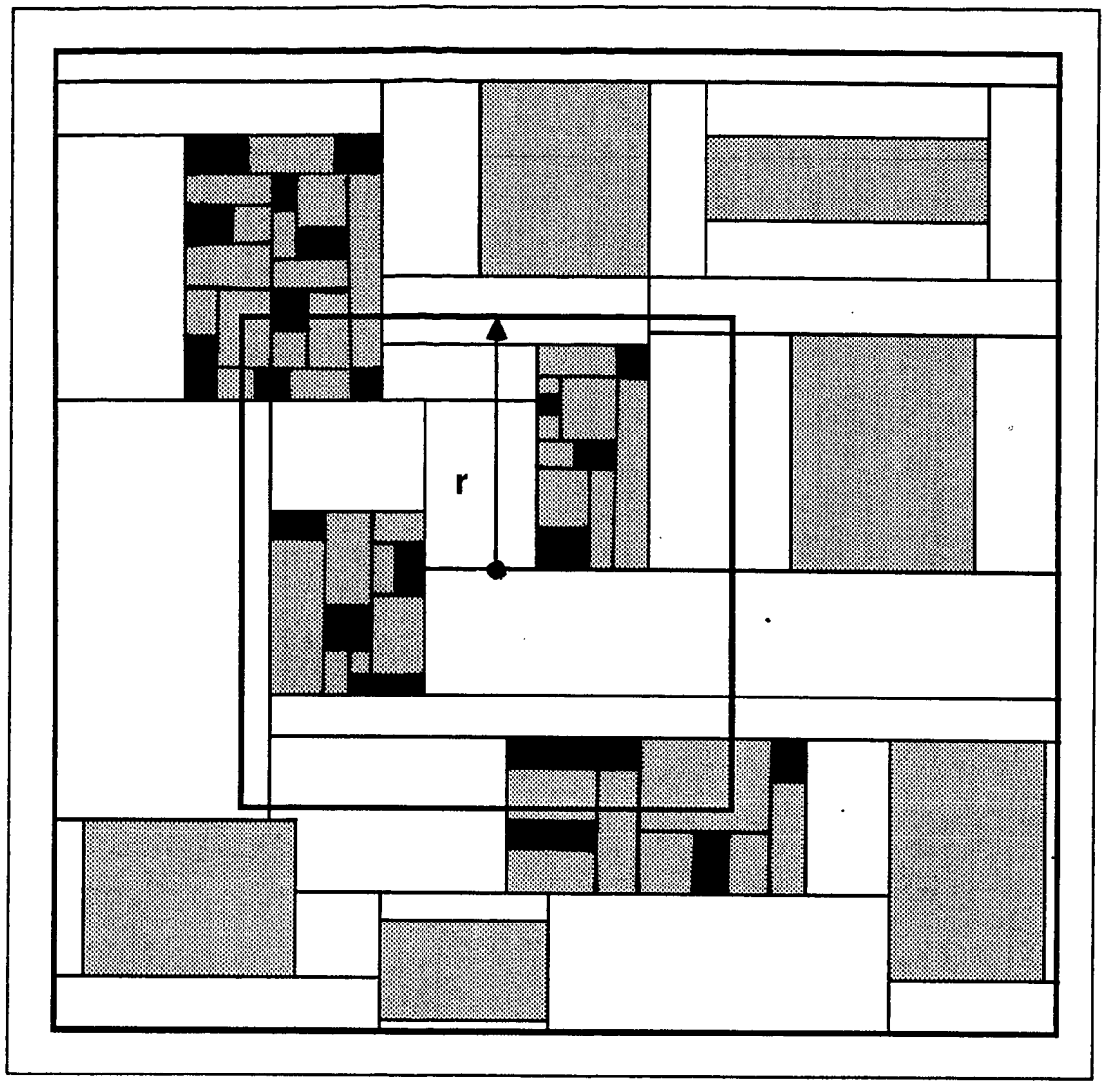

Figure 13. Gestion dynamique de la scène 3D

Pource faire, nous disposons donc

- d'une partition binaire de la scène (arbre – bsp et graphe de connexité) réalisée à partir des sous – scènes disjointes.
- d'une partition binaire de chaque sous – scène (arbre – bsp et graphe de connexité) réalisée à partir des objets.

L'algorithme qui combine à la fois la gestion dynamique de la mémoire et de la base de données locale à la forme générale suivante:

Gestion dynamique de la base de données d'une scène 3D

Début

Initialisation de la BD locale des sous – scènes:

– extraction de la liste des régions qui coupent le cube de vision C en parcourant le graphe de connexité (de la scène) à partir de la région contenant le centre du cube;

– BD locale des sous – scènes: = ensemble des sous – scènes contenues par des régions non vides qui coupent C;

Initialisation de la BD locale des objets:

– pour chaque sous – scène$_i$ de la BD locale extraction de la liste des régions qui coupent le cube de vision C en parcourant le graphe de connexité (de la sous – scène) à partir de la région contenant le centre de l'intersection du cube de vision et de la boîte englobante de la sous – scène;

– pour chaque sous – scène$_i$ la (BD locale des objets)$_i$: = ensemble des objets contenues par des régions non vides qui coupent C;

– BD locale des objets: = UNION (BD locale des objets)$_i$;

Pour chaque déplacement (d_u) de la caméra **faire**

début

Gestion de la BD locale des sous – scènes((d_u));

Pour chaque sous – scène$_i$ de la BD locale **faire**

début

Gestion de la BD locale des objets$_i$((d_u));

fin

BD locale des objets: = UNION (BD locale des objets)$_i$;

fin;

Fin.

7 APPLICATIONS ET CONCLUSION

Dans ce rapport nous avons développé une méthode de gestion dynamique de scènes 3D qui permet non seulement de connaître l'ensemble des objets qui entrent et sortent du champ de vision d'un observateur (capteur) en mouvement mais également de gérer dynamiquement l'occupation de la mémoire lorsque la base de données de l'univers 3D est très grande.

Nous avons intégré l'algorithme de gestion dynamique de base de données locale dans un

système de simulation qui permet de visualiser les images perçues par un robot mobile en mouvement dans un univers tridimensionnel [8]. L'une des hypothèses de base consiste en l'utilisation de capteurs possèdant une portée bornée et pouvant avoir un très grand angle de vue (du type fish – eye), voir une vision sphèrique;

Dans ce simulateur nous distinguons trois sous – ensembles fonctionnels (voir figure 14):

- création et gestion dynamique de la scène;
- synthèse de l'image de l'univers local vu par la caméra que porte le robot;
- analyse de la séquence d'images et reconnaissance des formes qui permettent la génération du mouvement du robot en fonction de la tâche à accomplir dans un cadre "boucle fermée" sur l'environnement local.

Le module de gestion de l'univers local fournit à l'algorithme de synthèse d'image la liste des objets contenus dans le champ de vision du capteur.

Cette technique de gestion dynamique de scène 3D pourrait être également utilisée à d'autres fins:

- calcul des objets accessibles dans un espace de travail: on détermine dans un premier temps les objets qui coupent la boîte englobant l'espace de travail. La boîte englobante joue ici le rôle du cube de vision. Puis dans une deuxième phase on extrait parmi ces objets ceux qui pénètrent réellement l'espace de travail en raisonnant sur les modèles géométriques exactes;
- évitement d'obstacles: gestion dynamique des objets susceptibles de rencontrer un objet mobile avant chaque mouvement en définissant autour de lui un anneau de protection (boîte englobante) qui jouerait comme précédemment le rôle du champ de vision.

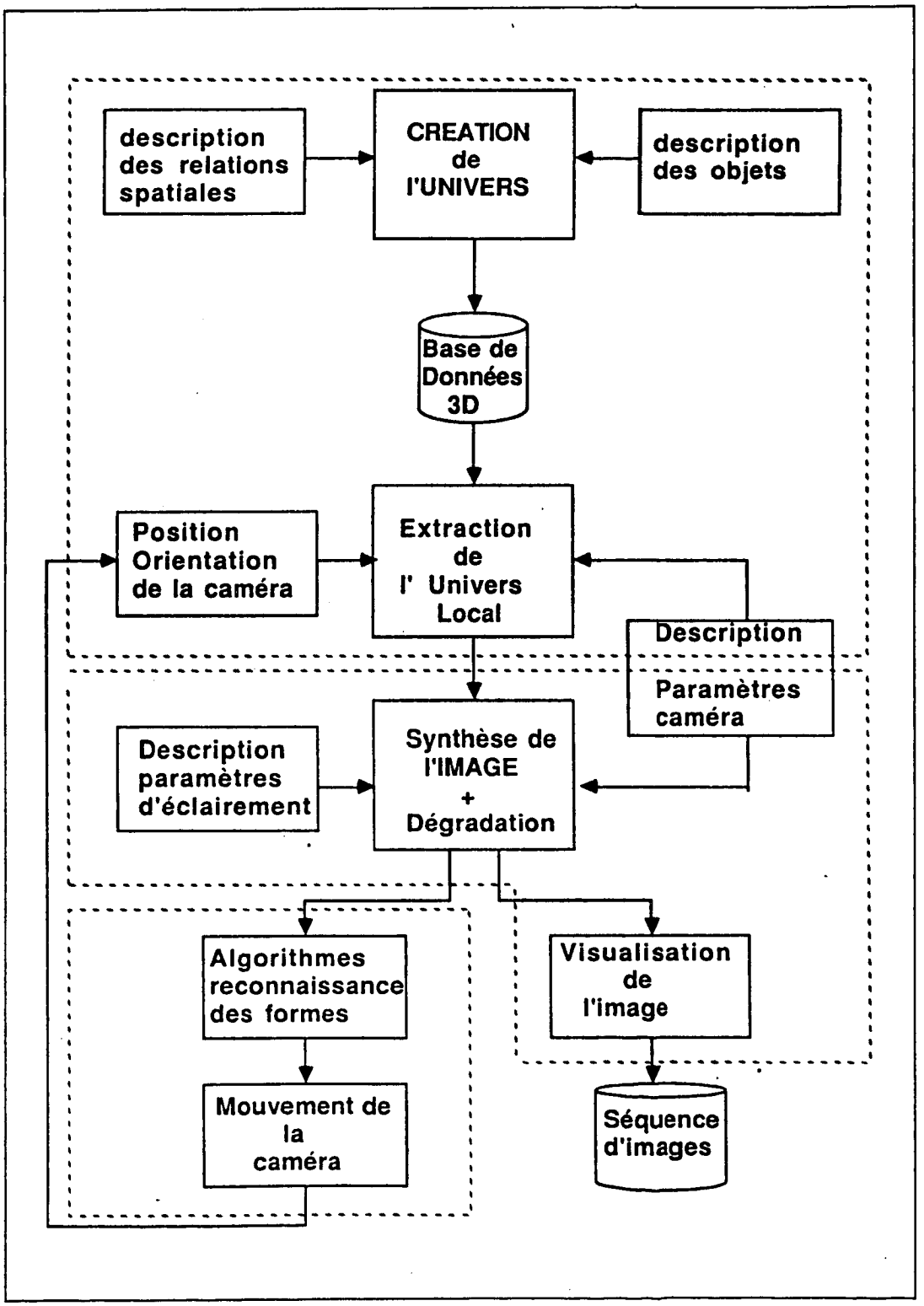

Figure 14. Schéma bloc du système de simulation

BIBLIOGRAPHIE

[1] **J.H. CLARK**
Hierarchical geometric models for visible surface algorithms.
CACM, 19 – 10, October 1976, pp.547 – 554

[2] **S.M. RUBIN, J.T. WHITTED**
A 3 – dimensional representation for fast rendering of complex scenes.
Proc. of SIGGRAPH'80, July 1980, pp.110 – 116

[3] **M. DIPPE, J. SWENSON**
An adaptative subdivision algorithm and parallel architecture for realistic image synthesis.
Computer Graphics, Vol. 18, n°3, July 1984.

[4] **G. WYVILL, T.L. KUNII**
A fonctional model for constructive solid geometry
The Visual Computer (1985) 1:3 – 14.

[5] **G. HEGRON, P. RIVES**
Modèlisation et gestion d'un univers 3D: une première approche à partir du logiciel PADL – 2.
Pubication Interne IRISA, n°269, Octobre 1985

[6] **H. FUCHS, Z.M. KEDEM**
On visible surface generation by a priori tree structures.
Proc. of SIGGRAPH'80, July 1980, pp.124 – 133.

[7] **B. ARNALDI, Th. PRIOL**
Synthèse d'image par lancer de rayon, Subdivision spatiale, Algorithmes et Architecture.
Rapport DEA Informatique, Univ. de RENNES I, Juin 1986

[8] **P. RIVES, G. HEGRON**
Design of a simulation tool for robots using vision sensors.
Proc. of the NATO Workshop on Languages for Sensors – based Control in Robotics, Castelvecchio Pascoli, Italy, September 1986.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique