N° d'ordre : 1246

UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

THÈSE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR DE 3ème CYCLE

Maxime PELERIN

SYNTHESE D'IMAGES ET PARALLELISMES : ALGORITHMES ET ARCHITECTURES.

Thèse soutenue le 3 janvier 1985 devant la Commission d'Examen

MEMBRES DU JURY :

Président	M. DAUCHET
Rapporteur	V. CORDONNIER
Examinateurs	C. CARREZ
	M. MERIAUX

P R O F E S S E U R S CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	I.E.E.A.
M. FOURET René	Physique
M. GABILLARD Robert	I.E.E.A.
M. MONTREUIL Jean	Biologie
M. PARREAU Michel	Mathématiques
M. TRIDOT Gabriel	Chimie
M. VIVIER Emile	Biologie
M. WERTHEIMER Raymond	Physique

P R O F E S S E U R S 1ère classe

M. BACCHUS Pierre	Mathématiques
M. BEAUFILS Jean-Pierre (dét.)	Chimie
M. BIAYS Pierre	G.A.S.
M. BILLARD Jean (dét.)	Physique
M. BOILLY Bénoni	Biologie
M. BOIS Pierre	Mathématiques
M. BONNELLE Jean-Pierre	Chimie
M. BOUGHON Pierre	Mathématiques
M. BOURIQUET Robert	Biologie
M. BREZINSKI Claude	I.E.E.A.
M. CELET Paul	Sciences de la Terre
M. CHAMLEY Hervé	Biologie
M. COEURE Gérard	Mathématiques
M. CORDONNIER Vincent	I.E.E.A.
M. DEBOURSE Jean-Pierre	S.E.S.
M. DYMENT Arthur	Mathématiques

PROFESSEURS 1ère classe (suite)

M. ESCAIG Bertrand	Physique
M. FAURE Robert	Mathématiques
M. FOCT Jacques	Chimie
M. GRANELLE Jean-Jacques	S.E.S.
M. GRUSON Laurent	Mathématiques
M. GUILLAUME Jean	Biologie
M. HECTOR Joseph	Mathématiques
M. LABLACHE COMBIER Alain	Chimie
M. LACOSTE Louis	Biologie
M. LAVEINE Jean Pierre	Sciences de la Terre
M. LEHMANN Daniel	Mathématiques
Mme LENOBLE Jacqueline	Physique
M. LHOMME Jean	Chimie
M. LOMBARD Jacques	S.E.S.
M. LOUCHEUX Claude	Chimie
M. LUCQUIN Michel	Chimie
M. MIGEON Michel Recteur à Grenoble	E.U.D.I.L.
M. MIGNOT Fulbert (dét.)	Mathématiques
M. PAQUET Jacques	Sciences de la Terre
M. PROUVOST Jean	Sciences de la Terre
M. ROUSSEAU Jean-Paul	Biologie
M. SALMER Georges	I.E.E.A.
M. SEGUIER Guy	I.E.E.A.
M. SIMON Michel	S.E.S.
M. STANKIEWICZ François	S.E.S.
M. TILLIEU Jacques	Physique
M. VIDAL Pierre	I.E.E.A.
M. ZEYTOUNIAN Radyadour	Mathématiques

P R O F E S S E U R S 2ème classe

M. ANTOINE Philippe	Mathématiques (Calais)
M. BART André	Biologie
Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mathématiques
M. BELLET Jean	Physique
M. BERZIN Robert	Mathématiques
M. BKOUCHE Rudolphe	Mathématiques
M. BODARD Marcel	Biologie
M. BOSQ Denis	Mathématiques
M. BRASSELET Jean-Paul	Mathématiques
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie
M. CARREZ Christian	I.E.E.A.
M. CAYATTE Jean-Louis	S.E.S.
M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean-Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
M. CROSNIER Yves	I.E.E.A.
M. CURGY Jean-Jacques	Biologie
Mle DACHARRY Monique	Géographie
M. DAUCHET Max	I.E.E.A.
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DELORME Robert	S.E.S.
M. DE MASSON D'AUTUME Antoine	S.E.S.
M. DEMUNTER Paul	C.U.E.E.P.

PROFESSEURS 2ème classe (Suite 1)

M. DENEL Jacques	I.E.E.A.
M. DE PARIS Jean-Claude	Mathématiques (Calais)
Mle DESSAUX Odile	Chimie
M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
Mme DHAINAUT Nicole	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean-Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique (Calais)
M. DUBUS Jean-Paul	I.E.E.A.
M. FAKIR Sabah	Mathématiques
M. FONTAINE Hubert	Physique
M. FOUQUART Yves	Physique
M. FRONTIER Serge	Biologie
M. GAMBLIN André	G.A.S.
M. GLORIEUX Pierre	Physique
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gabriel (dét.)	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREGORY Pierre	I.P.A.
M. GREMY Jean-Paul	S.E.S.
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HENRY Jean-Pierre	E.U.D.I.L.
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JEAN Raymond	Biologie
M. JOFFRE Patrick	I.P.A.

PROFESSEURS 2ème classe (suite 2)

M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LANGRAND Claude	Mathématiques
M. LATTEUX Michel	I.E.E.A.
Mme LECLERCQ Ginette	Chimie
M. LEFEVRE Christian	Sciences de la Terre
Mle LEGRAND Denise	Mathématiques
Mle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques
M. LEMAIRE Jean	Physique
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	C.U.E.E.P.
M. LOUAGE Francis(dét.)	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
M. MESSELYN Jean	Physique
M. MESSERLIN Patrick	S.E.S.
M. MONTEL Marc	Physique
Mme MOUNIER Yvonne	Biologie
M. PARSY Fernand	Mathématiques
Mle PAUPARDIN Colette	Biologie
M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L.
M. RACZY Ladislas	I.E.E.A.
M. RAOULT Jean François	Sciences de la Terre
M. RICHARD Alain	Biologie

PROFESSEURS 2ème Classe (suite 3)

M. RIETSCH François	E.U.D.I.L.
M. ROBINET Jean-Claude	E.U.D.I.L.
M. ROGALSKI Marc	Mathématiques
M. ROY Jean-Claude	Biologie
M. SCHAMPS Joël	Physique
Mme SCHWARZBACH Yvette	Mathématiques
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STAROSWIECKI Marcel	E.U.D.I.L.
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole
Mme TJOTTA Jacqueline (dét.)	Mathématiques
M. TOULOTTE Jean-Marc	I.E.E.A.
M. TURRELL Georges	Chimie
M. VANDORPE Bernard	E.U.D.I.L.
M. VAST Pierre	Chimie
M. VERBERT André	Biologie
M. VERNET Philippe	Biologie
M. WALLART Francis	Chimie
M. WARTEL Michel	Chimie
M. WATERLOT Michel	Sciences de la Terre
Mme ZINN JUSTIN Nicole	Mathématiques

CHARGES DE COURS

M. ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M. BAFCOP Joël	I.P.A.
M. DUVEAU Jacques	S.E.S.
M. HOFLACK Jean	I.P.A.
M. LATOUCHE Serge	S.E.S.
M. MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M. NAVARRE Christian	I.P.A.
M. OPIGEZ Philippe	S.E.S.

A ma femme, Cathy

A mes parents.

REMERCIEMENTS

Je tiens à remercier :

- *Monsieur le Professeur Max DAUCHET d'avoir bien voulu me faire l'honneur de présider le Jury de cette Thèse.*

- *Monsieur le Professeur Vincent CORDONNIER d'avoir accepté de diriger ce travail de recherche, et de m'avoir fourni par ses critiques matière à reflexion.*

- *Monsieur Michel MERIAUX, Chargé de Recherche au CNRS, pour l'aide qu'il a pu m'apporter et de la disponibilité dont il a fait preuve à mon égard.*

- *Monsieur le Professeur Christian CARREZ qui m'a fait l'honneur de participer à ce Jury.*

Je remercie également :

- *Madame Bénédicte VANDROEMME pour la mise en forme et la dactylographie de ces pages.*

- *Monsieur Henri GLANC qui a assuré la réalisation matérielle de cet ouvrage.*

SOMMAIRE

INTRODUCTION

La synthèse d'images, comme l'ensemble des domaines couverts par l'informatique graphique, demande des temps de traitement prohibitifs non conforme à l'interactivité et au temps réel. Par exemple, l'élaboration d'une image composée de 512×512 points demande, à raison d'un traitement élémentaire de 100 µs par point, plus de 26 secondes.

C'est pourquoi, depuis quelques années sont apparues dans les laboratoires de recherche publics et privés des architectures parallèles spécialisées en traitements graphiques.

Cette pléthore de machines, souvent disparates, pose le problème de leur adaptation particulière à la synthèse d'images et de la caducité des logiciels existants, car conçus pour des machines de type Von Neumann et inexploitables sur des architectures parallèles.

Cette thèse poursuit deux buts indissociables :

- D'une part, fournir les bases d'un logiciel nécessaires à une architecture parallèle.

- D'autre part, du fait de l'étude d'algorithmes propres à la synthèse, effectuer un début d'analyse descendante propre à déterminer des constantes et buts architecturaux.

Une nouvelle approche géométrique utilisant un découpage récursif du plan ou de l'espace est ensuite élaborée pour permettre par son utilisation la simplification des problèmes graphiques.

CHAPITRE I

GENERALITES

CHAPITRE I : GENERALITES

I.1. LA SYNTHESE D'IMAGES

De nos jours, quel que soit le domaine concerné (X.A.O., architecture, audio-visuel, etc...), l'image en tant que support d'informations prend une place prépondérante. Ceci est dû au fait que la compréhension d'un ensemble complexe de données présenté sous forme visuelle est plus facile et permet une prise de décision plus rapide ; tel est le cas, par exemple, d'un graphique de chiffres de ventes ou de la visualisation en perspective d'une maison en architecture, par rapport à une liste de nombre d'unités vendues ou de mesures.

Le domaine de l'informatique qui permet de passer d'un ensemble de données d'une application à une représentation visuelle est la synthèse d'images. [LUC 77], [LUC 84], [FOL 82], [MAR 82].

I.1.1. Structure logique

Pour passer des données d'une application à une image, plusieurs étapes de traitements sont nécessaires :

* Tout d'abord, il faut extraire des données de l'application les informations nécessaires à une description graphique et effectuer un codage de ces différents éléments pour obtenir la scène de représentation.

* Il faut alors à partir de la scène composer une scène bidimensionnelle pour préparer la visualisation sur l'écran.

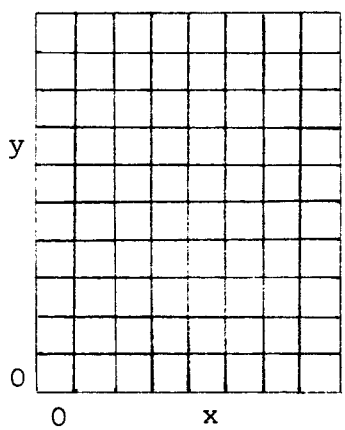
* Ensuite, à partir d'algorithmes élémentaires, on traite les différentes indications fournies par la scène bidimensionnelle pour obtenir l'image finale sous forme d'une liste de visualisation ou de valeurs en mémoire d'image en vue d'assurer la gestion de l'écran ou du dispositif d'affichage.

I.1.2. La mémoire d'images

Pour assurer la permanence et la régénération d'une image sur un moniteur spécialisé ou sur un téléviseur standard, on utilise le plus couramment une mémoire d'images.

La mémoire d'images est la représentation matricielle de l'écran sous forme de mots mémoires, où chaque mot définit la valeur numérique ou couleur à affecter à une surface élémentaire de l'écran appelée pixel.

Ce quadrillage de l'écran permet l'accessibilité des pixels à l'aide d'un couple de coordonnées (x, y). Par convention, l'origine est placée en bas de l'écran à gauche.



Le nombre de pixels définit la précision de l'écran, c'est-à-dire le nombre de surfaces élémentaires accessibles technologiquement. Les définitions les plus courantes actuellement pour des applications professionnelles sont de 512×512 ou 1024×1024 pixels.

La couleur d'un pixel est définie par une valeur numérique qui exprime le mélange des trois composantes additives : rouge, vert et bleu.

I.1.3. L'accroissement des besoins

A son origine, la synthèse d'images produisait des dessins au trait, c'est-à-dire des graphiques à base de lignes, mais très vite, la nécessité d'obtenir des images plus figuratives ou plus réalistes a exigé des traitements plus évolués et des techniques plus complexes. L'accroissement des temps

de calcul, qui en a résulté, pénalise l'interactivité et la production graphique en empêchant les traitements en temps réel.

Malgré les progrès de la technologie et l'optimisation des logiciels, la synthèse d'une image présentant des aspects réalistes nécessite parfois plusieurs dizaines de minutes.

Pour pallier cette carence, les architectes et concepteurs d'ordinateurs se sont alors tournés vers le parallélisme pour produire des machines spécialisées en informatique graphique. [ALL 83], [BAS 84], [FAT 83], [FLY 72], [GAI 84], [GOL 83], [PAI 79], [STE 83], [WAR 82].

I.2. LES DIFFERENTES ARCHITECTURES PARALLELES GRAPHIQUES

On peut schématiser une machine parallèle graphique en donnant les composantes fonctionnelles suivantes :

- Un espace de départ.
- Un ensemble de processeurs élémentaires chargés d'effectuer un ensemble d'actions.
- Un espace d'arrivée.

Les deux espaces correspondent à :

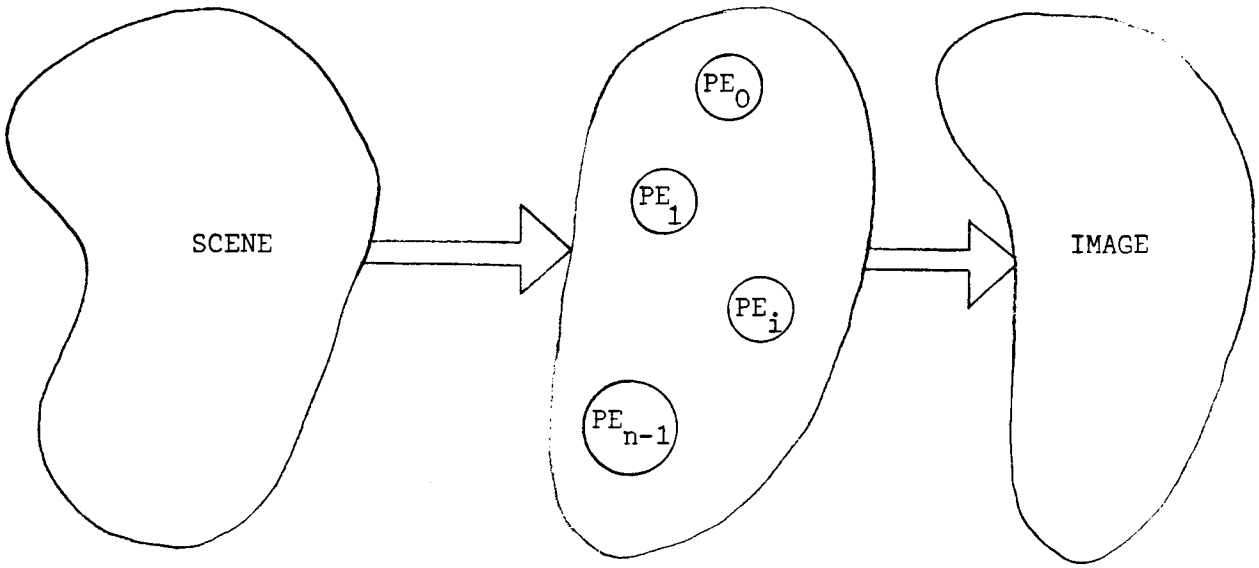
- D'une part, à une structure de données constituant la scène et les objets qui la composent ; donnant ainsi une représentation logique de l'image.

- D'autre part, à l'ensemble des pixels définissant l'image.

Suivant la nature de l'application graphique, on aura en général :

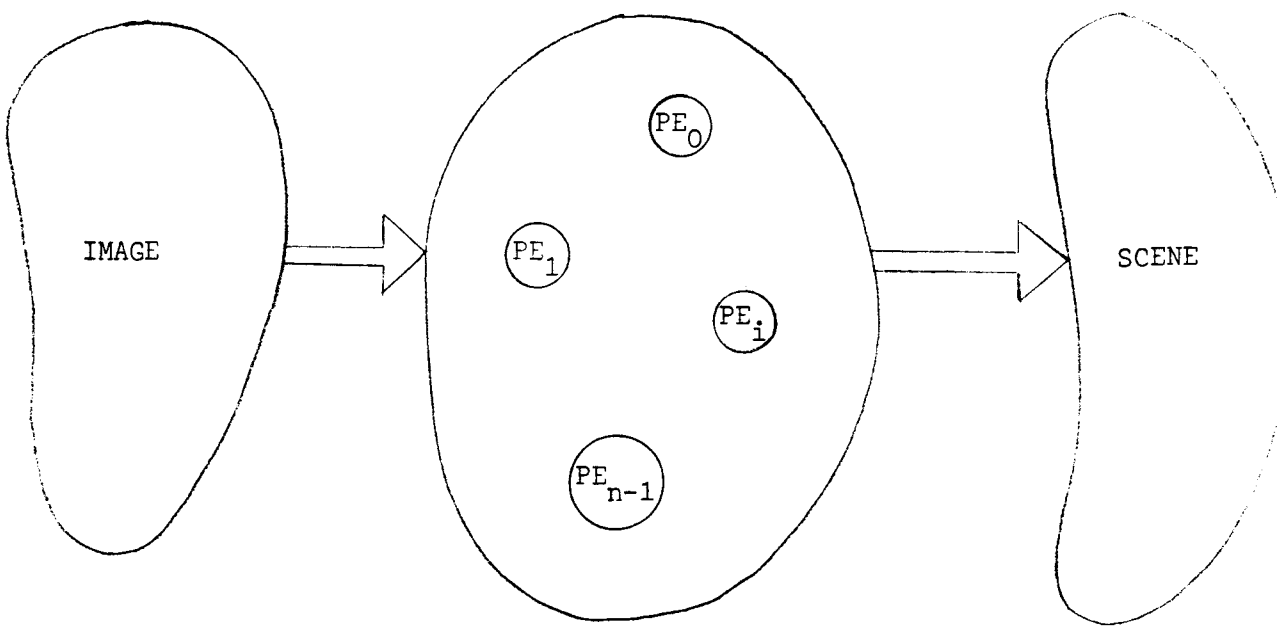
• en synthèse d'images

La scène comme espace de départ et l'image comme espace d'arrivée.



• en traitement d'images

Une image comme espace de départ et une scène comme espace d'arrivée.



On peut distinguer deux techniques principales pour exploiter le parallélisme :

- Soit partitionner l'un des deux espaces (le partitionnement des deux espaces à la fois étant logiquement incohérent).

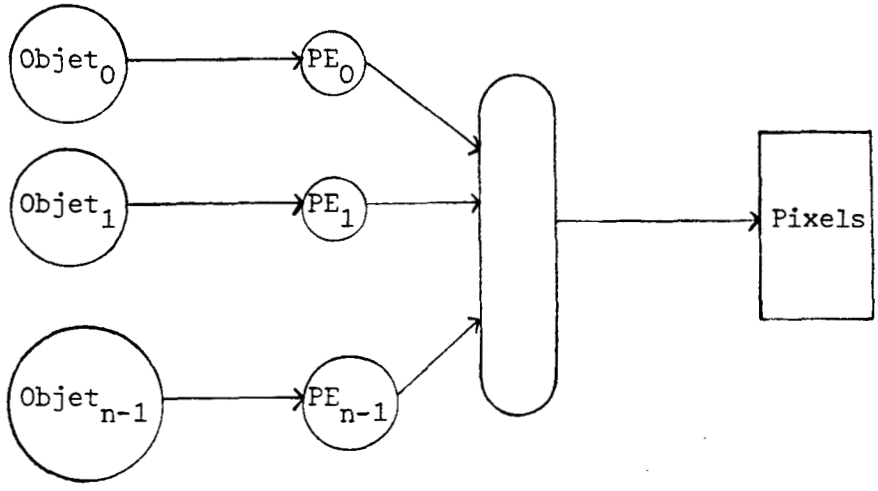
- Soit partitionner l'ensemble des actions.

I.2.1. Partition de l'espace de départ

Ce type de partition présente l'inconvénient de spécialiser la machine pour une application graphique suivant l'espace pris en considération.

I.2.1.1. Partition de la scène

Cette partition est effectuée en associant à chaque processeur élémentaire un objet (segment, cercle, sphère, etc...) de la scène sur lequel il effectue alors tous les traitements.

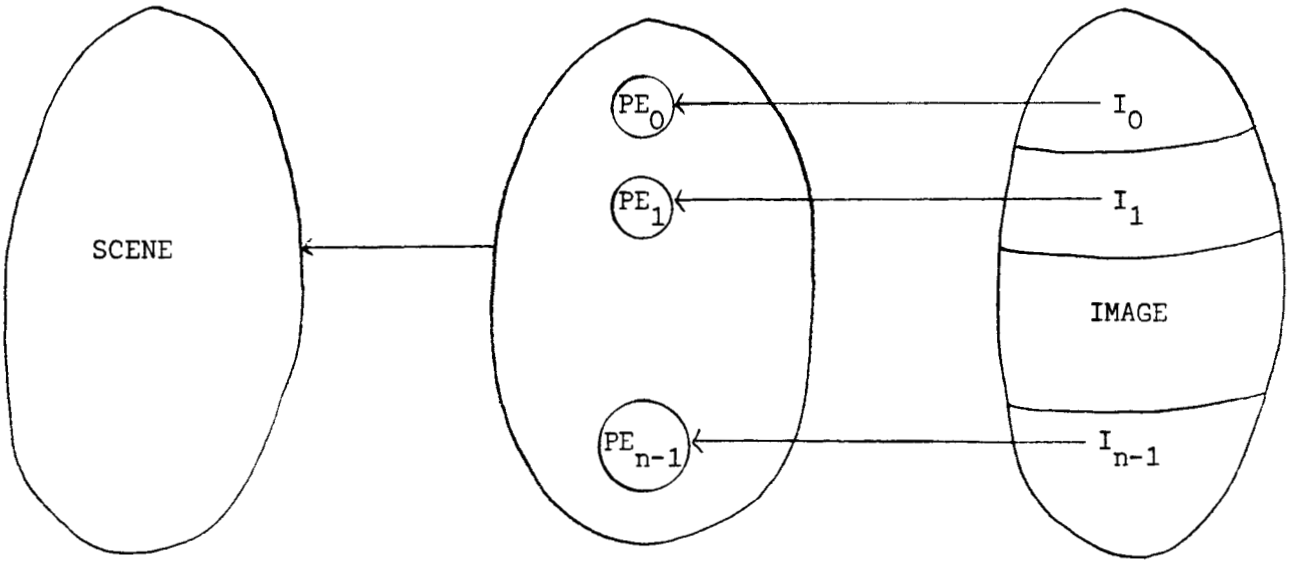


On parle alors de machines objets [LER 80], [WEIN 81], [DUR 82 et 83], on peut citer comme exemple la machine Artémis [GRU 81].

Un tel type de machine est spécialisée en synthèse d'images.

I.2.1.2. Partition de l'image

On attribue à chaque processeur une zone géométrique de l'image, celui-ci effectue alors toutes les actions relatives à cette zone.



La distribution ultime étant d'attribuer un processeur par pixel de l'image.

Relativement à cette partition on parle de machine tableau ou cellulaire.

Comme exemples on peut citer :

GLOPR, DIFF 3, PICAP, CIP, MPP,...

[DAN 81], [KID 83], [HWA 83], [PLOT 83], [REE 81], [SIEG 82], [YAN 83].

Les machines tableau ont été surtout étudiées pour le traitement d'images.

I.2.2. Partition de l'espace d'arrivée

L'utilisation d'une machine cellulaire semble être intéressante en synthèse d'images, toutefois, les possibilités d'une telle machine, du fait de la simplicité des processeurs élémentaires semblent être limitées pour cette application à des traitements simples (segments, cercles, remplissage de contours préinscrits) [STA 75], [MER 83 et 84], [PRE 83], [ROS 83], [STE 83].

Contrairement à une machine cellulaire, une machine objet ne possède aucune ambivalence.

I.2.3. Partition sur l'ensemble des actions

Etant donné un ensemble d'actions ou de traitements à effectuer on peut réaliser deux répartitions différentes :

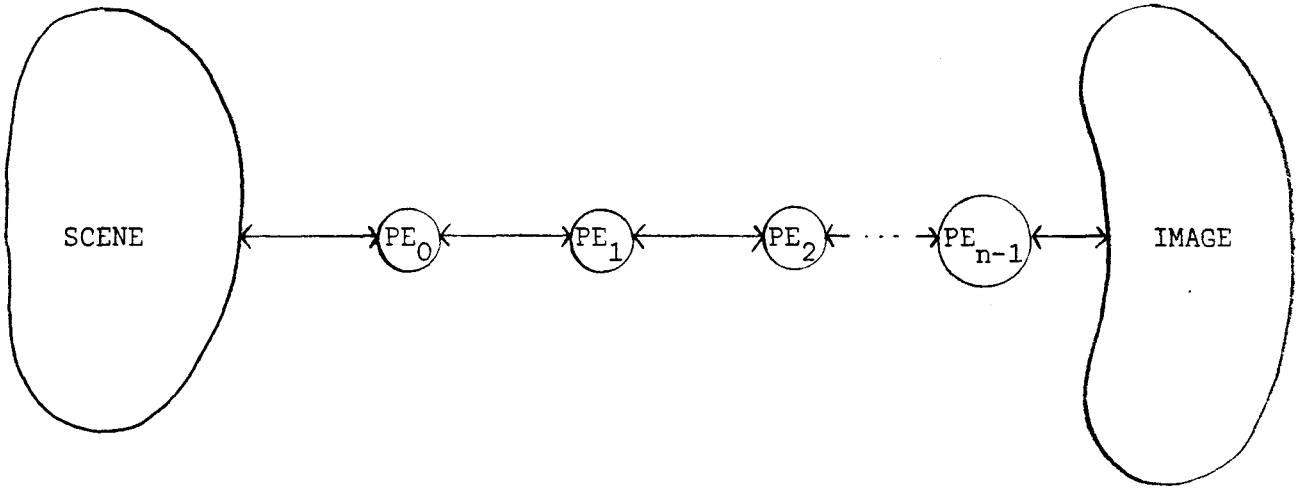
- Soit attribuer à chaque processeur élémentaire une action à réaliser.

- Soit répartir chaque traitement ou action sur l'ensemble des processeurs élémentaires.

I.2.3.1. Répartition bijective (action-processeur)

Chaque processeur est spécialisé dans la réalisation d'une action. Le traitement global est alors réalisé par la coopération de l'ensemble des processeurs.

Les actions étant généralement dépendantes dans le temps entre elles, une telle répartition conduit à une machine du type pipe-line [ABD 83].



Du fait des commodités qu'elle procure : problèmes de communication simples à résoudre, un seul flot de données, cette structure est la plus répandue. Elle présente toutefois l'inconvénient d'une grande rigidité rendant les extensions difficiles et d'une cadence de calcul déterminée par l'action la plus complexe.

Toutefois, l'utilisation de processeurs banalisés et de files d'attentes, permet à cette architecture une utilisation aussi bien en synthèse qu'en traitement d'images.

I.2.3.2. Répartition totale de chaque action

L'application graphique est décomposée en primitives ou actions nécessaires à l'utilisateur.

Le parallélisme est appliqué au niveau de chaque action, l'ensemble des processeurs coopérent alors à sa réalisation.

Ce type de répartition permet la transparence du parallélisme lors de la programmation de l'application et conduit à des machines utilisables en synthèse ou en traitement d'images. (On pourrait classer dans ce groupe le système complexe de microprocesseurs asynchrones développé à l'Université de Kioto [KID 83]).

Son adaptabilité, du fait de l'utilisation de processeurs banalisés et de l'absence de contraintes spatiales en fait un modèle pouvant implémenter plusieurs niveaux de parallélisme.

I.3. MACHINE SUPPORT

L'apparente anarchie de conception présente dans les architectures parallèles à but graphique résulte d'une conception restreinte vers un type particulier de traitements.

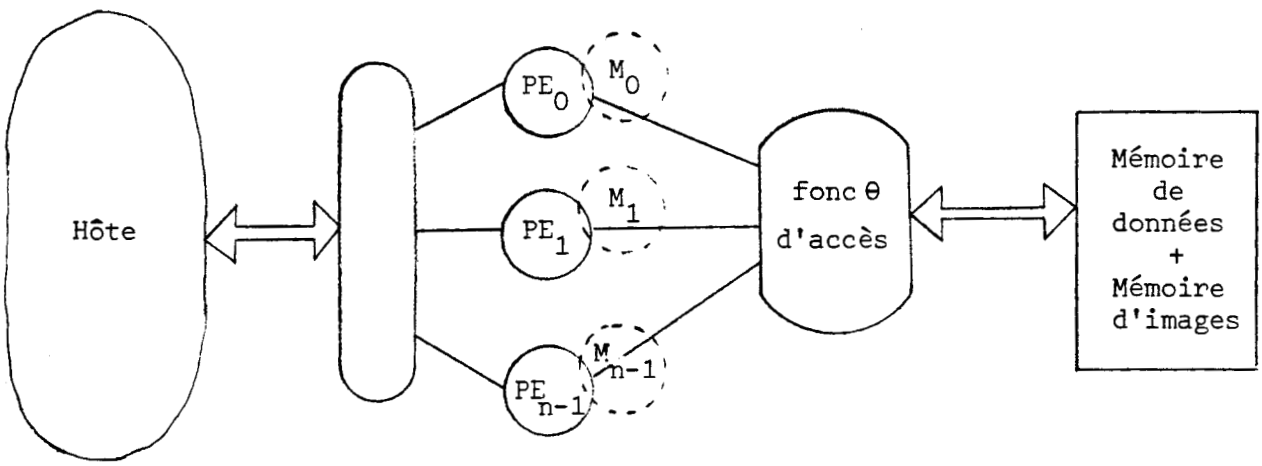
Cette méthololologie de conception peut conduire à des résultats parfois désastreux lorsque l'on cherche à étendre le champ d'application d'une telle machine.

L'approche idéale consiste, comme l'a bien montré Danielson [DAN 84] à effectuer une analyse descendante de chaque application graphique en vue de déterminer une architecture optimale pour tous les traitements de l'imagerie informatique ; c'est-à-dire, tirer les conséquences architecturales des divers algorithmes utilisés en vue de déterminer le meilleur compromis possible entre idéaux et technologie.

C'est l'approche qui a prévalu pour ce travail sur la synthèse d'images sur machine parallèle.

L'architecture est choisie la plus générale possible :

- Un ordinateur hôte.
- Un ensemble de processeurs élémentaires avec ou sans mémoire de programme et unité de contrôle (MIMD ou SIMD).
- Des fonctions d'accès non définies pour l'instant à la mémoire de données et à la mémoire d'images.



Le choix du parallélisme appliqué s'est porté sur la répartition d'une action sur l'ensemble des processeurs, permettant l'intégration de par sa généralité à d'autres niveaux de parallélisme (machine-objet, pipe-line, cellulaire).

L'étude réalisée s'est donc portée sur la parallélisation des algorithmes de synthèse et l'examen de ceux-ci en vue de déterminer les fonctions d'accès les moins contraignantes possibles pour conserver une pluralité de fonctions.

CHAPITRE II

ALGORITHMES DE BASE

Dans ce chapitre, nous examinons différents algorithmes élémentaires parallèles nécessaires dans toute application visant la synthèse d'images.

Dans un premier temps est abordé le tracé de segment qui constitue la pierre d'achoppement d'un logiciel parallèle graphique performant. En effet, de celui-ci, dépend le plus souvent l'une des étapes les plus complexes de la synthèse d'images consistant à supprimer, pour la visualisation, les faces cachées des objets relativement à un observateur.

Le tracé de cercle est ensuite étudié, montrant ainsi un bon exemple de la difficulté de réaliser certains algorithmes de synthèse de manière totalement parallèle et performante sans utiliser d'opérations complexes.

La partie suivante concerne la réalisation parallèle du remplissage de contours préinscrits et la dernière partie se rapporte à la parallélisation d'algorithmes de surfaces cachées par analyse de ligne.

Ces exemples ne représentent pas une étude exhaustive, mais caractérisent quatres familles d'opérateurs que l'on peut classer par rapport aux données manipulées.

1 - Suites contigües de points (1 D)
2 - Objets définis par leurs paramètres géométriques
3 - Surfaces (2 D)
4 - Volumes (3 D)

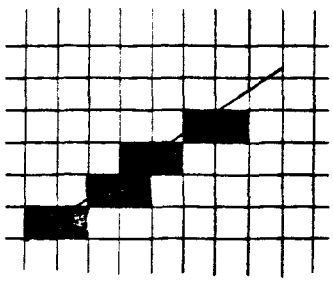
CHAPITRE II : ALGORITHMES DE BASE

II.1. LE TRACE DE SEGMENT

II.1.1. Introduction

La recherche d'un algorithme performant de tracé de segment est d'une grande importance pour la réalisation d'un logiciel graphique, car la représentation d'un segment dans un espace et par projection sur un plan discret est non seulement indispensable pour la réalisation de dessins au trait mais aussi lors de l'affichage d'objets modélisés par des facettes polygonales. En effet, cet algorithme est alors utilisé pour déterminer les intersections entre les lignes de la mémoire d'image et les objets contenus dans la scène définissant ainsi des pans horizontaux d'affichage.

Un segment de droite est défini par la donnée des coordonnées de ses extrémités. Le problème consiste alors à engendrer en parallèle la suite de points situés sur la grille d'écran approchant le mieux le segment initial.



Principe [MORV 76]

Les coordonnées écran étant entières, les calculs se font en entier afin d'éviter des conversions coûteuses. Une méthode donnant de bons résultats est la suivante :

On choisit de faire progresser d'une unité écran (en valeur absolue) à chaque pas la coordonnée x ou y suivant que la composante en x est plus grande que la composante en y (ou vice-versa). Ceci assure que l'on va afficher un maximum de points. La progression de l'autre coordonnée (y ou x) se fera dans les conditions suivantes : on prend une variable initialisée à zéro au début du tracé. Chaque fois que l'on progresse d'une unité sur la coordonnée de référence, on ajoute à cette variable la composante la plus petite (en valeur absolue). Cette somme se fait modulo la plus grande composante (en valeur absolue).

Chaque fois que la somme dépasse la valeur de la plus grande composante, on met à jour d'une unité (en valeur absolue) la coordonnée correspondant à la plus petite composante. Cette méthode fait donc progresser sur un axe systématiquement d'une unité et sur l'autre axe d'une unité également, mais avec une fréquence dépendant de la pente de la droite à tracer. On obtient ainsi des segments en "escalier" dont l'aspect inesthétique peut être amélioré par des traitements d'anti-aliasing.

L'algorithme le plus courant est celui de Bresenham [BRE 65] on pourra aussi se référer à [EAR 80], [LOC 80], [PIT 82].

Algorithme séquentiel (suivant le principe développé dans [BRE 65])

Notations

- La fonction SGN donne comme résultat +1, 0, ou -1 suivant le signe de l'argument.
- La fonction ABS donne comme résultat la valeur absolue de l'argument.
- La fonction DIV effectue la division entière.

```
Procédure GENSEG (Xa, Ya, Xb, Yb : entiers)

    var : U, V, D1X, D1Y, M, N, D2X, D2Y, S : entiers.

début
    U := Xb - Xa ; V = Yb - Ya ;
    D1X := SGN(U) ; D1Y := SGN(V) ;
    /* (D1X, D1Y) représente un pas dans la direction diagonale */
    M := ABS(U) : N := ABS(V)
  Si M ≤ N alors
    D2X := 0      /* (D2X, D2Y) pas vers le haut ou le bas */
    D2Y := SGN(V)
    échange (M, N)
  Sinon
    D2X := SGN(U)      /* pas à gauche ou à droite */
    D2Y := 0
  Fsi
  M est le plus grand de ABS(U) et ABS(V), et N est le plus petit.
```

```
S := M DIV 2
X := Xa
Y := Ya

Pour I := 0 à M
          Affichage (X, Y)
          S := S + N
          Si S ≥ M Alors
                S := S - M
                X := X + D1X
                Y := Y + D1Y        /* Pas diagonal */

          Sinon
                X := X + D2X
                Y := Y + D2Y     /* Pas vers le haut-bas ou droite-gauche */

          Fsi

Fin Pour

Fin GENSEG
```

II.1.2. Cadre général

II.1.2.1. Préliminaires

Le parallélisme des différents algorithmes présentés est comparé pour une phase du traitement (calcul et affichage) à l'aide de deux coefficients :

$$C = \frac{\text{nombre de processeurs calculant 1 point}}{\text{nombre de processeurs élémentaires}}$$

$$A = \frac{\text{nombre de processeurs affichant 1 point}}{\text{nombre de processeurs élémentaires}}$$

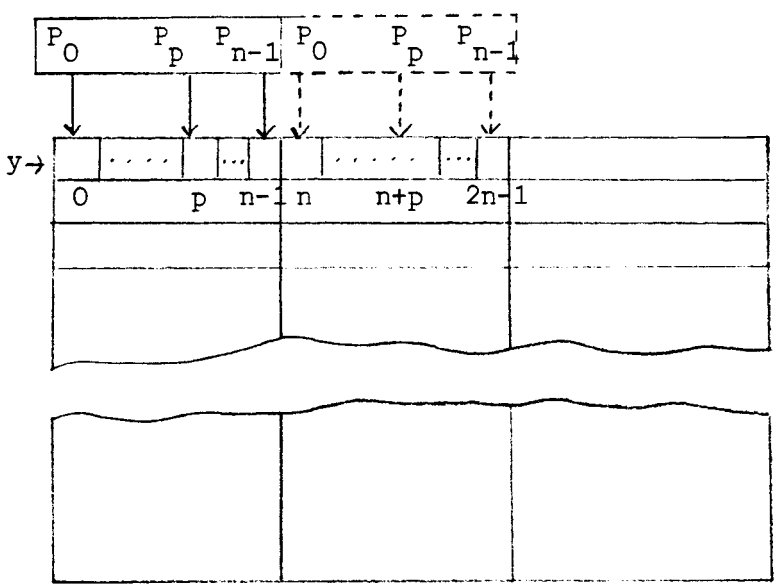
L'idéal étant d'obtenir le parallélisme maximal de calcul et d'affichage, soit respectivement C et A égaux à 1.

Deux approches sont possibles, on peut chercher à privilégier d'abord le parallélisme d'affichage (cas II.1.2.2. et II.1.2.3.) ou au contraire privilégier le parallélisme de calcul (cas II.1.2.4. et II.1.2.5.).

II.1.2.2. Algorithme réparti

L'utilisation parallèle la plus simple et la moins efficace d'un algorithme, consiste à faire exécuter le même algorithme séquentiel par chaque processeur avec un test permettant de déterminer si le point est dans sa région d'affichage.

Exemple : Si l'architecture de N processeurs a un découpage en sous-ligne, c'est-à-dire permettant à chaque processeur d'afficher les points pour une ordonnée y, d'abscisses x : P, N + P, 2N + P,... (P : numéro du processeur).



Il faut que chacun des N processeurs teste l'apparence du point calculé à sa région d'affichage ; l'étape de calcul i se terminant lorsque

x = (N-1) * i ou y = y ± 1, on affiche alors la sous-ligne, et ainsi de suite.

Dans cet exemple, le parallélisme d'affichage n'est intéressant que si dx est supérieur ou égal à dy.
(dx, dy : différences des coordonnées extrêmes).

A = si dx ≥ dy alors dy/dx sinon 1/N.

Quelle que soit l'organisation des accès, le parallélisme de calcul sera toujours égal à 1/N : il n'y a pas de parallélisme de calcul car chaque processeur traite tous les points.

II.1.2.3. (n, n) algorithme (cf. [SPR 82])

R.F. Sproull utilise une transformation intéressante de l'algorithme de Brésenham permettant à chaque processeur P : (0..N-1) de calculer et d'afficher les points d'abscisses : x = P, N+P, 2N+P,...

Cet algorithme répond à une organisation des accès telle qu'une colonne de la mémoire d'image soit adressée par un seul processeur.

La forme la plus générale de cet algorithme est la suivante : (on pourra consulter [SPR 82] pour la version transformée n'effectuant que des opérations simples en entiers).

```
(n, n) algorithme

début          /* l'une des extrémités du segment est considérée
                  comme origine */

     var : P : entier

Pour Processeur P : 0 à N-1
     var x_i, y_i : entiers, yt : réel
     Pour x_i = 0 + P à dx pas de n
          yt := (dy/dx)  x_i ;
          y_i := tronc(y_t + 0.5) ;
          Affichage(x_i, y_i)
     Fin Pour
Fin Processeur

fin
```

On obtient les coefficients :

C = A = si dx ≥ dy alors 1 sinon dx/dy.

II.1.2.4. Algorithme dichotomique

Principe

Si a : (x_a, y_a) et b : (x_b, y_b) sont les deux extrémités d'un segment S ; le point milieu M : $(x_a + x_b/2, y_a + y_b/2)$ appartient à S.

Le même raisonnement pour les sous-segments (a, m) et (m, b) nous définit 4 sous-segments de S. En poursuivant le processus, le segment S est entièrement tracé par une méthode récursive et n'utilisant que des opérations d'addition et de décalage.

Un processeur P calcule le milieu d'un sous-segment et envoi cette valeur à un autre processeur ainsi que l'une des deux extrémités. La variable locale I sert à adresser le processeur récepteur.

Primitives

Pour cet algorithme, des primitives de synchronisation et de communication sont nécessaires.

Communication :

- envoi (numéro processeur, paramètres) : envoi de paramètres au processeur désigné dans la primitive,

- réception (paramètres) : réception de paramètres en provenance d'un autre processeur.

Synchronisation :

- synchro (numéro processeur) : envoi d'un signal de synchronisation au processeur désigné,

- attente synchro : une fois cette primitive exécutée, le processeur attend un signal de synchronisation pour reprendre son traitement.

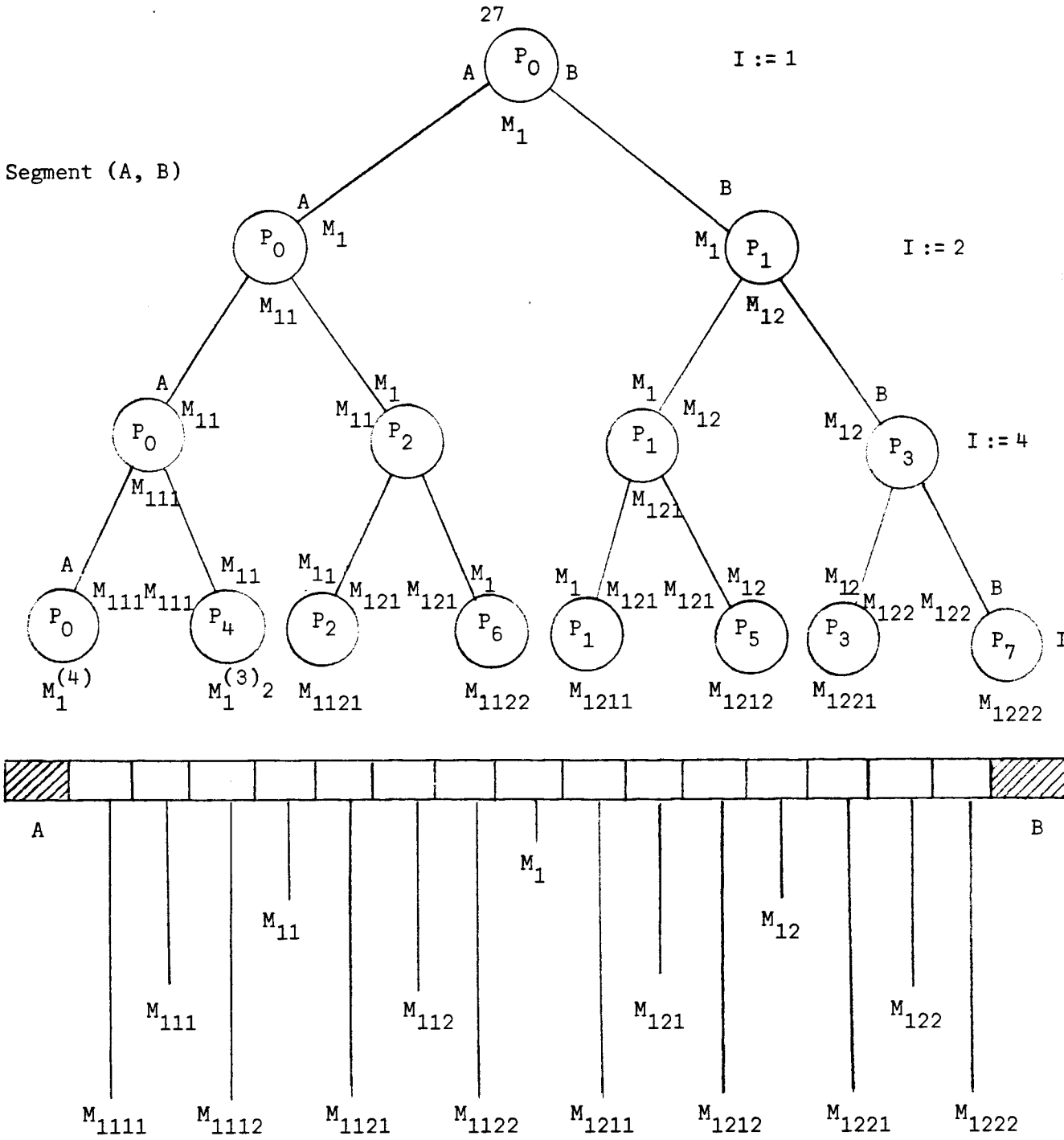
Algorithme :

Procédure dicho : (x_d, y_d, x_f, y_f : réels)

```
début
      tracé des extrémités (x_d, y_d), (x_f, y_f) ;
      Pour Processeur P = 0 à N-1
           Si P = 0 alors
                 I := 1 ;
                 réception ((x_d, y_d), (x_f, y_f)) ;
           Sinon
                 attente synchro
                 réception ((x_d, y_d), (x_f, y_f), I)
           Fsi

           TANT QUE E(x_d) ≠ E(x_f) et E(y_d) ≠ E(y_f)
                    /* E(x) : partie entière de x arrondie */
                    x_m := x_d + x_f/2 ;
                    y_m := y_d + y_f/2 ;
                    synchro (P+I) ;
                    envoi (P+I, (x_m, y_m), (x_f, y_f), 2*I) ;
                    affichage (E(x_m), E(y_m)) ;
                    x_f := x_m ;
                    y_f := y_m ;
                    I := 2 * I ;
           Fin
      Fin Processeur

fin
```



Exemple avec 8 processeurs élémentaires

Les calculs peuvent être faits sur deux registres, l'un contenant la partie entière, l'autre la partie fractionnaire résultant des décalages.

La pente du segment n'intervient pas pour le parallélisme de calcul. Si le nombre de processeurs N est supérieur ou égal à la définition maximale d'une ligne ou d'une colonne de la mémoire d'image, on obtient :

$C = \log_2(\max(dx, dy))/N$

C, est ici le coefficient de parallélisme de calcul moyen.

Une pile est nécessaire pour mémoriser N points lorsque $\max(dx, dy) > N$ pour pouvoir poursuivre la récursivité.

Si la sortie sur mémoire d'image est physiquement ou logiquement parallèle le coefficient d'affichage A est égal à celui de calcul C, sinon, du fait de l'algorithme il sera très voisin de 1/N, car la suite de points calculée ne correspondra pas à la zone de partition correspondant à l'accès à la mémoire.

II.1.2.5. Algorithme par analyse différentielle (DDA)

Principe

La résolution du tracé de segment par cette méthode consiste à calculer les différences en x et en y entre les deux extrémités et de déterminer par rapport à la plus grande des deux en valeur absolue l'incrément en x et en y.

L'algorithme séquentiel de base est le suivant : (on trouvera dans [NEW 79] une méthode pour réaliser les calculs en nombres entiers)

Procédure DDA (x_1, y_1, x_2, y_2 : entiers)

var : longueur, i : entier
x, y, x_{incr}, y_{incr} : réels

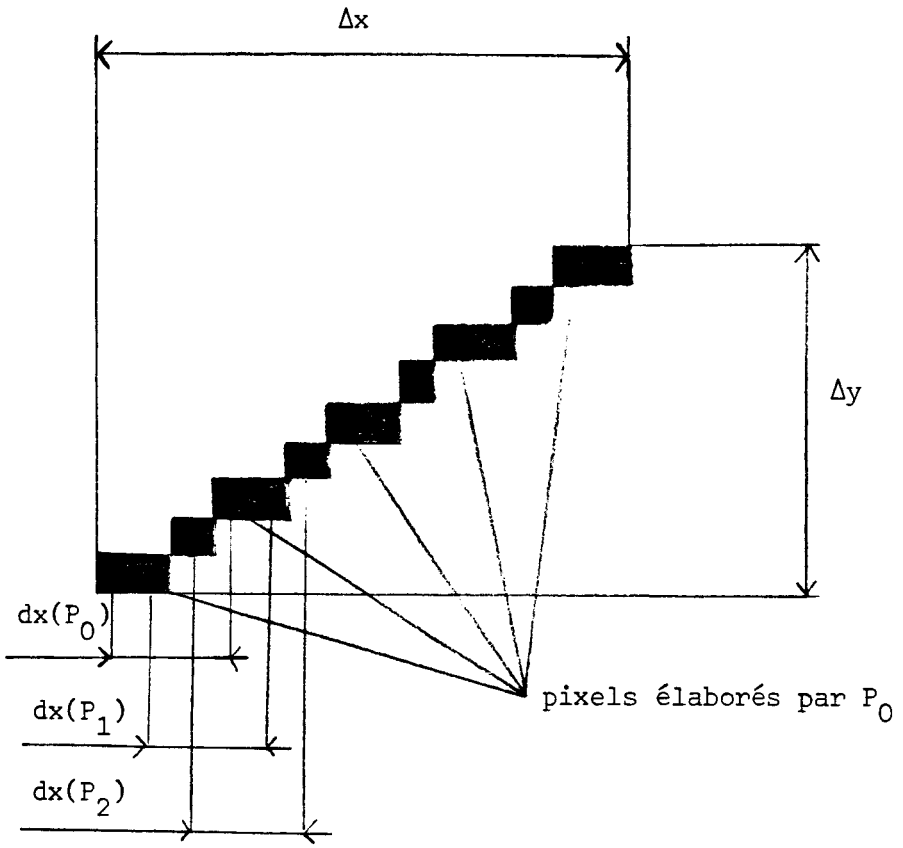
```
début
        longueur := ABS(x2 - x1) ;
        /* ABS(x) : valeur absolue de x */
        Si ABS(y2 - y1) > longueur alors
              longueur := ABS(y2 - y1) ;
        Fsi
        xincr := (x2 - x1) / longueur ;
        yincr := (y2 - y1) / longueur ;
        x := x1 + 0.5 ; y := y1 + 0.5 ;
        Pour i = 0 à longueur
             affichage (E(x), E(y)) ;
             x := xincr + x ;
             y := yincr + y ;
        Fpour
fin
```

La transformation parallèle de cet algorithme nous donne :

```
Procédure DDA//(x1, y1, x2, y2 : entiers)
        var : longueur, P, N, I : entiers
              x, y, xincr, yincr, dx, dy, pas x, pas y : réels.
```

```
début
        affichage (x1, y1)
        longueur := ABS(x1 - x2) ;
        Si ABS(y1 - y2) > longueur alors
              longueur := ABS(y2 - y1) ;
        Fsi
        xincr := (x2 - x1) / longueur ;
        yincr := (y2 - y1) / longueur ;
        x := x1 + 0.5 ; y := y1 + 0.5 ;
        pas x := xincr * N ; pas y := yincr * N ;
        Pour Processeur P = 0 à N-1
                        I := P+1 ;
                        dx := xincr * I ; dy = yincr * I ;
                        TQ I ⩽ longueur
                           x := x + dx ;
                           y := y + dy ;
                           affichage (E(x), E(y)) ;
                           dx := dx + pas x ;
                           dy := dy + pas y ;
                           I := I + N ;
                        FTQ
        Fin Processeur
fin
```

Schéma explicatif avec 3 processeurs (P_0, P_1, P_2)



Dans cet algorithme, tous les processeurs calculent au même moment un point du segment, d'où C = 1.

Si les accès à la mémoire d'image sont parallèles A est égal à 1. Sinon l'algorithme nécessite de légères modifications pour avoir un coefficient A meilleur que 1/N.

II.1.2.6. Récapitulatif et comparaisons

Algorithmes	C	A_{max} théorique	type
Algorithme réparti	1/N	Si dy ≤ dx : dy/dx Sinon 1/N	micro-SIMD
(n, n)	Si dx ⩾ dy : 1 Sinon dx/dy	C	micro-SIMD
dichotomique	$\frac{Log_2(max(dx, dy)}{N}$ *	C	macro-SIMD
DDA//	1	1	macro-SIMD

* N étant ≥ à la définition d'une ligne ou d'une colonne de la mémoire d'images.

On peut distinguer trois modes de fonctionnement :

micro-SIMD : Au cours de tout le traitement chaque processeur exécute la même instruction.

macro-SIMD : Chaque processeur exécute le même programme, mais un (ou plusieurs processeurs) peut exécuter une séquence d'instructions différente de celle des autres.

MIMD : Chaque processeur exécute un programme distinct de celui des autres.

Les meilleurs algorithmes de tracé de segment sont donc le (n, n) algorithme avec développement en y et le DDA//.

La complexité de l'étape de calcul est la même, mais l'implémentation du DDA// est plus simple à réaliser sur des processeurs élémentaires.

II.1.3. Tracé de segment sur machine cellulaire

II.1.3.1. Algorithme à propagation

Soit un segment S(A, B), la droite D supportant ce segment a pour équation :

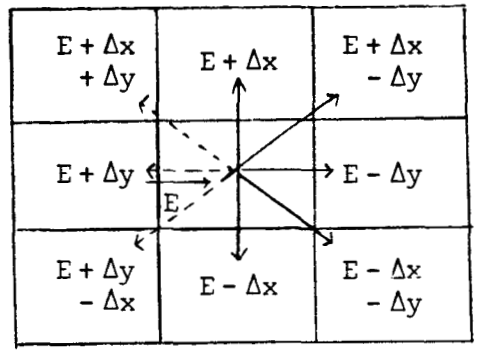
$$D : (-\Delta y)x + (\Delta x)y + c = 0$$

avec

$$\Delta x = x_B - x_A$$
$$\Delta y = y_B - y_A$$

Pour chaque cellule (x, y) on détermine sa valeur E par rapport à l'équation de D et l'on propage celle-ci aux 4 ou 8 cellules voisines.

On commence aux extrémités A et B avec la valeur E = 0 et l'on propage celle-ci de la manière suivante :



La valeur de seuil déterminant l'affichage est :

$$\max(|\Delta x|, |\Delta y|)/2$$

La comparaison avec le terme E s'effectue en valeur absolue.

- avec anti-aliasing

On peut se servir des valeurs calculées pour déterminer l'intensité de chaque pixel.

Le seuil S est alors

$$\max(|\Delta x|, |\Delta y|)$$

Si I_d est l'intensité affectée au segment, l'intensité I du pixel se calcule de la manière suivante :

$$I(x, y) = I_d * \frac{S - |E|}{S}.$$

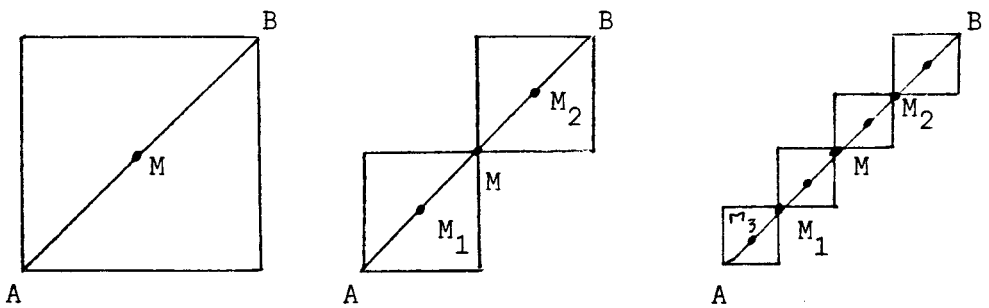
Le temps d'exécution de cet algorithme est proportionnel à $\frac{n}{2}$, n étant le nombre de points à afficher.

Pour cette approche on pourra consulter aussi [MER 83], [JOR 74].

II.1.3.2. Algorithme dichotomique

Principe

L'ensemble des cellules comprises dans le rectangle $((X_A, Y_A), (X_B, Y_A), (X_B, Y_B), (X_A, Y_B))$ calcule le point milieu M. La cellule approchant M affiche le point. Le traitement suivant est effectué pour les deux rectangles $((X_A, Y_A), (X_M, Y_A), (X_M, Y_M), (X_A, Y_M))$ et $((X_M, Y_M), (X_B, Y_M), (X_B, Y_B), (X_M, Y_B))$ nous définissant deux nouveaux points et 4 rectangles de traitement. Le processus se poursuit ensuite de la même manière.



étape 1 étape 2 étape 3

algorithme :

cellule (i, j)

var : x_A, y_A, x_B, y_B, x_m, y_m : entiers

fin : booléen.

début

fin := faux ;
réception (x_A, y_A, x_B, y_B) ;
Si (i = x_A et j = y_A) ou (i = x_B et j = y_B) alors
Affichage ;
fin := vrai ;
Fsi

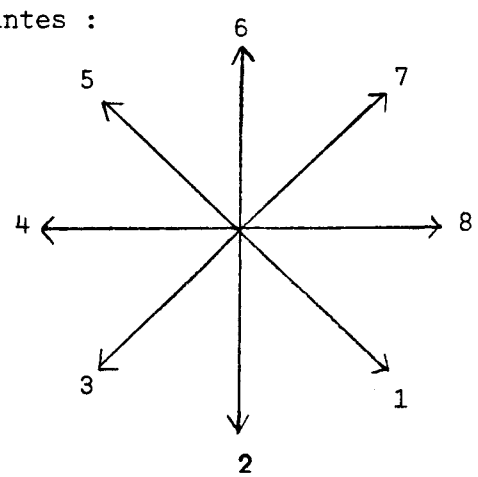
```
TQ fin = faux ;
   fin := vrai ;
   x_m := E((x_A + x_B)/2) ;
   y_m := E((y_A + y_B)/2) ;
   Si ((x_A ≤ i ≤ x_m) ou (x_A ≥ i ≥ x_m)) et
      ((y_A ≤ j ≤ y_m) ou (y_A ≥ j ≥ y_m)) alors
            x_B := x_m ;
            y_B := y_m ;
            fin := faux ;
   Fsi
   Si ((x_B ≥ i ≥ x_m) ou (x_B ≤ i ≤ x_m)) et
      (y_B ≤ j ≤ y_m) ou (y_B ≥ j ≥ y_m)) alors
           x_A := x_m ;
           y_A := y_m ;
           fin := faux ;
   Si x_m = i et y_m = j alors
           affichage ;
           fin := vrai ;
   Fsi
FTQ
fin
```

Le temps d'exécution de cet algorithme est proportionnel à $\log_2 n$.

II.1.3.3. Algorithme logique

Stamopoulos dans [STA 74] donne une solution originale de tracé de segment pouvant fonctionner sur des cellules très élémentaires puisque n'utilisant aucun opérateur arithmétique.

On considère une cellule et ses 8 voisines ainsi que les directions de propagation suivantes :



L'algorithme utilise deux mémoires de booléens d'une taille égale à celle de la mémoire d'image, ce qui revient à mémoriser dans chaque cellule deux booléens, le principe en est le suivant :

1) Tracer les droites booléennes en chacune des deux extrémités par propagation suivant la direction 1 et son opposée.

2) Remplir l'espace booléen (booléen = vrai) entre ces deux droites.

3) Garder le résultat dans la première mémoire.

4) Répéter les actions 1) et 2) pour la direction 2 et stocker le résultat dans la deuxième mémoire.

5) Faire un ET logique entre le contenu des mémoires 1 et 2, et stocker le résultat dans la mémoire 1.

6) Répéter les actions 4) et 5) pour les deux directions restantes.

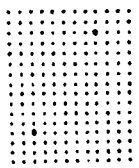
7) Le résultat est ensuite soumis à une procédure d'examen qui élimine successivement dans chaque direction les points qui satisfont à la condition d'examen.

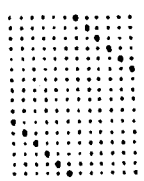
II.1.3.4. Algorithme parallèle [MER 83]

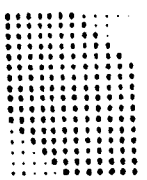
La solution pour obtenir un algorithme totalement parallèle consiste à calculer en chaque point (x, y) l'erreur verticale, c'est-à-dire :

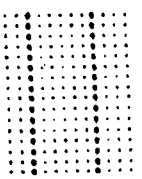
$$\varepsilon = (y - y_a) - (x - x_a) * \frac{\Delta x}{\Delta y}.$$

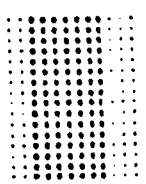
De cette manière, chaque cellule peut calculer sa propre valeur et par comparaison avec un seuil fixé déterminer si elle est une composante du segment.

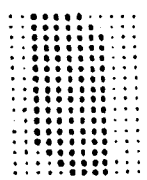


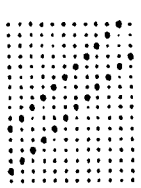


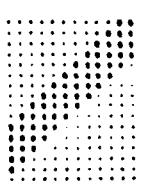


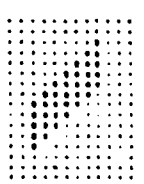


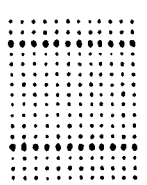


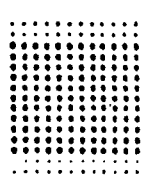


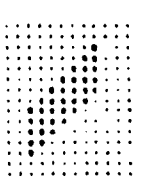


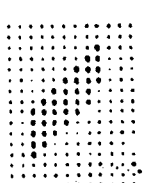


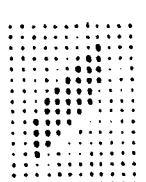


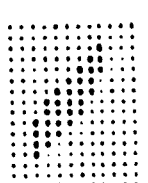


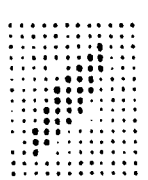


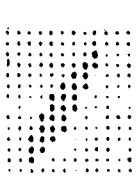


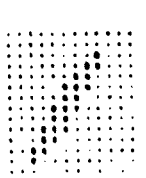


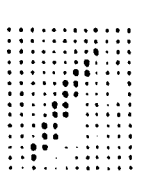


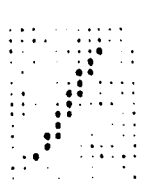


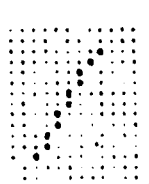












II.2. LE TRACE DE CERCLE

II.2.1. Algorithmes séquentiels [DAN 70], [JOR 73], [MCI 83], [PIT 67]

II.2.1.1. Introduction

Pour élaborer un algorithme de tracé de cercle, on peut employer deux méthodes :

- Soit partir des coordonnées polaires des points du cercle.

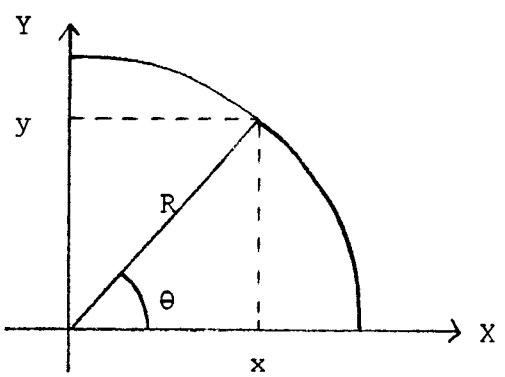
- Soit utiliser la distance d'un point par rapport au centre du cercle.

II.2.1.2. "Méthode polaire"

Les coordonnées polaires d'un point du cercle sont :

$x = R \cos\theta$

$y = R \sin\theta$



On en déduit :

$$\begin{cases} x_i = R \cos\theta_i \\ y_i = R \sin\theta_i \end{cases} \quad \text{et} \quad \begin{cases} x_{i+1} = R \cos(\theta_i + \Delta\theta) \\ y_{i+1} = R \sin(\theta_i + \Delta\theta) \end{cases}$$

Après simplification et approximation [MOR 76] on obtient :

$$x_{i+1} = x_i - \frac{1}{R} * y_i$$

$$y_{i+1} = y_i + \frac{1}{R} * x_{i+1}$$

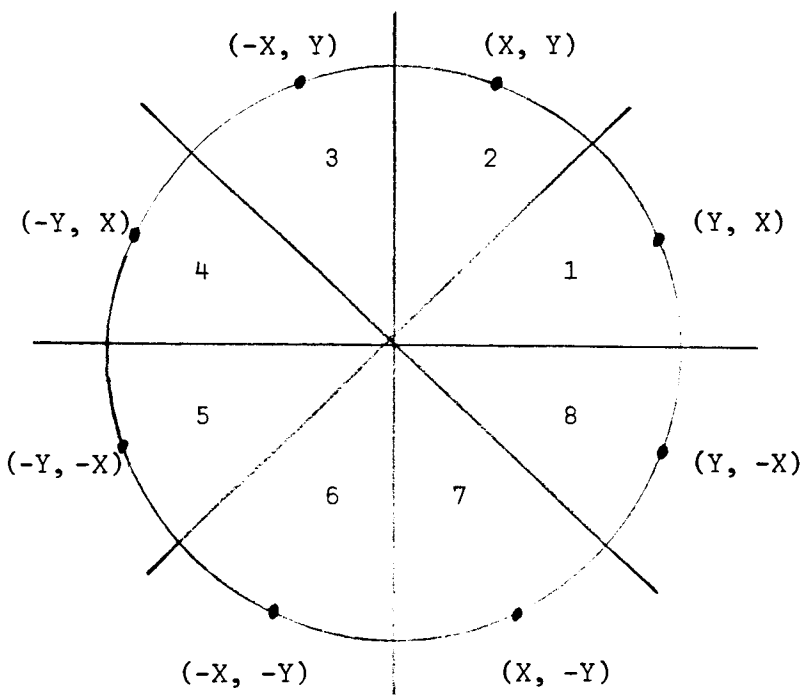
A chaque étape on a :

2(*), 1(-), 1(+), 2(:=), 1(test)

II.2.1.3. Utilisation de la distance

Préliminaires

Les algorithmes présentés tracent les points du 2ème octant, les autres points sont tracés par symétrie de la manière suivante :



La procédure "affichage" trace les huit points correspondants au huit octants.

Algorithmes

La distance d'un point A(X, Y) à l'origine O est :

$$d(O, A) = \sqrt{X^2 + Y^2}$$

Si le centre du cercle C est l'origine on a :

$$A \in C \iff d(O, A) = R$$
$$\iff d^2(O, A) = R^2$$
$$\iff X^2 + Y^2 - R^2 = \emptyset \qquad (1)$$

C'est cette équation qui sert de support pour évaluer un terme d'erreur E(X, Y). Comme :

$$X^2 = \sum_{I=\emptyset}^{X-1} 2I + 1$$

$$Y^2 = \sum_{J:O}^{Y-1} 2J + 1$$

Pour tout point (x, y) de terme d'erreur E(x, y) on a :

$$E(x+1, y) = E(x, y) + 2x + 1$$
$$E(x+1, y-1) = E(x+1, y) - 2y + 1$$

et si l'on prend $E(O, R) = \emptyset$; ce qui permet la soustraction directe de R^2 on obtient :

```
Procédure Brésenham (/* Rayon */ R : entier [BRE 77]

    var : x, y, dx, dy : entiers

début
        x := 0 ;
        y := R ;
        dx := 0 ;
        TQ y > x
            affichage
            dx := dx + 2x + 1
            dy := dx - 2y + 1
            Si ABS(dx) > = ABS(dy) alors
                    y := y-1 ;
                    dx := dy ;
            Fsi
            x := x + 1 ;
        FTQ
        Si x = y alors affichage ;
fin
```

Evaluation

$$t_B = E(R\ (\frac{\sqrt{2}}{2}\ (3(:=) + 4(+) + 1(-) + 2(test)) +$$

$$(1 - \frac{\sqrt{2}}{2}) \times (2(:=) + 1(-)))$$

L'une des premières simplifications possibles [SUE 79] consiste à utiliser un seuil fixe R à la place du test des différentielles x et y permettant ainsi de calculer la valeur de E(x+1, y-1) que lorsque cela est nécessaire.

```
Procédure ASEUIL1 (R : entier)

      var : x, y, dx : entiers

début
        x := 0 ;
        y := R ;
        dx := 0 ;
        TQ : y > x
             affichage
             dx := dx + 2x + 1 ; /* E(x+1, y) */
             Si ABS(dx) > = R alors
                dx := dx - 2y + 1 ; /* E(x+1, y-1) */
                y := y-1 ;
             Fsi
             x := x+1 ;
        FTQ
        Si x = y alors affichage
fin
```

Le seuil fixe bien que permettant une plus grande rapidité d'exécution conduit à une approximation du cercle. En effet certains points sont décalés sur l'ordonnée d'une position.

Pour remédier à cet inconvénient on peut utiliser un seuil variable égal à 2y [CAR 77].

```
Procédure ASEUILVARIABLE (R : entier)

        var : x, y, dx : entiers

début
        x := 0 ;
        y := R ;
        dx := R ;
        TQ : Y > X
             AFFICHAGE
             dx := dx + 2x + 1
             Si ABS(dx) ⩾ 2y alors
                dx := dx - 2y + 1 ;
                y := y-1 ;
             Fsi
             x := x + 1 ;
        FTQ
        Si X = Y alors AFFICHAGE ;
fin
```

Changement de distance

On peut encore améliorer la rapidité de cet algorithme en posant comme distance d'un point A(X, Y) à l'origine O.

$$d'(O, A) = \frac{1}{\sqrt{2}} * \sqrt{X^2 + Y^2 + X + Y}$$

Si un point B appartient au cercle C on a :

$$\begin{aligned} A \in C &\iff d'(O, A) = d'(O, B) \\ &\iff d'^2(O, A) = d'^2(O, B) \\ &\iff \frac{X(X+1)}{2} + \frac{Y(Y+1)}{2} = d'^2(O, B) \end{aligned}$$

ORS

$$\frac{X(X+1)}{2} = \sum_{I:1}^{X} I$$

et

$$\frac{Y(Y+1)}{2} = \sum_{J:1}^{Y} J$$

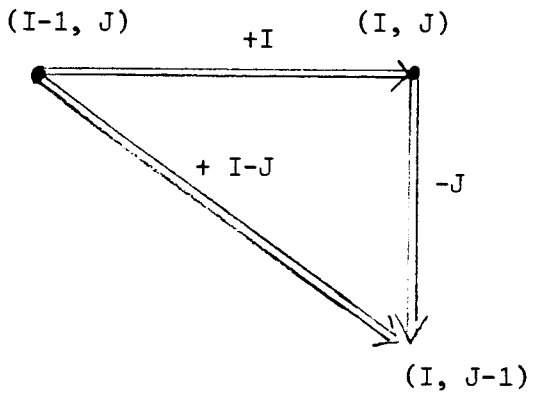
en prenant comme approximation de

$d'^2(0, B) = \frac{R(R+1)}{2}$ on obtient :

$$E(X, Y) = \frac{X(X+1)}{2} + \frac{Y(Y+1)}{2} - \frac{R(R+1)}{2}$$

Si l'on se restreint au deuxième octant et si l'on prend E(0, R) = 0 on a :

$$E(I, J) = E(I-1, J) + I$$
$$E(I, J-1) = E(I, J) - J$$



Pour le seuil S on utilise la valeur moyenne du terme d'erreur E pour les points appartenant au cercle affiché.

Le résultat obtenu empiriquement nous donne :

$$S = 0,6\ R + 0.28$$

Remarque : Du fait de l'utilisation d'un seuil fixe, une faible proportion de points diffère des points affichés par l'algorithme de Brésenham (exemple : 124 pour un cercle de rayon 2048).

Dans ce cas au lieu de (X, Y) on a (X, Y±1).

```
PROCEDURE CERCLE ({RAYON} R : ENTIER)

        {le centre est l'origine}

        var : x, y, E, S : entiers.

début
        x := 0 ;
        y := R ;
        E := 0 ;
        S := PARTIE ENTIERE (0,6 R + 0.28)
        TQ : y > x
                AFFICHAGE
                  x := x+1 ;
                  E := E+x ;
                Si ABS(E) > S alors
                     E := E-y
                     y := y-1 ;
                Fsi
        FTQ
        Si x = y alors AFFICHAGE ;
fin {CERCLE}
```

$$t_i = E(R\ (\frac{\sqrt{2}}{2}\ (2(:=) + 2(+) + 2(test)) + (1-\frac{\sqrt{2}}{2}) \times (2(:=) + 2(-)))$$

II.2.2. Etude du parallélisme

II.2.2.1. Parallélisme total

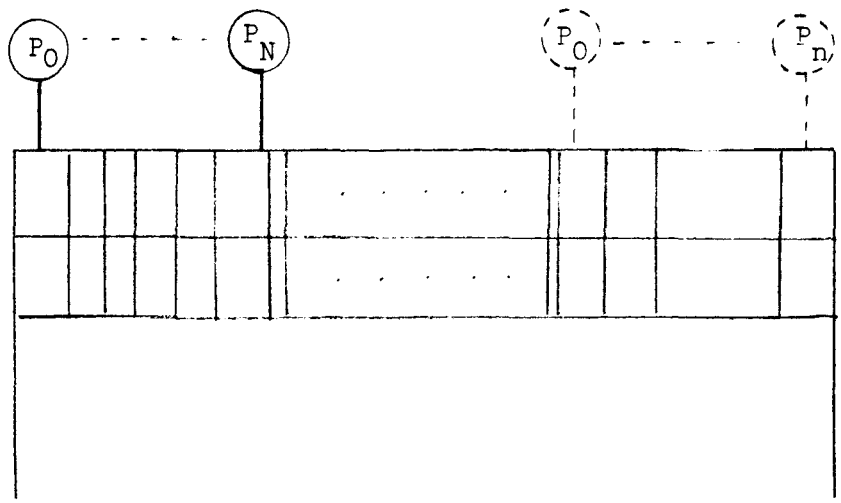
Du fait des aspects incrémentaux des divers algorithmes de tracé de cercle, il n'est pas possible de réaliser, à partir de ceux-ci un algorithme totalement parallèle.

La seule solution, coûteuse en temps de calcul car utilisant des opérations complexes consiste à résoudre pour chaque point du plan d'affichage l'équation du cercle.

II.2.2.2. Parallélisme partiel

L'étude est réalisée à titre d'exemple et en vue d'une implémentation sur une machine "ligne" (MAP : c'est-à-dire que les N processeurs gèrent chacun un sous-ensemble de la mémoire d'image. Le p^e processeur gère les pixels de numéro : p + i * N, i étant le nombre de blocs dans la ligne.

Les processeurs disposent de moyens de communication et de synchronisation.



* Principe de l'algorithme :

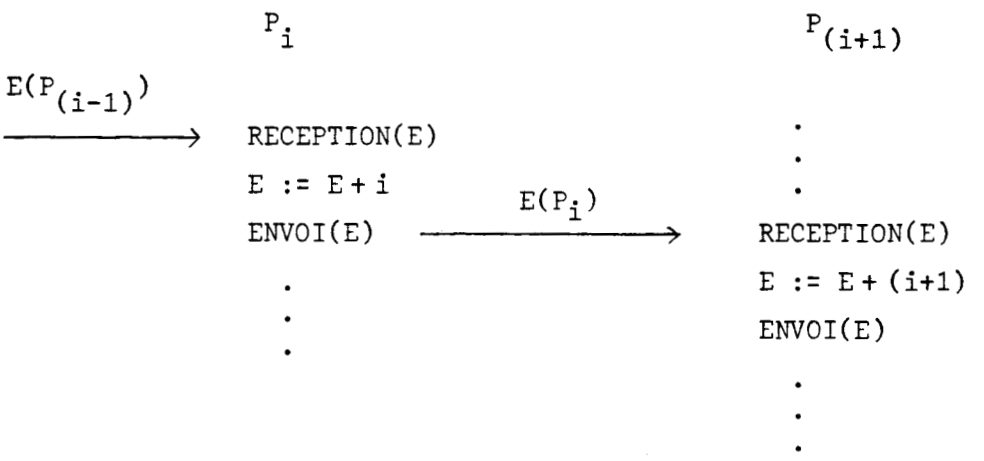
n : nombre de processeurs.

```
Procédure cercle (R : rayon)

début
        Si n ≥ 2R alors
                DEMI-CERCLE
        Sinon
                Quadrant
        Fsi
fin
```

Les deux procédures demi-cercle et quadrant comportent deux phases de calcul distinctes.

a) Un calcul pipe-line du terme d'erreur E par rapport à X.

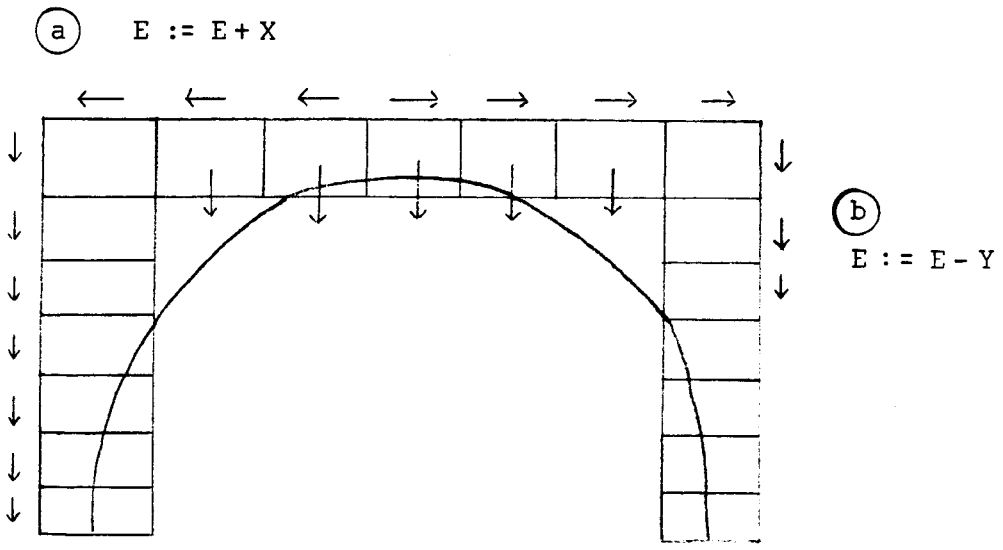


b) Un calcul parallèle du terme d'erreur E par rapport à y.

A chaque étape les processeurs testent la valeur E par rapport au seuil fixe S.

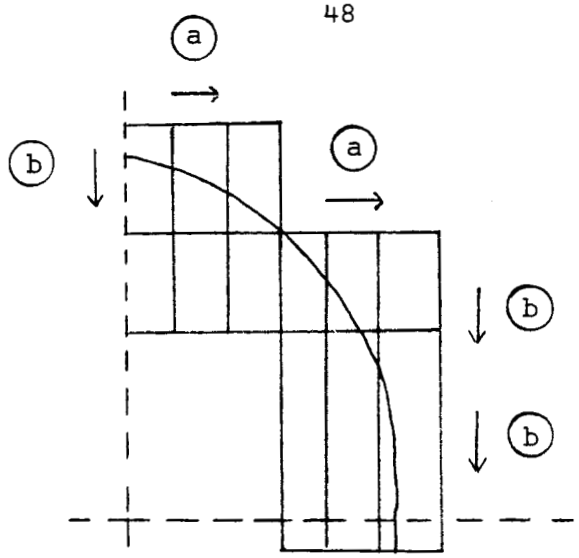
Procédure demi-cercle.

```
début
        Y := R ;
        . CALCUL PIPE-LINE de E(X, R)
        . Ecriture // des blocs de ligne (R, -R)
        TQ : Y ≥ 0
             CALCUL // de E(X, Y)
             Y := Y - 1 ;
             Ecriture // BLOCS DE LIGNE (Y, -Y)
        FTQ
fin
```



Procédure quadrant.

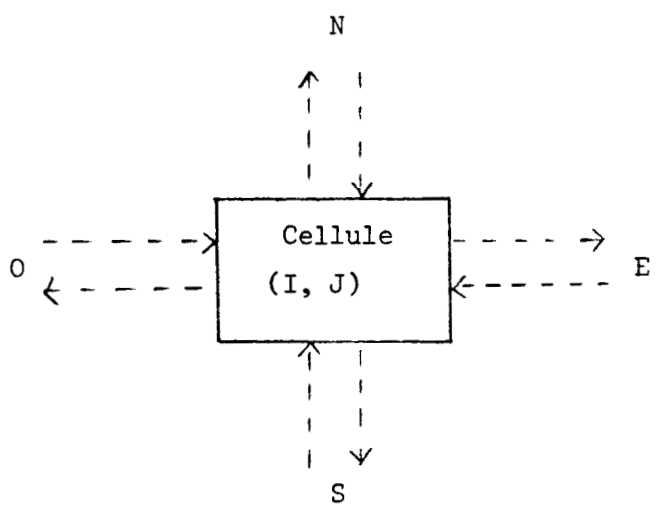
```
début
        y := R ;
        x0 := 0 ;
        TQ   Y ≥ 0 ;
             B := FAUX ;
             CALCUL PIPE-LINE DE E
             ECRITURE DES BLOCS
             TQ ٦ B
                  CALCUL // de E
                  Si E(X(Pn), Y) < -S
                     B := VRAI
                  SINON
                     Y := Y - 1
                  Fsi
                  ECRITURE DES BLOCS
             FTQ
        FTQ
fin
```



Remarque : Il n'y a pas de parallélisme effectif dans le premier octant.

II.2.2.3. Application sur machine cellulaire

On considère une cellule ayant quatre voies de communication.



La procédure de tracé du cercle de rayon R et de centre (X_C, Y_C) consiste alors à :

- Dans un premier temps, initialiser à 0 les termes d'erreur des cellules

$(X_C + R, Y_C)$; $(X_C - R, Y_C)$; $(X_C, Y_C + R)$; $(X_C, Y_C - R)$

Chacune de ces cellules envoyant alors cette valeur à ses voisines on obtient l'algorithme suivant.

Cellule (i, j)

```
début
        attente réception /* E(m, n) */
        Calcul du terme d'erreur E(i, j)
        envoi E(i, j) /* aux 4 voisines */
        Si -S ≤ E(i, j) ≤ +S alors
            affichage
        Fsi
fin.
```

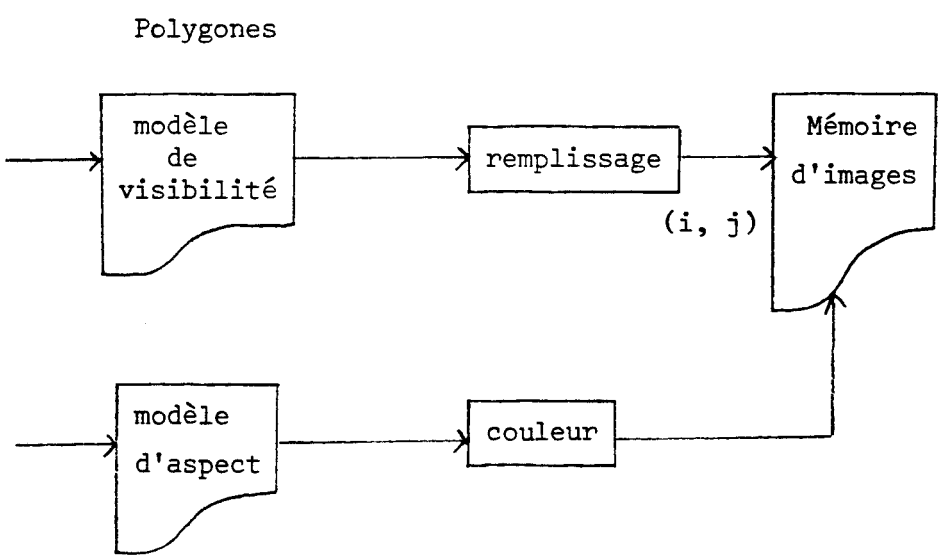
Calcul du terme d'erreur E(i, j)

```
début
        test origine de la valeur
          Si N alors E(i, j) := E(i, j) + j-1
          Si S alors E(i, j) := E(i, j) - j+1
          Si E alors E(i, j) := E(i, j) - i+1
          Si O alors E(i, j) := E(i, j) + i-1
fin
```

Le temps d'exécution de cet algorithme est proportionnel à $(\frac{\sqrt{2}}{2} R)$.

II.3. LE REMPLISSAGE

Le remplissage consiste à effectuer la coloration d'une zone définie par un contour fermé (modèle de visibilité) à l'aide d'une règle de remplissage (modèle d'aspect) dépendante de divers paramètres (nombre de sources lumineuses, ombrage, texture,...).



Suivant "l'accès" au contour, on peut définir deux grandes classes d'algorithmes :

- La première comprenant les algorithmes de remplissage de contours préinscrits en mémoire d'images.

- La deuxième classe étant composée des algorithmes de remplissage d'un contour polygonal décrit sous forme d'une liste de coordonnées de ses sommets. Cette deuxième classe est un sous produit des algorithmes d'élimination de surfaces cachées.

A partir de la cohérence et de la connexité des pixels, deux techniques principales peuvent servir à effectuer le remplissage de contours préinscrits :

- L'une réalise le remplissage à partir d'un point donné comme étant à l'intérieur du contour.

- L'autre consiste à partir de l'extérieur du contour et à examiner pour chaque ligne le nombre d'intersections afin de définir en tenant compte de certaines singularités les pans horizontaux de remplissage.

On peut trouver des algorithmes correspondants à ces méthodes dans [ACK 81], [DUN 83], [GOU 71], [PAV 81], [LAN 83], [LIE 81], [LIE 83], [LIT 79].

II.3.1. Remplissage à partir d'un point intérieur

II.3.1.1. Algorithme séquentiel

Le fait de donner un point intérieur au contour permet la propagation du remplissage aux pixels voisins jusqu'aux bords de celui-ci.

L'algorithme séquentiel classique tiré de [LUC 79] est le suivant :

Remarque : On trouvera dans [SMI 79] une version plus performante.

```
Procédure remplissage (var : xdeb, ydeb : entiers)

      var : x, y, xsauve, ibord, x_max, x_min, xdroit, xgauche : entiers.

      image (m, n) : tableau

/* image (i, j) : donne la valeur correspondant à la couleur du pixel (i, j) */

/* ibord : valeur correspondant à la valeur du bord */

début
      x := xdeb ;
      y := ydeb ;
      empiler (x, y) ;
      TQ ¬ Pile vide
            dépiler (x, y) ;
            Si image (x, y) ≠ ibord alors
                  a-droite
                  a-gauche
                  voir-dessus
                  voir-dessous
            Fsi
      FTQ
fin
```

Procédure a-droite.

```
début
      xsauve := x ;
      TQ image (x, y) ≠ ibord et x ≤ x_max
            image (x, y) := ibord ;
            x := x + 1 ;
      FTQ
      xdroit := x - 1 ;
      x := xsauve -1 ;
fin
```

Procédure a-gauche.

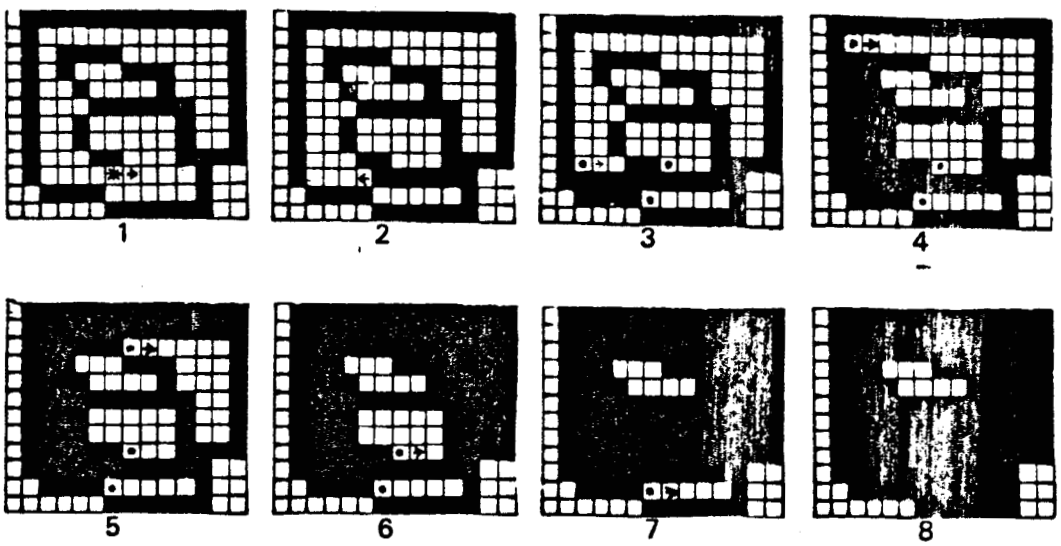
```
début
       xsauve := x ;
       TQ image (x, y) ≠ ibord et x ≥ x_min
                       image (x, y) := ibord ;
                       x := x - 1 ;
       FTQ
       xgauche := x + 1 ;
       x := xsauve ;
fin
```

Procédure voir-dessus.

```
début
       Si y+1 ≤ y_max alors
              xsauve := x ;
              ysauve := y ;
              x := xgauche ;
              y := y + 1 ;
       TQ x ≤ xdroit
                 TQ image (x, y) = ibord et x ≤ xdroit
                    x = x + 1 ;
                 FTQ
          Si x ≤ xdroit alors
                 empiler (x, y) ;
                 TQ image (x, y) ≠ ibord et x ≤ xdroit
                      x := x + 1 ;
                 FTQ
          Fsi
       FTQ
       x := xsauve ;
       y := ysauve ;
    Fsi
fin
```

La procédure voir-dessous est dérivée de voir-dessus en remplaçant le test $y+1 \leq y_{max}$ par $y-1 \geq y_{min}$ et l'affectation y := y+1 par y := y-1.

La figure suivante montre le mécanisme du remplissage. Les procédures a-droite et a-gauche effectuent le remplissage jusqu'à la détection d'un bord droit et gauche. Les procédures voir-dessus et voir-dessous examinent la ligne supérieure et inférieure à la ligne courante afin de déterminer les points de départ des pans horizontaux de remplissage.



II.3.1.2. Etude du parallélisme

On peut trouver deux méthodes pour paralléliser efficacement cet algorithme :

(1) Réaliser une scrutation de la pile par tous les processeurs et lancer un traitement complet pour un processeur dès que les coordonnées d'un pixel sont présentes.

(2) Avoir un algorithme identique mais réaliser un examen et un remplissage multiple par ligne.

La première solution sous réserve d'avoir réglé les conflits d'accès à la pile et à la mémoire d'image permet une programmation aisée de par son aspect séquentiel au niveau du processeur.

La solution (2) du fait du principe d'exploration de l'algorithme présente l'avantage de s'accommoder facilement d'une mémoire d'images partitionnée en colonne avec adressage par ligne et des processeurs disposant de moyens de communication et de synchronisation, les conflits d'accès sont alors inexistants.

La procédure "à-droite" consiste alors pour l'ensemble des processeurs concernés à lire, à partir des coordonnées dépilées d'un pixel, la partie droite de la ligne. L'ensemble des processeurs se communiquent le résultat de l'examen de leur pixel (bord/non bord), il suffit ensuite de réaliser un tri en x pour déterminer les extrémités du pan horizontal d'affichage, les processeurs n'ayant plus qu'à tester leur position par rapport à ce pan pour déterminer l'action à entreprendre (remplissage ou non).

La procédure à-gauche est similaire et les procédures voir-dessus et voir-dessous se réduisent à l'examen multiple et au tri pour déterminer les coordonnées à empiler.

II.3.1.3. Algorithme cellulaire

Dans un réseau de cellules possédant quatre voisines, la procédure la plus simple consiste à s'inspirer du célèbre "jeu de la vie".

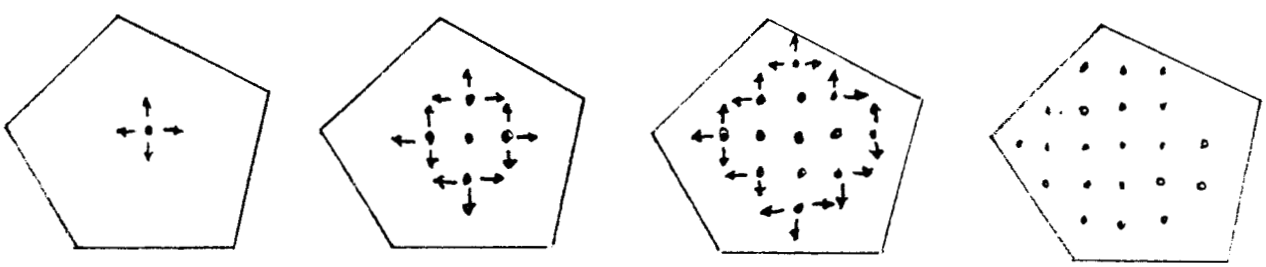
Le remplissage consiste alors à partir du point intérieur au contour et de propager, à partir de celui-ci, dans toutes les directions la notion d'intériorité jusqu'à la rencontre des bords.

On déduit de ce principe l'algorithme suivant :

Cellule (i, j)

```
début
        Réception (x, y) /* (x, y) est le point intérieur */
        Si i = x et j = y alors
             image (i, j) := couleur
             envoi (indicateur)
        Sinon
             Attente (indicateur)
             Si image (i, j) ≠ ibord alors
                image (i, j) = couleur
                envoi (indicateur)
             Fsi
        Fsi
fin
```

Schéma



On remarquera qu'on ne peut pas déterminer la fin du traitement, en effet les cellules externes au contour attendent toujours. Pour résoudre ce problème on peut imaginer deux solutions [MER 84] telles qu'un time-out ou un signal collectif d'activité.

II.3.2. Remplissage par examen de voisinage

La détermination d'un point intérieur à un contour préinscrit pouvant s'avérer coûteuse en temps de calcul si celui-ci n'est pas donné, il s'avère intéressant de pouvoir effectuer le remplissage en partant de l'extérieur du contour (rectangle englobant ou bord de la mémoire d'images).

Le principe consiste à tenir compte des informations établies pour les pixels examinés auparavant, pour déterminer à l'aide de la connexité et de la cohérence de celle-ci l'action à entreprendre.

Dans un premier temps, nous allons faire l'étude pour une machine parallèle à mémoire d'image partitionnée en colonne. La méthode pouvant être appliquée facilement suivant le même principe à d'autres types de partitionnement. La machine doit posséder des moyens de synchronisation et de communication, l'adressage de la mémoire s'effectue sur une ligne à la fois.

Le processeur élémentaire adressant un pixel de la ligne.

II.3.2.1. Mémoire d'image sans conflit d'accès

Contraintes

- Effectuer un balayage de la mémoire d'images ligne par ligne de haut en bas, chacune d'elles étant traitée par l'ensemble des processeurs.

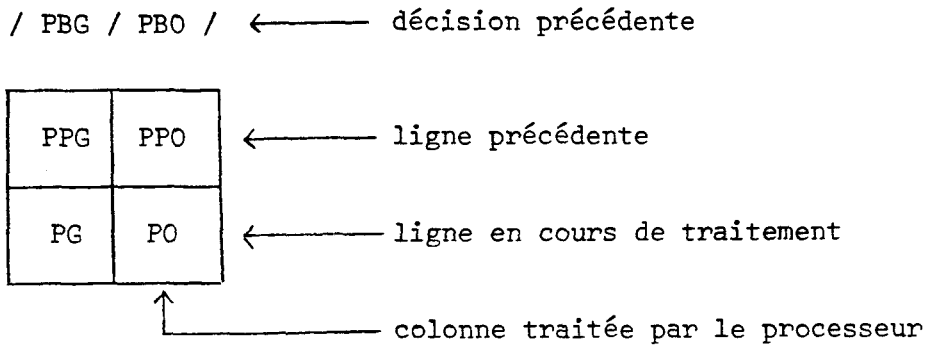
- Chaque processeur décide en évitant d'utiliser le résultat de son voisin de gauche afin de préserver le parallélisme.

Méthode

Le processeur traitant un pixel PO doit prendre une décision (point intérieur/extérieur).

Il peut observer le pixel PO (bord/non bord) et son voisin de gauche PG car le processeur à sa gauche peut lui communiquer l'état de ce pixel, ainsi que l'état du pixel précédent PPG et de la décision qui avait été prise PBG (point intérieur/extérieur) lors du traitement de ce pixel sur la ligne précédente (même colonne) si l'on a pris soin de mémoriser ces deux informations booléennes.

On peut pour les mêmes raisons disposer de PPO et de PBO selon le schéma ci-dessous.



On rappelle que les informations PBO (Précédent Booléen Elaboré) et PPO (Précédent Pixel Observé) sont mémorisées lors du traitement de la ligne précédente.

Il reste à étudier tous les cas possibles ($2^6 = 64$) et pour chacun de ces cas prendre une décision en élaborant un booléen :

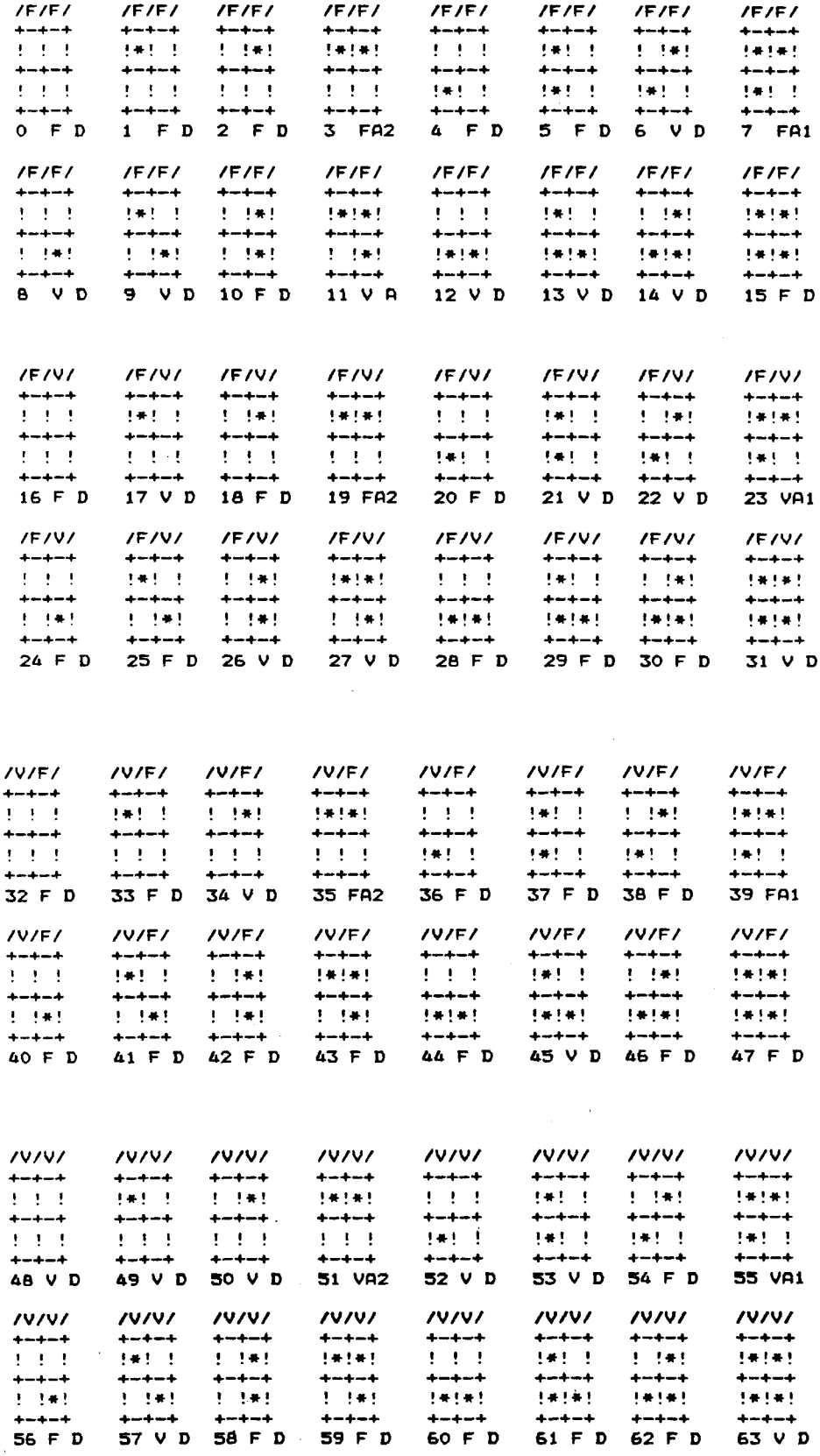
BO - V (point intérieur)
- F (point extérieur)

et une information

ALEA - A1, A2 (il y a un aléa de type 1 ou 2)
- D (la décision est valable).

Remarque : La première ligne et la première colonne demandent un traitement particulier du fait de l'abscence de 4 informations (colonne ou ligne précédente).

On considèrera alors le pixel comme étant intérieur que lors de la rencontre d'un bord.


/F/F/ /F/F/ /F/F/ /F/F/ /F/F/ /F/F/ /F/F/ /F/F/
0 F D 1 F D 2 F D 3 FA2 4 F D 5 F D 6 V D 7 FA1
/F/F/ /F/F/ /F/F/ /F/F/ /F/F/ /F/F/ /F/F/ /F/F/
8 V D 9 V D 10 F D 11 V A 12 V D 13 V D 14 V D 15 F D
/F/V/ /F/V/ /F/V/ /F/V/ /F/V/ /F/V/ /F/V/ /F/V/
16 F D 17 V D 18 F D 19 FA2 20 F D 21 V D 22 V D 23 VA1
/F/V/ /F/V/ /F/V/ /F/V/ /F/V/ /F/V/ /F/V/ /F/V/
24 F D 25 F D 26 V D 27 V D 28 F D 29 F D 30 F D 31 V D
/V/F/ /V/F/ /V/F/ /V/F/ /V/F/ /V/F/ /V/F/ /V/F/
32 F D 33 F D 34 V D 35 FA2 36 F D 37 F D 38 F D 39 FA1
/V/F/ /V/F/ /V/F/ /V/F/ /V/F/ /V/F/ /V/F/ /V/F/
40 F D 41 F D 42 F D 43 F D 44 F D 45 V D 46 F D 47 F D
/V/V/ /V/V/ /V/V/ /V/V/ /V/V/ /V/V/ /V/V/ /V/V/
48 V D 49 V D 50 V D 51 VA2 52 V D 53 V D 54 F D 55 VA1
/V/V/ /V/V/ /V/V/ /V/V/ /V/V/ /V/V/ /V/V/ /V/V/
56 F D 57 V D 58 F D 59 F D 60 F D 61 F D 62 F D 63 V D

Il existe deux possibilités d'élaborer ces deux informations :

- D'une part, en utilisant les fonctions booléennes simplifiées.

- D'autre part, en stockant en mémoire les tables de décisions.

Traitement de l'aléa du type 2 $\begin{matrix}**\\**\end{matrix}$

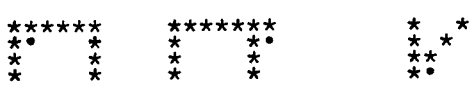
Dans le cas ou un aléa du type A2 est détecté, le processeur doit nécessairement utiliser une information supplémentaire, constituée par la décision de son voisin de gauche. Le traitement en parallèle n'est plus possible car les processeurs élaborant les colonnes de droite doivent attendre.

Une fois que le processeur à reçu le résultat de son voisin de gauche, il peut élaborer sa propre décision et communiquer celle-ci à son voisin de droite. On utilise alors la technique des rendez-vous.

Traitement de l'aléa du type 1 $\begin{matrix}**\\*.\end{matrix}$

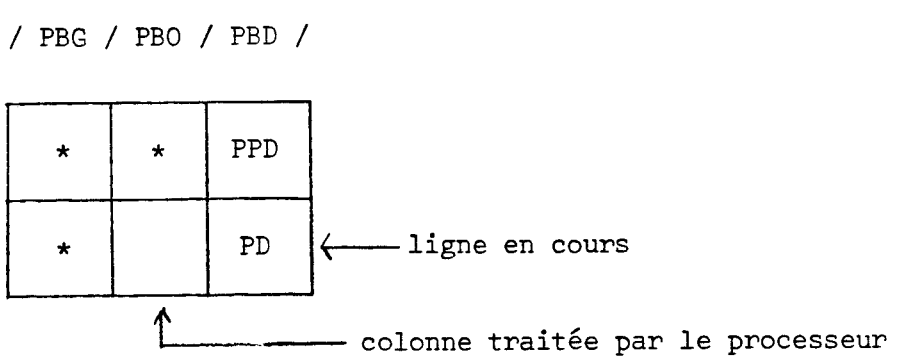
Dans ce cas, la décision ne peut être immédiate car l'on ignore ce qui est à droite.

Exemples :

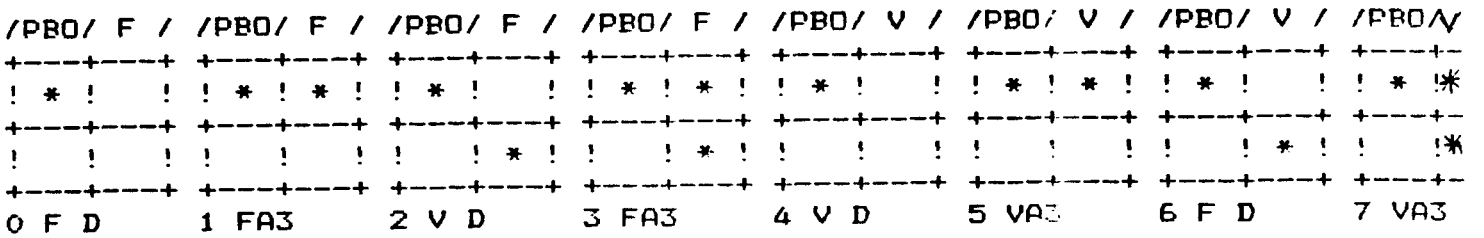


Pour pallier cet inconvénient on peut accéder rapidement à trois informations supplémentaires pour essayer de prendre une décision.

- L'état du pixel situé à droite PD (bord/non bord).
- L'état du pixel droit de la ligne précédente PPD.
- La décision qui s'y rattache PBD.



On obtient 8 cas :



Dans cette seconde décision on voit apparaître un aléa de type 3 (A3) :

Traitement de l'aléa du type 3

L'existence de cet aléa ne dépend que des contraintes imposées au contour :

- Si le contour est fermé, on peut prendre une décision et celle-ci est correcte.

- Si le contour est quelconque, l'aléa existe et peut-être résolu par un algorithme de suivi de contour (le parallélisme de remplissage est alors interrompu).

Conclusion

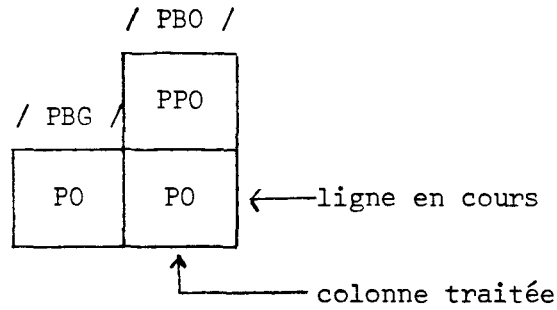
On peut constater que le balayage s'effectue en une seule fois et que le nombre d'aléas possibles (1/8 des cas)perturbe très peu le taux de parallélisme obtenu du fait de leur faible fréquence de rencontre.

Cet algorithme peut traiter des contours fermés ainsi que des contours quelconques si l'aléa du type 3 est résolu par un suivi de contour. De plus, le remplissage de plusieurs contours disjoints (si A3 n'est pas résolu) peut être effectué simultanément.

II.3.2.2. Mémoire d'image à accès aléatoire ou machine cellulaire

Dans l'algorithme précédent 6 informations étaient nécessaires pour éviter au maximum les conflits d'accès. Si ceux-ci n'ont plus de raison d'être, on peut élaborer une décision pour un pixel donné avec seulement 5 informations.

- L'état du pixel observé PO (bord/non bord).
- L'état du précédent pixel observé PPO.
- La décision PBO se rattachant au PPO.
- L'état du pixel voisin gauche PG.
- La décision PBG rattaché au PG.



II.4. PARALLELISATION D'ALGORITHMES DE SURFACES CACHEES PAR ANALYSE DE LIGNE

L'un des problèmes de la synthèse d'images consiste à éliminer les parties cachées des représentations d'objets solides dans les images. Lorsque les objets de la scène sont projetés sur l'écran, toutes les parties de chaque objet, dont de nombreuses qui devraient être invisibles, sont affichées.

Pour créer une image plus réaliste, il faut enlever ces parties, pour cela nous devons utiliser un algorithme de surfaces cachées appliqué à l'ensemble de ces objets.

De nombreux algorithmes ont été développés dans ce but (cf. [BOU 80], [AV 81], [GRO 79]).

Certains de ceux-ci effectuent une analyse par ligne de l'image afin de respecter la cohérence d'affichage des moniteurs à balayage.

L'étude, ci-après, essaie de déterminer la meilleure implémentation de ces algorithmes pour leur parallélisation.

II.4.1. Introduction

Un objet est souvent décomposé en facettes, elles-mêmes exprimées, le plus couramment, sous forme de polygones. Les polygones sont alors les faces d'un polyhèdre mémorisées dans le modèle de la scène ou représentent simplement une plaque opaque dans celle-ci.

On représente un polygone à l'aide de l'énumération de ses sommets.

Par exemple :

$$P : (A(x_a, y_a, z_a), B(x_B, y_B, z_B), C(x_C, y_C, z_C))$$

Cette suite permet de définir le contour de la face constitué par les segments (A, B), (B, C), (C, A).

Les algorithmes de traitement de faces cachées utilisant le système de coordonnées écran, il est nécessaire d'effectuer avant ceux-ci, la transformation de tous les polygones à l'aide d'une matrice de multiplication pour obtenir les changements de vue et de perspective, ainsi que leurs découpages à l'aide d'une fenêtre afin de s'assurer qu'ils résident dans le cadre de vue (clôture).

La plupart des algorithmes de traitement des surfaces cachées rencontrés travaillent par ligne, le choix de l'étude s'est porté sur cette approche.

Pour traiter, dans cette optique, l'affichage d'un ensemble de polygones, il existe deux techniques principales permettant d'obtenir deux types de traitements parallèles distincts.

- Le premier travaille ligne après ligne ; cette méthode ne permet qu'un parallélisme sur le traitement de la ligne, (on peut se référer à [HWA 83] comme exemple d'application), ou utilisation d'un traitement pipe-line.

- Le deuxième élabore dans un premier temps, en parallèle, une structure de données dont on précise la nature par la suite, pour réaliser le traitement par un parallélisme sur les lignes.

Cette deuxième méthode, malgré un besoin mémoire beaucoup plus conséquent, permet un parallélisme plus poussé et logiquement plus simple. En effet, si la

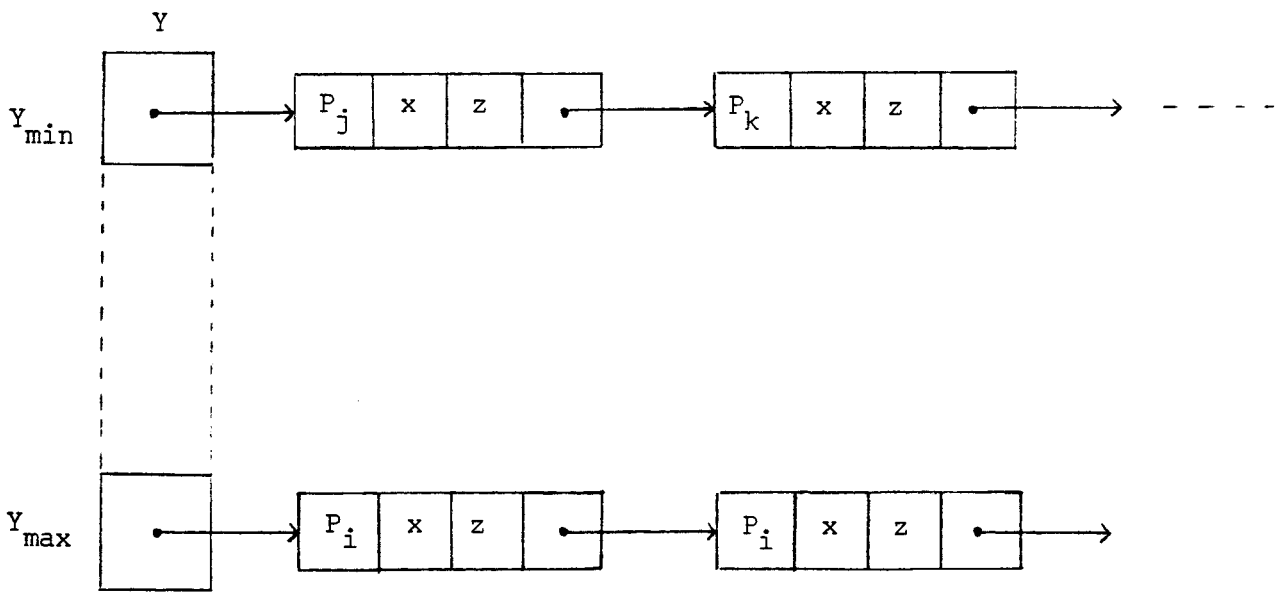
structure de données est entièrement constituée, chaque processeur peut prendre en charge le traitement d'une ligne, celui-ci étant réalisé de manière séquentielle.

II.4.2. Parallélisme sur les lignes

II.4.2.1. Constitution de la structure de données

La structure de données peut être considérée comme constituée d'une table de pointeurs de lignes écran et pour chaque ligne d'une liste de quadruplets représentant l'intersection de tous les segments des polygones avec celle-ci :

- Identificateur de polygone.
- Coordonnée x.
- Coordonnée z.
- Pointeur vers l'élément suivant.



Pour une élaboration performante, la constitution de la structure de données est réalisée à l'aide d'un algorithme parallèle de tracé de segment. Les polygones sont traités un par un, et tous les points des segments sont déterminés en parallèle et placés dans la structure de données en tenant compte des singularités ci-après :

- Lorsqu'un segment est commun à plusieurs polygones il n'est pas recalculé, mais ses points sont dupliqués avec des identificateurs de polygone différents.

- Lorsque l'incrément en y change de signe pour deux segments successifs d'un même polygone, le sommet commun est dupliqué, sinon il n'apparaît qu'une seule fois.

- Pour des segments horizontaux, les points ne sont pas traités, seuls les sommets sont indiqués et n'apparaissent qu'une fois.

- Pour les verticaux, tous les points sont traités, et les sommets n'apparaissent qu'une fois

- On détermine l'intervalle (y_{min}, y_{max}) des lignes à traiter lors de la constitution de la structure de données.

II.4.2.2. Les différents tris

On doit réaliser différents tris suivant l'algorithme utilisé par la suite.

Dans l'algorithme présenté seul deux tris sont nécessaires : l'un en x, l'autre en y (il est à remarquer qu'il n'y a pas de tri en z).

Le tri en y est réalisé par adressage de la table, tandis que le tri en x est réalisé par insertion dans la liste.

Une autre solution consisterait à effectuer dans un premier temps toutes les interpolations de segments et à réaliser par la suite deux tris parallèles successifs : l'un en y, l'autre en x. Pour la réalisation de ces deux tris, on pourra se référer à [RAN 82], [NAS 82] et [ALLI 82].

II.4.2.3. Traitement général

Une fois la constitution de la structure de données terminée, le remplissage et le traitement des faces cachées pour chaque ligne écran peut facilement se faire en parallèle en attribuant le traitement d'une ligne à chaque processeur.

Algorithme général :

```
début
      Pour Processeur P : 0 à n-1
          I := Y_min + P ;
          TQ : I ≤ y_max
               traitement ligne (I) ;
               I := I+N ;
          FTQ
      Fin Processeur
fin
```

Le traitement ligne se réalise, au niveau de chaque processeur élémentaire de manière séquentielle, ce qui présente l'avantage d'une implémentation aisée. Pour ce traitement on peut utiliser n'importe quel algorithme de traitement des faces cachées travaillant en ligne sous réserve d'effectuer les divers tris nécessaires.

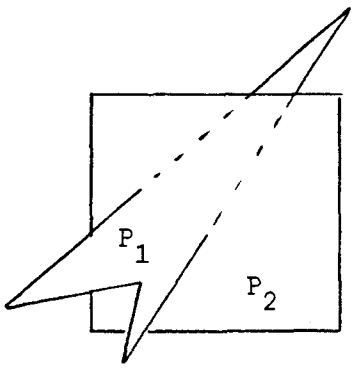
L'algorithme présenté ci-après utilise le même principe que l'algorithme du "peintre" ; mais il traite les parties de polygones recouvertes par l'utilisation de la structure de données et non grâce à l'affichage.

II.4.2.4. Un algorithme de traitement ligne

L'algorithme présenté ci-après, effectue le remplissage avec une couleur uniforme et pour des facilités de présentation sans anti-aliasing.

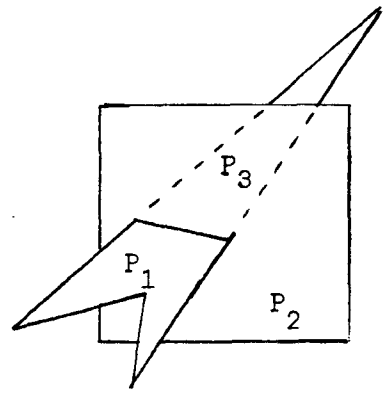
Dans ce cas, seul un point d'intersection entre un segment et la ligne est nécessaire, les coordonnées sont alors arrondies.

Il est à noter que, de par son principe l'algorithme ne traite pas les polygones qui s'interpénètrent sans intersection.



non

Il y a interpénétration



oui

Le polygone P_1 a été découpé en deux polygones P_1 et P_3.

Principe

Chaque pan horizontal d'un polygone étant déterminé par un couple de coordonnées (x, y, z) et les listes lignes étant triées en x, il devient aisé de déterminer pour chaque ligne les divers recouvrements.

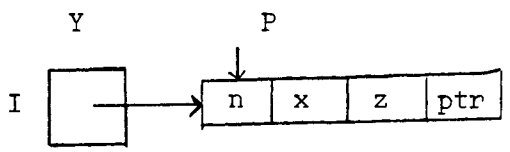
En effet, le premier quadruplet rencontré peut alors servir de référence jusqu'à la rencontre du quadruplet correspondant au même polygone, sauf lors de la rencontre d'un quadruplet correspondant à un autre polygone. Plusieurs cas peuvent alors se produirent.

- Ce dernier peut avoir une profondeur z supérieure : il représente alors l'extrémité gauche ou droite d'un pan de polygone recouvert. On se sert alors d'une liste auxiliaire permettant de mémoriser les quadruplets représentant les débuts de ces pans. Si le quadruplet possède un identificateur de polygone non présent dans cette liste, il définit l'extrémité gauche d'un pan recouvert et est alors inséré dans la liste, sinon il indique l'extrémité droite, le quadruplet correspondant peut être supprimé de cette liste.

- Si le quadruplet à une profondeur z inférieure, il définit le début d'un pan recouvrant le pan en cours d'examen. On peut alors afficher le début du pan de référence et insérer avec la coordonnée x de l'autre le quadruplet de référence dans la liste auxiliaire, le nouveau quadruplet sert alors de nouvelle référence.

Lors de la rencontre avec un quadruplet correspondant à celui de référence, on affiche le pan ainsi défini. Pour prendre une nouvelle référence (z_{min}), on examine alors la liste auxiliaire.

algorithme



n : identificateur de polygone.
ptr : pointeur.
Y : tableau de pointeurs (lignes).
I : indice de ligne.
S : pointeur courant de la liste C des polygones recouverts.
réf : adresse l'élément de comparaison.
p : pointeur de la liste définissant la ligne.

```
début
      p := Y(I) :
      Si p ≠ nil alors
           copie elt(p) dans elt(réf) ;
           p := p . ptr
           TQ : p ≠ nil
                Si p . n = réf . n alors
                   /* cas des deux points du même polygone */
                   affichage de réf . x à p . x ;
                   recherche de z_min dans C ;
                   Si s ≠ nil alors
                      copie elt(s) dans elt(réf) ;
                      réf . x := p . x + 1 ;
                   Sinon
                      p := p . ptr ;
                      copie elt(p) dans elt(réf) ;
                   Fsi
```

```
Sinon
      Si p . x : ref . x alors
               /* un bord commun à deux polygones */
           Si p . z ≥ ref . z alors
              insertion en tête (C, p) ;
           Sinon
              insertion en tête (C, ref) ;
              copie elt(p) dans elt(ref) ;
           Fsi
      Sinon
      Si p . z ≥ ref . z alors
               /* recherche dans C */
           S := tête (C) ;
           TQ S ≠ nil et s . n ≠ p . n
              S := S . ptr
           FTQ
           Si s . n = p . n alors
              /* polygone recouvert */
              suppression (C, S)
           Sinon
             insertion en tête (C, p) ;
             /* début de polygone recouvert */
           Fsi
      Sinon
           /* recherche dans C */
           S := tête (C) ;
           TQ S ≠ nil et s . n ≠ p . n
              S := S . ptr ;
           FTQ
           Si S . n = p . n alors
              /* même points de début */
              affichage (de s . x à p . x) ;
              suppression (C, s) ;
              ref . x := p . x ;
           Sinon
```

```
                        /* début de polygone recouvrant */
                        affichage (de ref . x à p . x - 1) ;
                        ref . x := p . x
                        insertion en tête (C, ref) ;
                        copie de elt(p) dans elt(ref) ;
                    Fsi
                Fsi
            Fsi
        Fsi
        p := p . ptr ;
    FTQ
  Fsi
fin
```

Des améliorations sont possibles :

- Recherche dans les listes.
- Utilisation de tableaux au lieu de listes (ce qui impliquerait des limitations).

L'un des inconvénients de cet algorithme est de ne pouvoir effectuer le remplissage qu'avec des couleurs uniformes, le principe n'admettant pas l'interpolation de profondeur entre deux bords d'un polygone.

L'un des remèdes possibles consiste à tenir à jour par ligne, une liste de bords de polygone constituée par les données des extrémités, permettant alors la détermination de la profondeur z en un point donné du polygone.

II.4.2.5. Remarques

Ce type de parallélisme effectué globalement permet de meilleures performances, car celui-ci autorise un traitement entièrement parallèle à tous les niveaux, c'est-à-dire lors de la constitution de la structure de données, de même qu'à son traitement.

Son implémentation est facilitée par l'aspect séquentiel au niveau de chaque processeur du traitement ligne.

De plus, cette approche présente l'avantage de favoriser l'animation. En effet, lors d'une séquence d'images, les différences entre deux images sont souvent minimes (décors fixes, translation d'objets,...) et le fait de disposer de la structure de données entièrement constituée pour une image complète permet d'éviter les traitements relatifs aux mêmes objets (interpolation redondante pour les mêmes objets, translations simples à effectuer, etc...).

II.4.3. Parallélisme lors du traitement par ligne

Pour illustrer cette seconde approche nous allons prendre comme exemple l'algorithme du z-buffer. Les conclusions relatives à cette approche pouvant s'appliquer à d'autres types d'algorithmes.

II.4.3.1. L'algorithme du z-buffer

De tout les algorithmes travaillant sur l'espace image, l'algorithme du z-buffer est le plus simple. Pour chaque pixel de l'écran, on garde en mémoire la profondeur z de l'objet qui se trouve le plus près de l'observateur.

Avec la profondeur, on garde aussi l'intensité qui devra être affichée pour représenter l'objet. A cet égard, le z-buffer est une extension de la mémoire d'images.

L'algorithme du z-buffer nécessite donc, deux tableaux, l'un pour mémoriser la profondeur, l'autre pour mémoriser l'intensité, chacun étant indexé par les coordonnées (x, y) du pixel.

Algorithme du z-buffer :

1. Pour tous les pixels de l'image, mettre profondeur (x, y) à la plus grande valeur et intensité de (x, y) à une valeur d'arrière plan.

2. Pour chaque polygone de la scène, trouver tous les pixels (x, y) qui se tiennent dans les limites du polygone projeté sur l'écran.

Pour chacun de ces pixels :

a - Calculer la profondeur z du polygone en ce point.

b - Si z est inférieur à profondeur (x, y), ce polygone est plus près de l'observateur que ceux déjà rencontrés pour ce pixel. Dans ce cas,mettre profondeur (x, y) à z et intensité (x, y) à la valeur correspondant à l'ombrage du polygone. Si par contre z est supérieur à profondeur (x, y), le polygone déjà rencontré au point (x, y) se tient plus près de l'observateur que ce nouveau polygone, aucune action n'est faite.

Une fois tous les polygones traités le tableau intensité contiendra la solution.

On peut déterminer la profondeur z d'un polygone en un point (x, y) de deux manières :

En partant de l'équation du plan :

$$Ax + By + Cz + D = 0$$

D'où

$$z = \frac{-D - Ax - By}{C} \qquad (1)$$

Maintenant, si à (x, y) correspond la valeur z_1 établie par l'équation (1) alors à $(x + \Delta x, y)$ la valeur de z est :

$$z_1 - \frac{A}{C} \Delta x$$

Le quotient $\frac{A}{C}$ est constant et Δx est égal à 1. Une simple soustraction est donc nécessaire pour déterminer la profondeur en un point (x+1, y).

Une autre méthode consiste, une fois les coordonnées des intersections du polygone avec les lignes images déterminées, à effectuer l'interpolation linéaire entre une paire de ces coordonnées pour déterminer toutes les profondeurs du pan horizontal.

L'algorithme du z-buffer peut être parallélisé d'une manière globale similaire à la méthode vue précédemment, mais vu les besoins mémoires supplémentaires qu'il requiert pour mémoriser les différents tableaux, ceci n'est pas toujours réalisable.

Pour réduire la place mémoire nécessaire, l'image peut être divisée en de nombreuses sous-images et l'algorithme du z-buffer est alors appliqué à chacune d'elles. Ce découpage peut amener à traiter alors plusieurs fois les mêmes polygones.

Une alternative à ce problème consiste à tirer parti de la cohérence qu'il peut y avoir entre deux lignes successives de l'image. Pour ce faire, il est alors nécessaire de mémoriser pour le traitement d'une ligne les informations nécessaires à la suivante :

Pour cela on utilise une table de listes indicées par la coordonnée de la ligne.

On range alors pour chaque côté des différents polygones, dans la liste correspondant à la plus grande valeur y des extrémités du côté, diverses informations relatives à celui-ci.

- Un identificateur de polygone : permettant par la suite de déterminer les pans horizontaux.

- x : la situation de l'extrémité du segment sur la ligne.

- dx : l'incrément en x par lequel la valeur change de ligne en ligne.

- dy : l'incrément en y nécessaire pour traiter l'anti-aliasing.

- Δy : le nombre d'intersections que chaque côté génèrera.

- z : la profondeur du point considéré.

- dz_1 : l'incrément $\frac{dz}{dy}$ par ligne.

- dz_2 : l'incrément $\frac{dz}{dx}$ nécessaire pour l'ombrage des points du bord situés sur la même ligne.

Il est à remarquer que cette structure diffère de celles rencontrées le plus couramment car elle réalise le traitement de la ligne pour tous les polygones simultanément ceci en vue d'optimiser le traitement parallèle.

Cette structure permet alors de déterminer à chaque nouvelle ligne une liste de pans horizontaux à traiter. Cette liste se construit à l'aide des informations relatives aux intersections triées suivant l'identificateur de polygone et ensuite par la coordonnée x, et en associant celles-ci par paires.

Une fois la ligne traitée, on effectue la mise à jour des différentes extrémités $x = x + dx$, $\Delta y = \Delta y - 1$, $z = z + dz_1$. Si $\Delta y = 0$ suppression de l'élément, et insertion des nouvelles extrémités par consultation de la table des lignes. On vérifie ensuite la cohérence du tri et on réitère l'association en couples, pour réaliser le traitement de la ligne suivante.

Parallélisation

Chaque processeur élémentaire doit prendre en charge l'ensemble des segments d'un polygone, du fait des traitements des singularités relatives à celui-ci, pour collaborer à la constitution de la table générale.

Les tris sur identificateur de polygone et sur la coordonnée x peuvent être réalisés en parallèle.

La constitution de la liste des pans horizontaux en parallèle n'est applicable qu'à l'aide d'une gestion de consommation.

En effet, il faut pouvoir déterminer quelles sont les couples d'informations traités par les autres processeurs.

Une solution, consiste alors à répartir lors du tri parallèle en x, les couples dans les mémoires locales des processeurs.

Chaque processeur peut alors traiter un pan horizontal. Pour chaque pixel de ce pan, il peut déterminer la profondeur et l'intensité, et mettre à jour les zones mémoires relatives à ces deux informations.

Il peut alors mettre les informations en sa possession à jour et ranger celles-ci en vue de la constitution de liste de pans nécessaires au traitement de la ligne suivante, et passer au pan suivant.

II.4.4. Conclusion

La comparaison des deux approches de parallélisme, c'est-à-dire l'affectation d'un traitement d'une ligne par processeur et le traitement parallèle par ligne, montre la simplicité de la première et la complexité de la seconde.

Le rapport de performance penche dans le même sens :

- La méthode globale ne nécessite aucune synchronisation et permet par l'utilisation d'une mémoire partitionnée d'éviter les conflits d'accès aux données par la répartition des informations relatives à une ligne dans un seul ban mémoire. Le problème reporté à l'initialisation de la structure de données, donc en écriture, peut alors être résolu en différé indépendamment des processeurs lors des interpolations de segments et n'influe pas sur le temps de traitement.

- Tandis que la réalisation parallèle du traitement sur une ligne, nécessite par contre des synchronisations pour chaque étape de traitement. Les performances sont encore réduites par la nécessité de réaliser des accès aux variables en exclusion mutuelle pour garantir la cohérence de l'algorithme.

L'implémentation de la première approche est facilitée de par les traitements totalement indépendants qu'elle autorise aux processeurs. Elle pose toutefois le problème des capacités mémoires, ce qui pourra conduire alors au choix de la deuxième approche.

II.5. CONCLUSION GENERALE

Nous avons décrits divers algorithmes parallèles de base nécessaires à tout logiciel graphique. Certains algorithmes sont une adaptation d'algorithmes existants, d'autres présentent une originalité intéressante de par leurs approches et leurs performances (remplissage multiple possible de contours préinscrits, tracé de cercle,...).

Ces algorithmes ne sont pas toujours utilisables tels quels sur n'importe quelle machine, mais ils peuvent être facilement adaptés ou servir de bases à d'autres développements. Certains détails propres aux architectures ont été laissés de côté, car ils sont développés dans le chapitre suivant. En effet, il est apparu au cours de cette étude, que les problèmes de compatibilité entre les algorithmes parallèles et les architectures multiprocesseurs étaient fortement dépendants des accès mémoires et de leur gestion, c'est pouquoi, ce point précis a fait l'objet d'une attention particulière.

CHAPITRE III

ACCES MEMOIRES ET CONTRAINTES ARCHITECTURALES

ACCES MEMOIRES ET CONTRAINTES ARCHITECTURALES

III.1. INTRODUCTION

Les différents algorithmes présentés montrent que l'on peut prétendre le plus souvent à un parallélisme maximal de calcul, d'où des performances n fois plus rapides lorsqu'on dispose de n processeurs.

Malheureusement les performances réelles d'une machine parallèle dépendent aussi du parallélisme relatif aux accès mémoires.

Pour des raisons fonctionnelles et logiques, la mémoire de données et la mémoire d'images sont considérées comme deux entités matérielles distinctes. Ceci est dû à la nature des informations et des échanges mais aussi à la nécessité d'opérer une interface directe entre la mémoire d'images et la visualisation.

En se basant sur ces différents accès, l'étude qui suit analyse les diverses possibilités d'optimiser les échanges d'informations avec les mémoires afin d'obtenir le minimum de contraintes algorithmiques et un taux de parallélisme global le plus élevé possible, c'est-à-dire le plus près de l'optimum possible pour les calculs.

III.2. SOLUTION IDEALE : MEMOIRE A ACCES MULTIPLES

Les problèmes rencontrés lors des conflits d'accès sont dus à l'abscence de parallélisme dans les boîtiers mémoires. En effet, les circuits rencontrés sur le marché n'admettent en entrée qu'une seule adresse et n'autorisent qu'un accès aux données. Si une mémoire acceptait simultanément plusieurs accès, les problèmes de conflits n'auraient plus de raison d'être, sauf si ces accès concernaient la même adresse.

Bien qu'un tel circuit soit considéré comme chimérique, l'apparition de nouvelles techniques telles que les très hautes intégrations (VLSI, HVLSI) où la logique optique permettent, peut-être, de reconsidérer la question.

Ces solutions ne sont cependant pas immédiatement disponibles et il convient de limiter l'étude à l'emploi de composants et de technologies plus classiques.

Examinons, tout d'abord, la logique des circuits mémoires existants, afin de déterminer celle nécessaire pour atteindre le but fixé dans une optique de très grande intégration.

III.2.1. La sélection d'une cellule mémoire

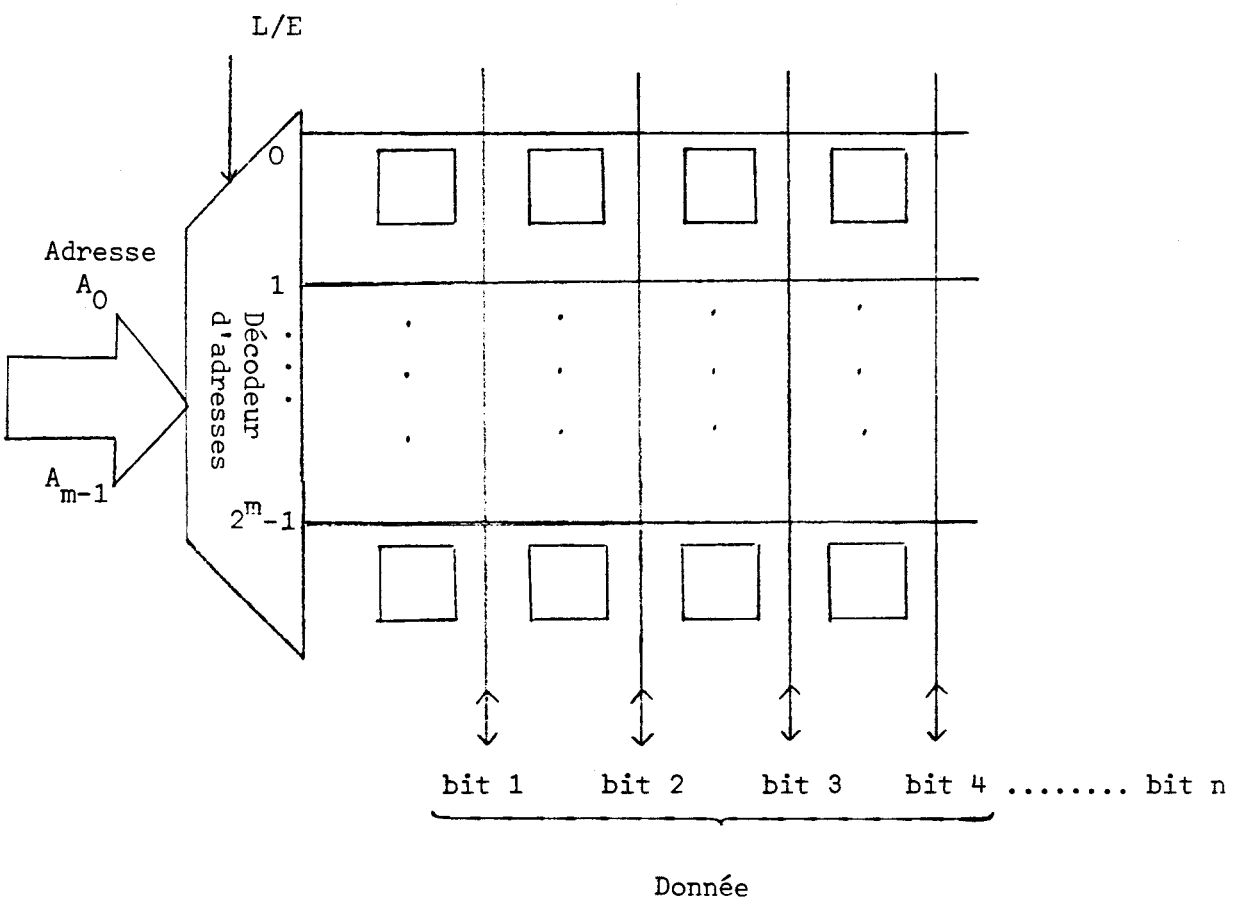
Il y a deux types différents de sélection d'une cellule dans les mémoires à semi-conducteurs.

- La sélection linéaire.

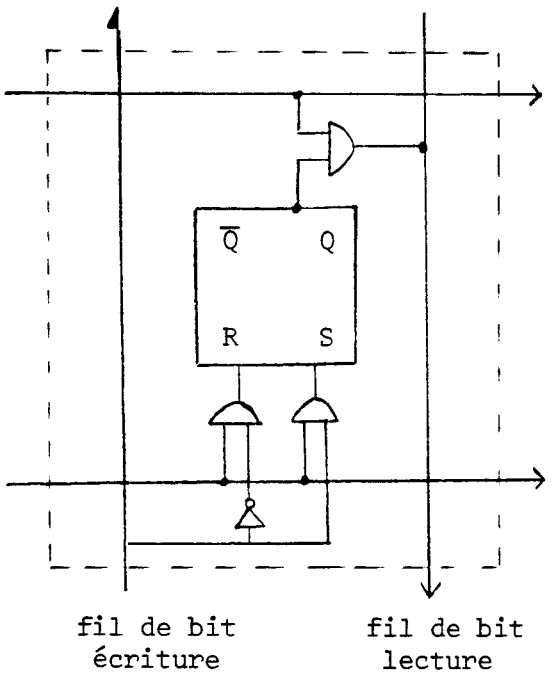
- La sélection par coïncidence ou matricielle.

- *La sélection linéaire*

Le décodeur d'adresses sélectionne une de ses sorties qui active une rangée de cellules. La structure délivre un mot mémoire.



Structure de la cellule :

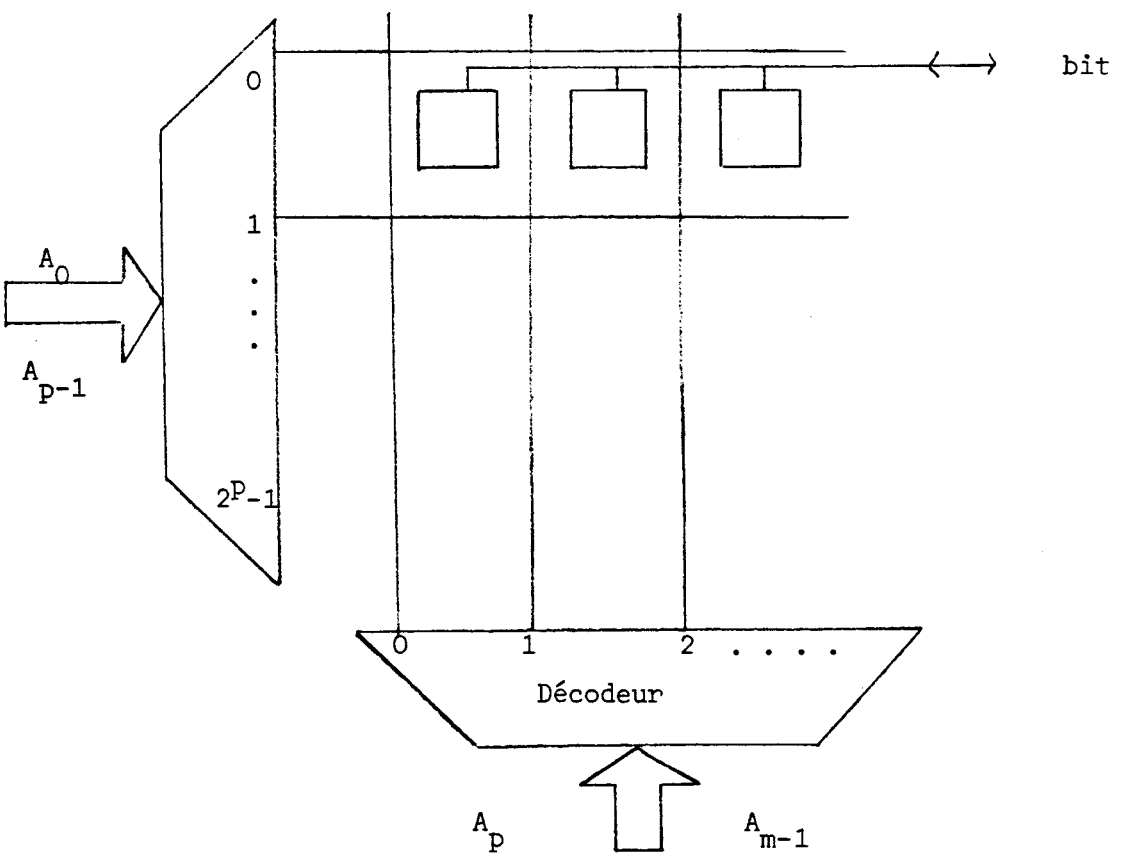


Sélection en lecture de la rangée.

Sélection en écriture de la rangée.

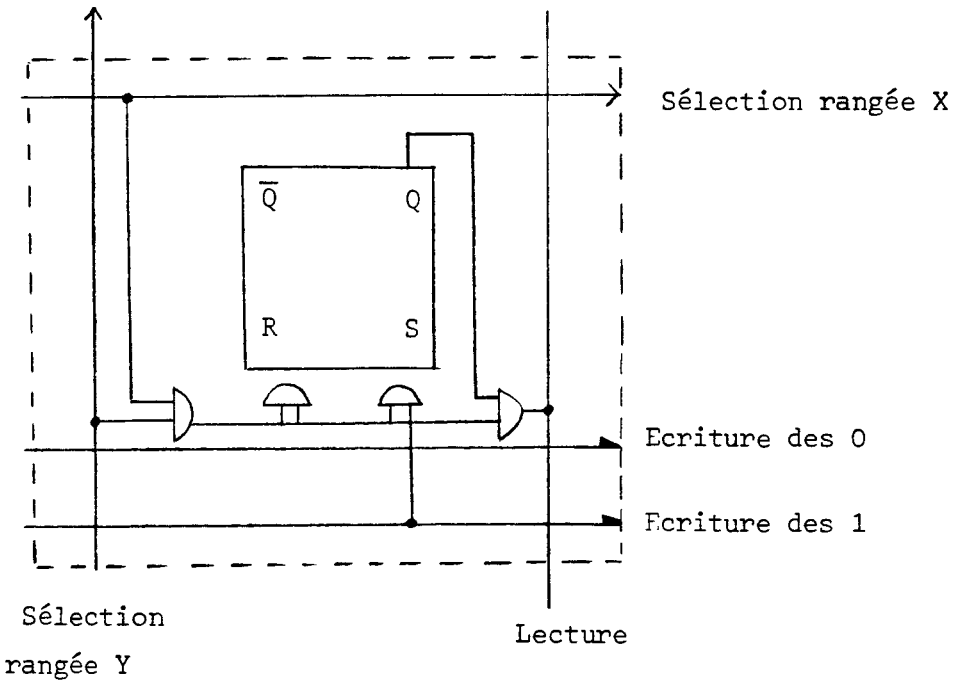
- *La sélection par coïncidence*

La cellule est directement adressée par une ligne et une colonne. la donnée issue de cette structure est un bit.



Dans la sélection par coïncidence on associe plusieurs plans mémoires pour obtenir une organisation en mots. On parle parfois de mémoire 2D pour la sélection linéaire et de mémoire 3D pour la sélection matricielle.

Structure de la cellule :

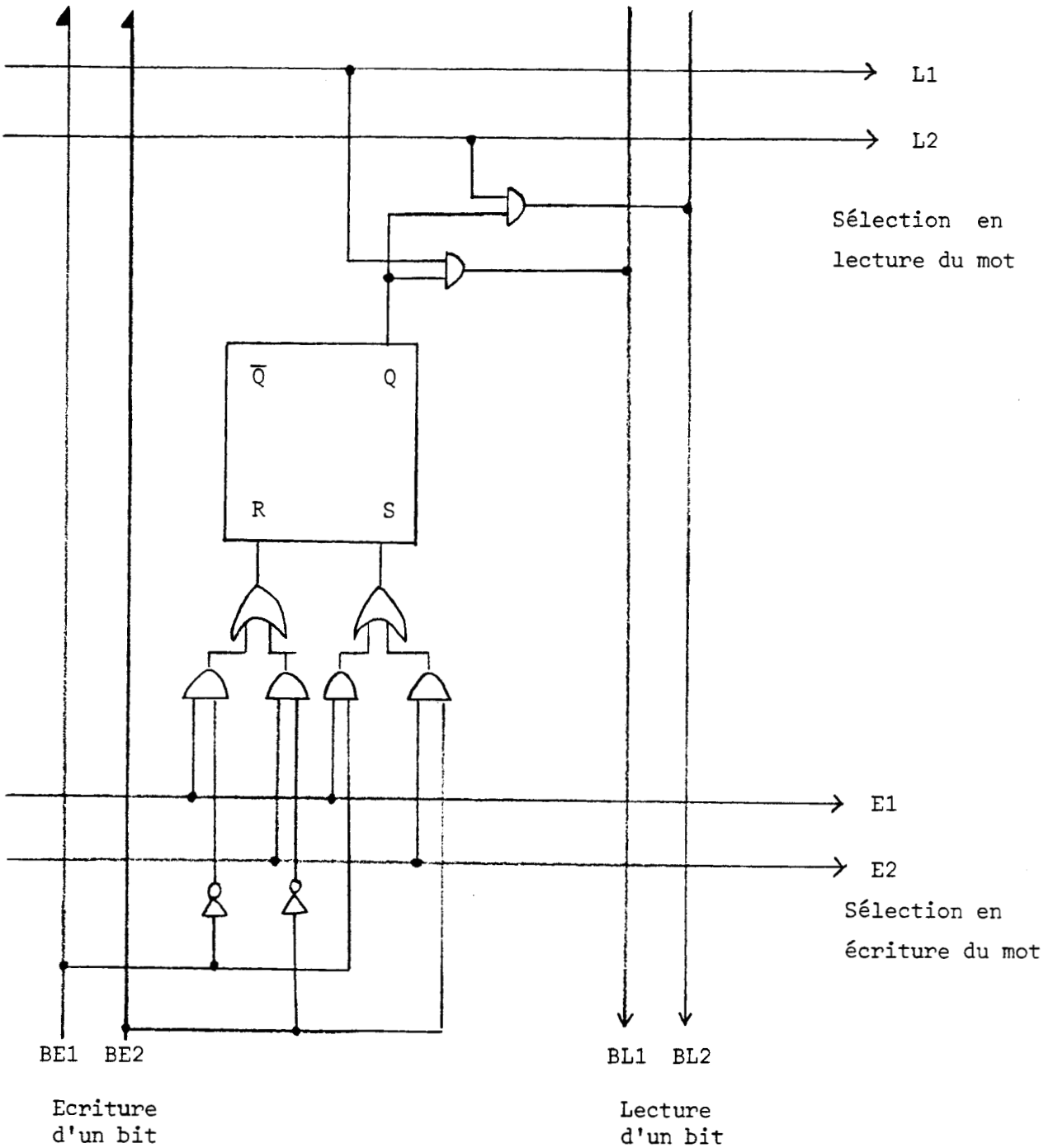


III.2.2. Circuit proposé

Il est possible, au niveau logique, de réaliser une cellule à accès multiples, le type de sélection proposé est la sélection linéaire car elle implique une structure beaucoup plus simple que celle par courants coïncidents.

Examinons la cellule nécessaire pour un accès simultané par deux processeurs :

Cellule d'une mémoire à deux accès :



On peut généraliser la structure :

Une cellule mémoire à n accès nécessite :

4n "fils"
3n opérateurs ET
2n opérateurs OU
2n inverseurs
1 bascule

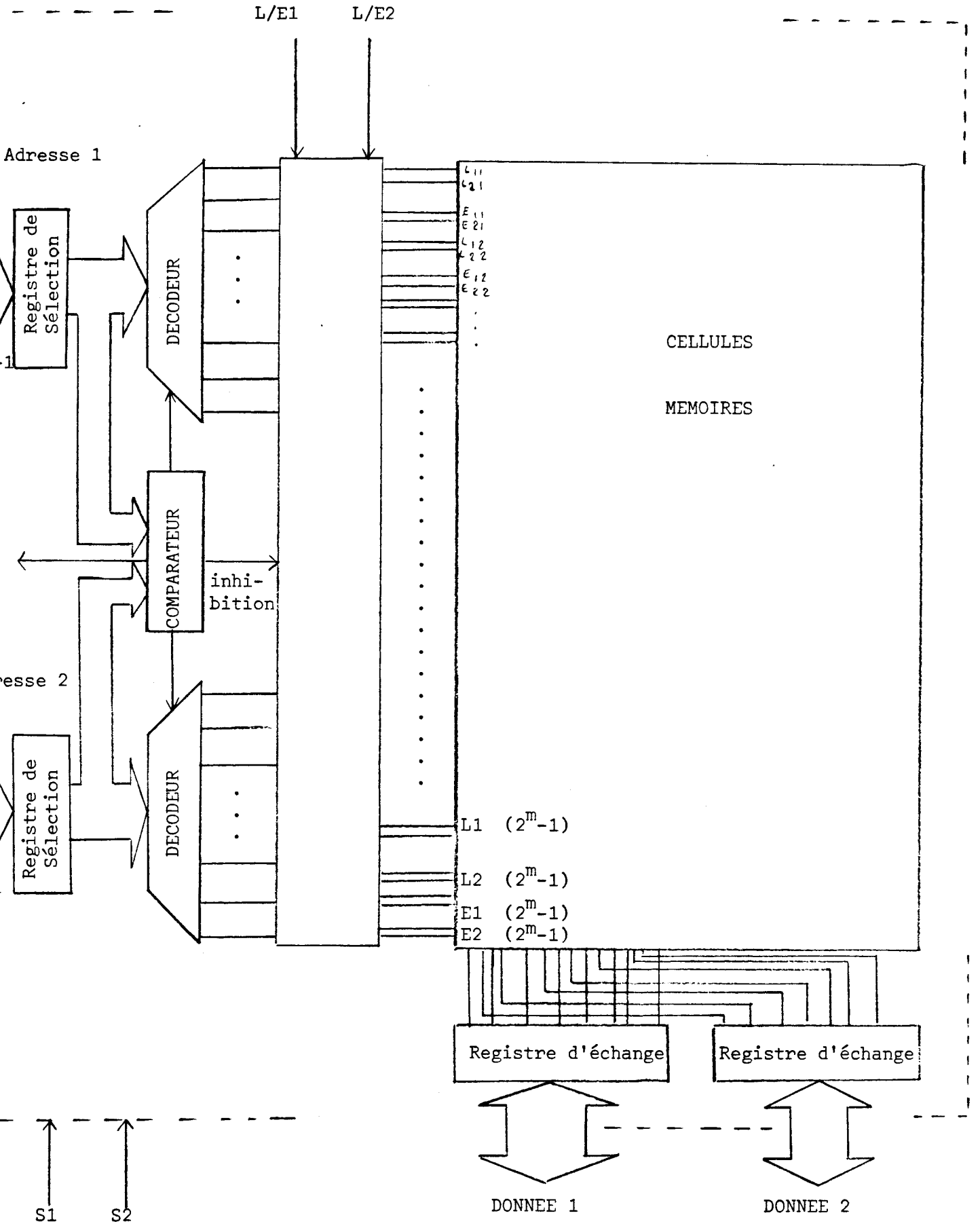
On voit que la complexité de la cellule élémentaire limite fortement le nombre d'accès possibles.

Structure du circuit global :

Un circuit composé de ces cellules admet n lectures ou écritures simultanées sauf lorsque les adresses référencent un même mot mémoire, dans ce cas la lecture est possible et sans problème, tandis que l'écriture génère un conflit.

Il conviendra donc d'adjoindre un comparateur d'adresses dont la tâche sera d'interdire les adressages simultanés d'un même mot mémoire.

Le schéma suivant correspond à un circuit général à deux accès.


L/E1
L/E2
Adresse 1
Registre de Sélection
DECODEUR
CELLULES
MEMOIRES
COMPARATEUR
inhi-
bition
esse 2
Registre de Sélection
DECODEUR
L1 (2^m-1)
L2 (2^m-1)
E1 (2^m-1)
E2 (2^m-1)
Registre d'échange
Registre d'échange
DONNEE 1
DONNEE 2
S1
S2
Sélection
du circuit

Un tel circuit reçoit ou délivre :

n × m signaux d'adressage
n × p signaux de données (p étant le format du mot mémoire)
n signaux de L/E
n signaux de sélection du circuit

Une limitation technologique réduit la portée de ces circuits, en effet, l'accès à un circuit intégré se fait par l'intermédiaire de broches via les connexions. Dans la technologie actuelle de grande intégration, le nombre maximum de broches se situe entre 150 et 200.

Pour pallier cet inconvénient, il est toujours possible de réduire l'adressage du circuit en répartissant la mémoire sur plusieurs circuits et en utilisant des bits de poids forts de l'adresse pour sélectionner le boîtier. De même, il est possible d'utiliser l'association de plusieurs boîtiers afin de réaliser le format demandé pour les mots de données.

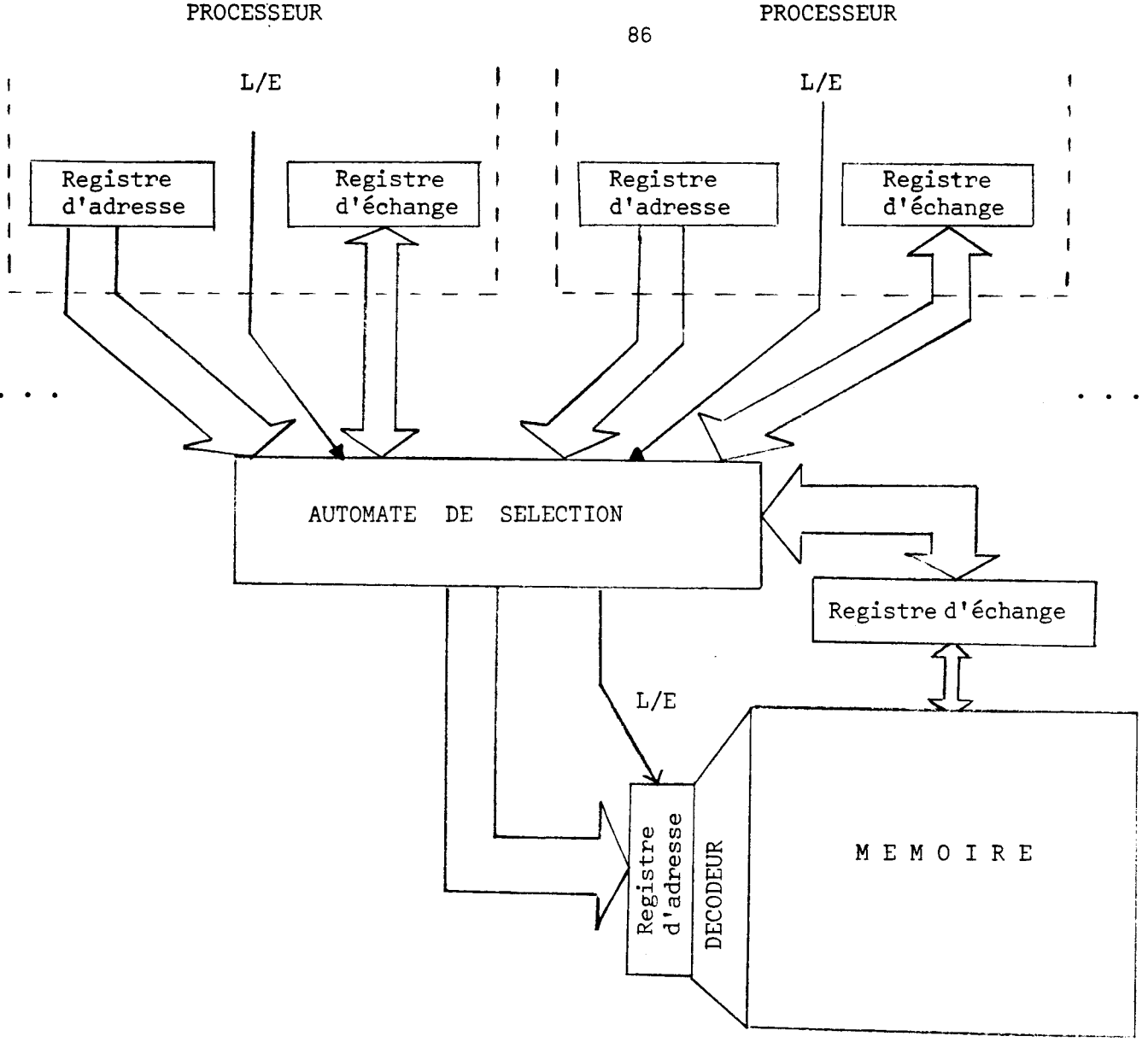
III.2.3. Remarques

Un tel type de circuit résoudrait les conflits d'accès à un même module mémoire. Le seul conflit existant serait alors l'écriture simultanée par plusieurs processeurs à une même adresse.

Bien qu'autorisant un parallélisme optimal, la réalisation de ce circuit pour un nombre important de processeurs n'est pas envisageable avec la technologie actuelle. Toutefois dans le contexte d'un nombre très réduit de processeurs, vu l'accroîssement important des performances qu'il apporterait, sa fabrication pourrait s'avérer possible et rentable.

III.3. SOLUTION DU MULTIPLEXAGE TEMPOREL

Une autre méthode pour résoudre le dilemme posé par les conflits consiste à multiplexer les accès dans le temps. Pour cela, il est nécessaire d'utiliser des mémoires ayant des temps d'accès très nettement inférieurs à la période de l'horloge pilotant les processeurs élémentaires.



Le multiplexage est transparent pour les processeurs. Chaque processeur effectue ses accès comme si il était le seul à disposer de la mémoire.

L'automate de sélection examine tout à tour les registres d'échanges des processeurs et déclenche l'accès mémoire s'il y a lieu.

Cette solution très intéressante n'est réalisable qu'avec des mémoires à accès très rapides. En fait, lorsqu'on dispose de N processeurs fonctionnant avec une période d'horloge T, le temps d'accès de la mémoire doit être inférieur à $\frac{T}{N}$ pour éviter les conflits.

Il est évident qu'à technologie égale, le nombre de processeurs ne pourra être très élevé.

III.4. PARTITION DES MEMOIRES

La méthode la plus commune utilisée pour résoudre les conflits d'accès sur les machines actuelles spécialisées en graphique consiste à effectuer une partition des mémoires.

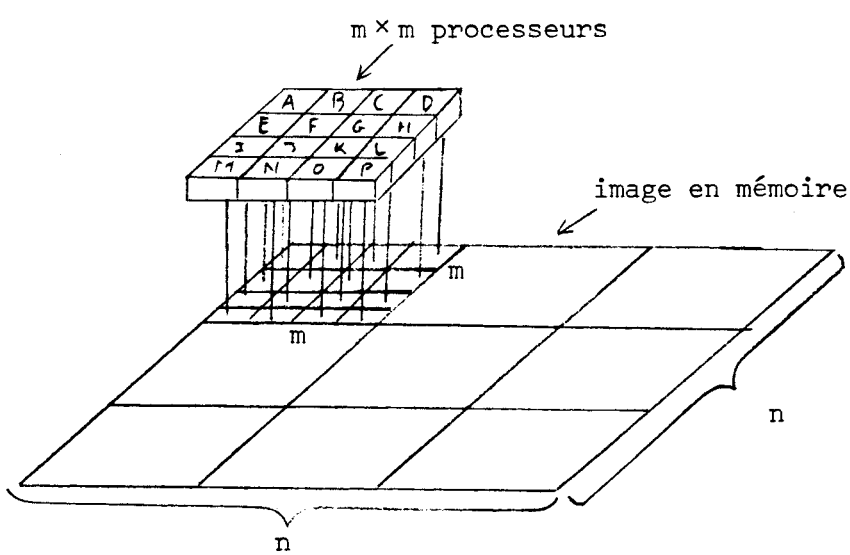
La mémoire de données est alors une mémoire partitionnée en banks accessibles par l'intermédiaire d'un réseau d'interconnections qui se charge de différer temporellement les demandes générant un conflit.

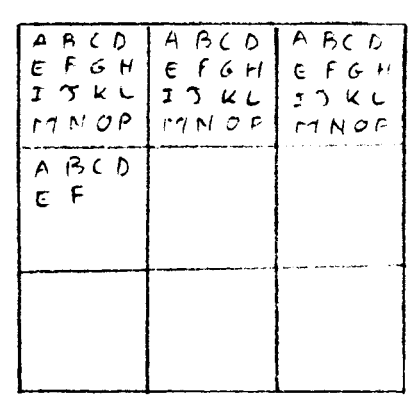
La mémoire d'image est découpée en zones géométriques, chacune de ces zones est alors affectée à un seul processeur.

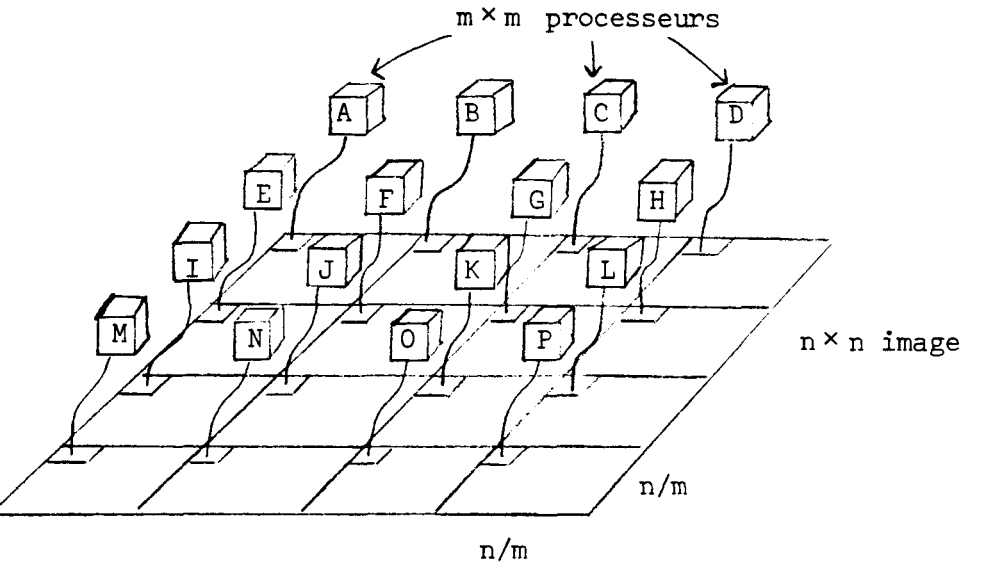
Les découpages les plus employés sont des découpages adaptés au traitemen d'images du fait de l'utilisation courante d'algorithmes (lissage, filtre, suivi de contour,...) traitant des pixels voisins.

Ils consistent à réaliser un ensemble de fenêtres, activées par sélection dans la mémoire d'images ; chaque fenêtre est alors un sous-tableau ou une sous-image est accessible séparémment.

Les deux exemples suivants tirés de [DAN 84] illustre cette méthode.







AAA AAA AAA	BBB BBB BBB		
			PPP PPP PPP

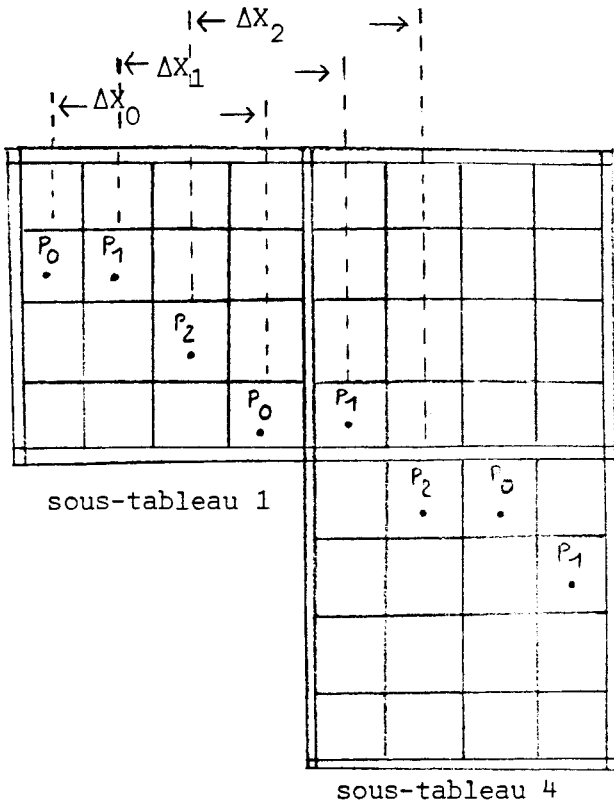
Dans ce genre de découpage, on rencontre souvent dans les algorithmes des problèmes de bord, il existe à cela deux solutions :

- Soit la possibilité de mémoriser les pixels précédemment lus et d'avoir des moyens évolués de communication interprocesseurs.

- Soit réaliser l'adressage de la mémoire d'images par recouvrement de zones.

Ces genres de découpages sont mal adaptés à la synthèse d'images car leur utilisation nécessite l'emploi d'algorithmes à faible taux de parallélisme.

Si l'on prend l'exemple d'un tracé de segment DDA//, on s'aperçoit que l'écriture en mémoire d'images ne peut s'effectuer sans perturber les performances attendues. En effet les points générés par un processeur n'ont aucune liaison avec les sous-tableaux.



Exemple

- A la 1ère étape :
  P_0, P_1, P_2 adresse le 1er sous-tableau
- A la 2ème étape :
  Il y a un problème d'adressage car
  . P_0 adresse le 1er sous-tableau
  . P_1 adresse le 2ème sous-tableau
  . P_2 adresse le 4ème sous-tableau

On doit donc effectuer trois accès à la mémoire d'images pour cette étape de traitement.

Une solution consiste à effectuer le traitement en trois phases :

- Traitement et rangement des résultats en mémoire de données.

- Tri sur critère de zone.

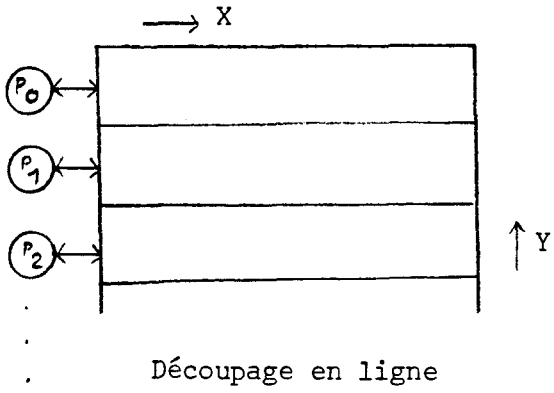
- Affichage par zone successives.

Le temps de rangement et de tri s'ajoute au temps de traitement optimal théorique.

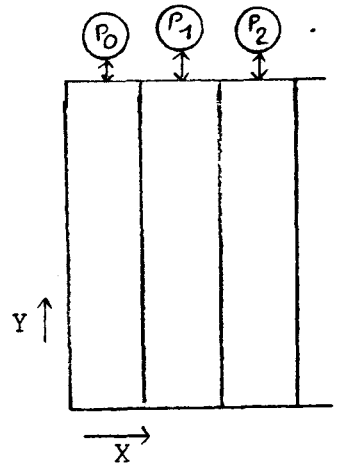
De l'étude de différents algorithmes de synthèse d'images, on peut déduire deux découpages engendrant le moins de dégradation de performances. Ces deux découpages correspondent à l'entité structurelle la plus souvent rencontrée dans les algorithmes : la ligne de mémoire d'images.

- Un découpage en ligne qui permet d'obtenir un parallélisme poussé en attribuant le traitement d'une ligne par processeur. Il présente en outre l'avantage de s'accommoder très facilement de bon nombre d'algorithmes existants en synthèse d'images.

- Un découpage en colonne permettant de réaliser un parallélisme sur le traitement d'une ligne.



Découpage en ligne



Découpage en colonne

Bien que ces deux découpages de la mémoire d'images soient les plus adaptés à la synthèse d'images, ils ne résolvent pas totalement les conflits d'adressage.

Pour obtenir le parallélisme optimal avec ce procédé de découpage, on peut tenir compte du sens des échanges d'informations processeurs-mémoire d'images du traitement de synthèse en utilisant le principe exposé ci-après.

III.5. PARTITION DES FONCTIONS D'ACCES

Il semble intéressant de s'inspirer des techniques d'échanges sur les ordinateurs classiques de moyenne ou haute gamme.

Ceux-ci utilisent pour réaliser les entrées-sorties d'informations un processeur spécialisé qui autorise la réalisation simultanée de l'exécution d'un programme et d'opérations d'échanges.

Ce principe peut être avantageusement appliqué pour les accès mémoires dans le cas de machines parallèles.

Pour cela, il est nécessaire de distinguer les deux fonctions d'accès possibles, c'est-à-dire la lecture et l'écriture. En effet, les accès mémoires en écriture peuvent être réalisés indépendamment des processeurs car ils ne génèrent pas de blocage immédiat des programmes ; par contre, il en va tout-à-fait différemment pour les accès en lecture, car dans ce cas les processeurs

attendent des informations des mémoires pour poursuivre leur tâche, les programmes sont alors bloqués.

L'écriture en mémoire peut s'effectuer indépendamment de l'exécution à l'aide d'un processeur spécialisé sur le même principe que les unités d'échanges des machines classiques, par contre pour ne pas nuire aux performances, la lecture devra s'effectuer avec un taux de parallélisme le plus élevé possible.

Cette méthode présente l'avantage de supprimer totalement les conflits d'accès en écriture.

III.5.1. Accès à la mémoire de données

L'utilisation d'un processeur spécialisé pour les accès en écriture présent en outre l'avantage de pouvoir organiser les données dans la mémoire partitionnée en vue de leur utilisation ultérieure. Cette ventilation des données dans les différents banks mémoires est alors assurée par une programmation adéquate de ce processeur correspondant à la tâche à effectuer.

Si l'on prend l'exemple d'un traitement de synthèse d'images ou chaque processeur élémentaire assure l'élaboration finale d'une seule ligne de la mémoire d'images à la fois (cf. II.4.2.). Il est nécessaire de programmer le processeur spécialisé de manière à assurer lors de la détermination des intersections de segments de polygones avec les lignes images, la répartition de celles-ci afin que les différentes intersections se rapportant à une même ligne se trouvent dans le même bank de la mémoire partitionnée. Ceci permet une exploitation entièrement parallèle des données lors de la génération de l'image, du fait de la présence dans un même bank mémoire des informations nécessaires à un processeur, chaque processeur se voit ainsi affecter pour lui seul l'accès à un bank dans cette deuxième étape de traitement. En opérant de cette manière aucun conflit d'accès n'est possible.

En ce qui concerne la lecture des données par les processeurs, on pourra le plus souvent trouver une répartition dans la mémoire partitionnée autorisant l'accès en parallèle sans conflit.

Pour permettre une grande adaptabilité d'utilisation vis-à-vis de différents traitements, il est nécessaire pour chacun des processeurs de pouvoir accéder à n'importe quel bank par l'intermédiaire d'un réseau d'interconnection.

III.5.2. Accès à la mémoire d'images

De même que pour la mémoire de données, l'accès en écriture à la mémoire d'images se réalise par l'intermédiaire d'un processeur spécialisé.

Bien que l'utilisation du même processeur puisse être envisagée pour les deux types de mémoires, il paraît plus opportun pour des raisons fonctionnelles et de simplicité de programmation d'attribuer à chacun d'eux un processeur particulier.

La lecture de la mémoire d'images, bien qu'exceptionnelle en synthèse d'images mais courante en traitement ou manipulation, pose par contre le problème de l'accès en parallèle. Il conviendra dans ce cas de déterminer la meilleure partition possible de la mémoire d'images vis-à-vis des différentes applications envisagées.

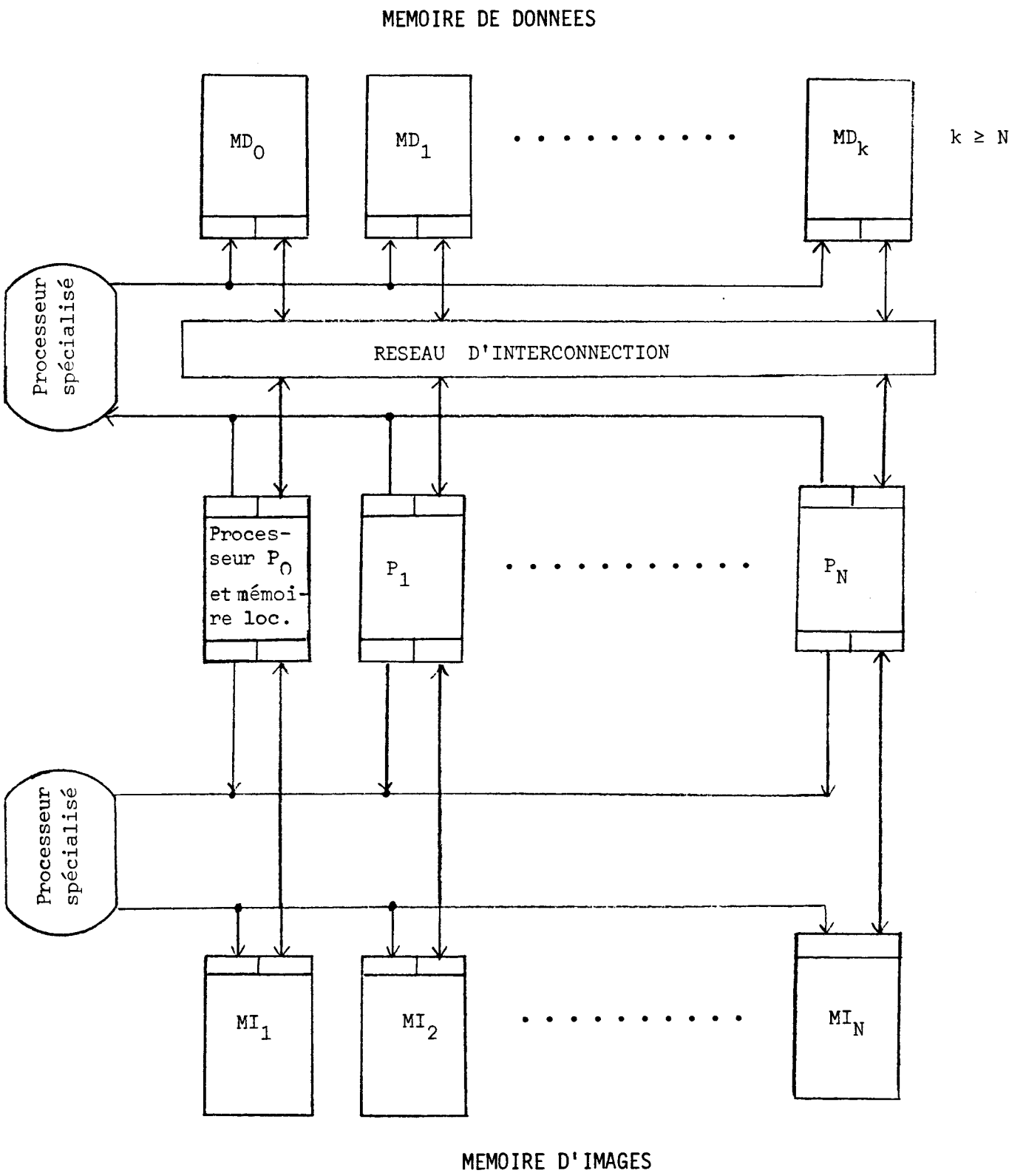


Schéma synoptique des accès mémoires

Lorsqu'un processeur doit effectuer un accès en écriture pour une donnée ou un pixel, il stocke l'information dans le buffer correspondant et poursuit son traitement. Le processeur spécialisé explore cycliquement les différents buffers des processeurs. Si une écriture est demandée, il détermine par rapport à sa programmation l'emplacement mémoire correspondant et lance alors un accès mémoire au module concerné.

Il est nécessaire de prévoir un système de files d'attente pour résoudre les situations de blocage dues aux demandes incessantes d'écriture. On peut prendre l'exemple d'un ajout de couleur à l'ensemble d'une image, car chaque processeur n'effectue alors qu'une lecture et une addition avant chaque écriture, le processeur spécialisé devant alors gérer N écritures durant le temps de deux instructions.

Une autre façon de résoudre ce problème typique à de nombreux processus de traitement ou manipulation d'images est de laisser la possibilité de choix du moyen d'accès.

Dans ces cas précis, on peut assister à une dégradation des performances du système.

III.6. CONCLUSION

SOLUTIONS	DEPENDANCE TECHNOLOGIQUE	CONTRAINTES ALGORITHMIQUES
Mémoires à accès multiples	Très forte	Non
Multiplexage temporel	Oui	Non
Partition des mémoires	Non	Oui
Partition des fonctions d'accès	Non	Non en écriture Oui en lecture

Seules les deux premières solutions, les mémoires à accès multiples et le multiplexage temporel, donnent une grande souplesse algorithmique et des performances maximales.

Elles présentent toutefois l'inconvénient d'être très dépendantes de la technologie et de limiter fortement le nombre de processeurs élémentaires, la première par le nombre important de broches nécessaires aux circuits, la seconde par le temps d'accès aux mémoires.

La partition des mémoires est la solution la plus commune du fait de son indépendance par rapport à la technologie, de sa mise en oeuvre simple et de son coût réduit. Toutefois cette structure influe fortement sur les performances par le choix des algorithmes qu'elle implique et par sa gestion lourde des accès.

La partition des fonctions d'accès représente le compromis de ces diverses solutions, elle procure une grande souplesse algorithmique et autorise dans la plupart des cas des performances optimales.

CHAPITRE IV

UN MODELE GEOMETRIQUE POUR LA REPRESENTATION DES IMAGES

UN MODELE GEOMETRIQUE POUR LA REPRESENTATION DES IMAGES

IV.1. PRESENTATION

Dans certains algorithmes utilisés en informatique graphique (synthèse d'images, reconnaissance, traitement,...) on utilise une procédure récursive de découpage de l'image en quadrants ; tel est le cas de l'algorithme de Warnock qui en est un exemple typique. Cette démarche peut même être poursuivie pour réaliser sur ce principe des machines parallèles spécialisées en traitements graphiques [TAN 83], [MER 84].

Nous présentons un moyen simple d'effectuer la conversion des coordonnées d'un pixel de l'image en coordonnées que l'on pourrait nommer quaternaires, résolvant d'une manière triviale les problèmes relatifs au partitionnement en quadrants.

Une approche géométrique inhabituelle est ensuite proposée pour permettre une résolution nouvelle des problèmes algorithmiques rencontrés lors du traitement d'images par ordinateur ; ainsi par exemple une primitive de base en synthèse d'image telle que le tracé de segment est ici traitée d'une manière simple et peu courante.

L'aspect architectural est alors abordé, afin de mieux définir les diverses organisations parallèles adaptées au modèle quaternaire.

IV.2. DECOUPAGE DE L'IMAGE

Pour résoudre un problème complexe lié à une image, tel que le remplissage, l'élimination de surfaces cachées ou la reconnaissance de formes, il est courant en informatique graphique de décomposer l'image en sous-images afin de pouvoir analyser des phénomènes simples. L'une des partitions de l'image, utilisée pour cela, est la décomposition de l'image en quatre sous-images de tailles égales, elles-mêmes décomposées de manière récursive si la décision à prendre est impossible.

La représentation d'une image est alors l'arbre quaternaire correspondant au découpage de celle-ci en quadrants ; eux-mêmes découpés s'il y a lieu, jusqu'à atteindre le niveau élémentaire (pixel ou fraction de pixel si l'on utilise une surdéfinition).

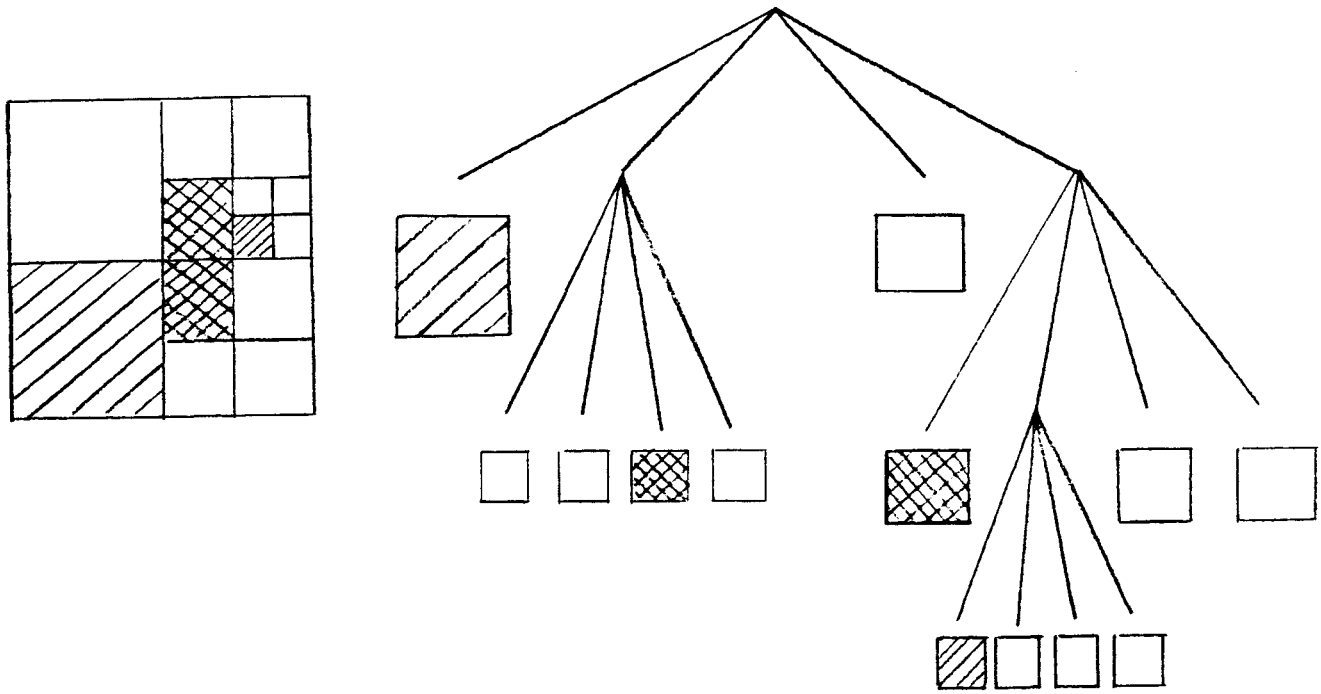
Ces découpages peuvent être employés pour construire une structure de données représentant la scène ou l'image pour différents traitements graphiques (synthèse, traitement, reconnaissance,...). [DYE 80], [FUC 80], [GAR 82], [OLI 83, 83 bis, 84, 84 bis], [SAM 80, 81], [WOO 82], [YAU 83].

Cette représentation conduit à une méthode de traitement qui contrairement à d'autres qui utilisent la cohérence d'une zone limitée (ligne, colonne, sous-tableau), permet de tirer parti de la cohérence de zones variables (quadran ou partie octaire de différents niveaux).

Une conséquence importante de ce type de cohérence est la possibilité d'un compactage important lors du stockage d'une image.

On peut en effet définir entièrement une image par la définition d'un arbre quaternaire dont la décomposition s'arrête pour une branche donnée si la cohérence de ce quadrant est établie : même couleur des pixels qui le composent ou même lien logique (exemple : diminution linéaire de la couleur).

Exemple :



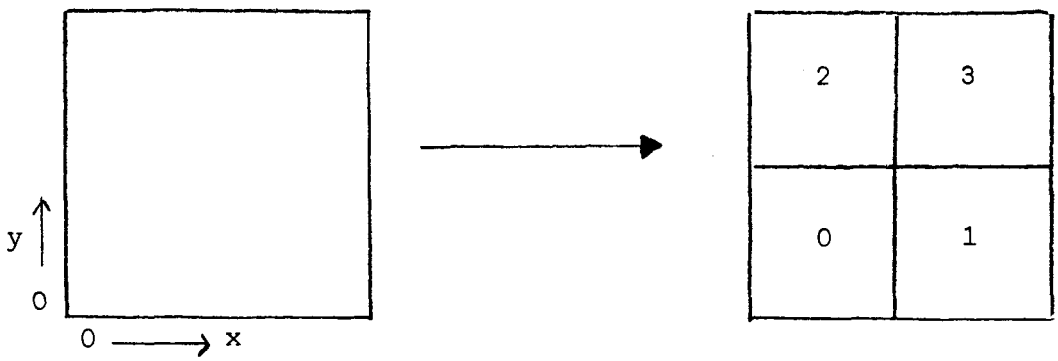
Le même principe de compactage peut s'utiliser pour le stockage tridimensionnel d'une image. On utilise alors un arbre à huit branches appelé octree.

La description et la réalisation d'une image dans le modèle quaternaire ou octaire pose différents problèmes. En effet, ce modèle ne correspond plus à la géométrie classique et il importe d'établir soit une correspondance entre le modèle cartésien et le modèle quaternaire soit des algorithmes basés sur ce dernier modèle.

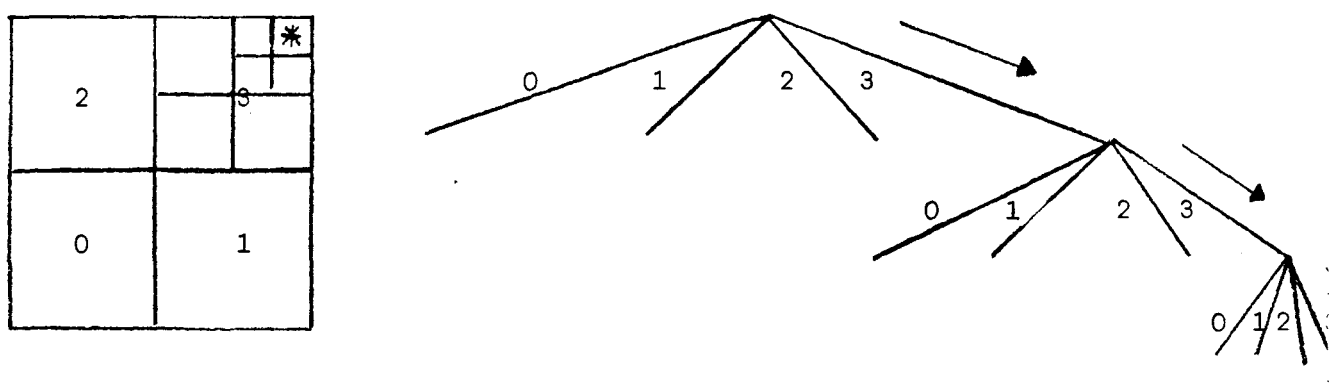
C'est ces deux approches qui seront détaillées dans les paragraphes suivants.

Pour permettre une identification des quadrants il est nécessaire de numéroter ceux-ci.

De part, la convention, il place l'origine en bas à gauche de l'image, nous numérotons les quadrants de la manière suivante :



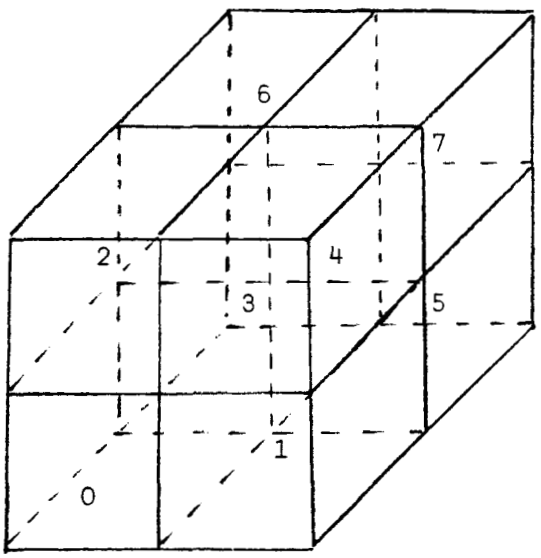
L'identification d'un pixel est réalisé par le cheminement dans l'arbre quaternaire correspondant à l'image.

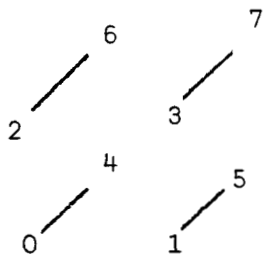


Soit dans l'exemple le chemin 3.3.3..

De la même façon, l'aspect tri-dimensionnel d'une image, la scène, peut se représenter sous forme d'un arbre ayant à chaque noeud huit descendants. On peut faire correspondre le terme "arbre octaire" au terme anglais "octree".

La numérotation dans ce cas est la suivante :





IV.3. TRANSFORMATIONS ET CODAGE

On voit l'intérêt évident qu'il y aurait à disposer d'un moyen aisé de passer des coordonnées d'un pixel ou d'un voxel (élément tri-dimensionnel) à son chemin d'accès dans l'arbre quaternaire ou octaire.

La codification suivante répond à ce but, permettant pour un pixel ou un voxel donné de déterminer d'une manière univoque le chemin d'accès dans l'arbre, c'est-à-dire son positionnement géométrique.

IV.3.1. Transformation 2D

En considérant une définition carrée de l'image exprimable en puissance de 2 (exemple 512×512, 1024×1024) un pixel est déterminé par ses deux coordonnées x et y.

Soit x, y exprimées en base 2

$$(x, y)_{10} = (x_{n-1}\ x_{n-2} \cdots\ x_0,\ y_{n-1}\ y_{n-2} \cdots\ y_0)_2$$

avec

$$(x)_{10} = x_{n-1} \times 2^{n-1} + x_{n-2} \times 2^{n-2} + \ldots + x_0 \times 2^0$$

$$(y)_{10} = y_{n-1} \times 2^{n-1} + y_{n-2} \times 2^{n-2} + \ldots + y_0 \times 2^0$$

et $(x_i, y_i) \in \{0, 1\}$ pour i : [0.. n[

Le chemin d'accès est alors défini par la suite de choix (0, 1, 2 ou 3) suivante :

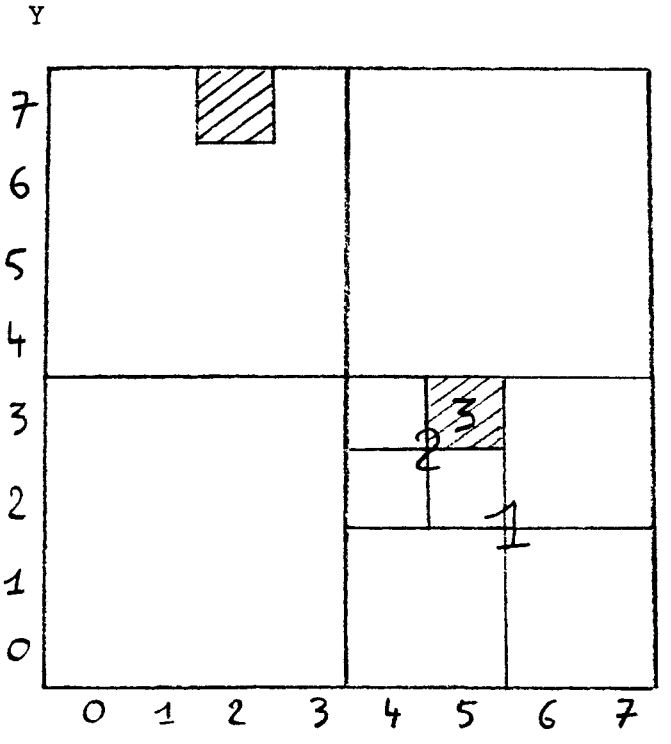
$(y_{n-1}\ x_{n-1})_2$
$(y_{n-2}\ x_{n-2})_2$
.
.
.
$(y_0\ x_0)_2$

Le code quaternaire proposé pour un pixel (x, y) est alors

$$(y_{n-1}\ x_{n-1})_2\ \ (y_{n-2}\ x_{n-2})_2 \cdots\ (y_0\ x_0)_2.$$

Le passage du code quaternaire aux coordonnées cartésiennes s'effectue par la transformation inverse.

Exemple : Pour une définition 8×8.



. pixel (5, 3)

$Y = (3)_{10} = (011)_2$
$X = (5)_{10} = (101)_2$

code quaternaire $\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}$

$(01)_2\ (10)_2\ (11)_2 = 123.$

. pixel (2, 7)

$Y = 7 = \begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}$

$X = 2 = \begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}$

code = 232

Il est facile de démontrer que l'opération de codage C est bijective.

IV.3.2. TRANSFORMATION 3D

De la même façon un voxel donné à l'aide de ses coordonnées x, y et z est exprimé par un code octaire (0 à 7)
$(z_{n-1}\ y_{n-1}\ x_{n-1})_{10}\ (z_{n-2}\ y_{n-2}\ x_{n-2})_{10} \cdots (z_0\ y_0\ x_0)$.

Exemple : Sur une définition $8 \times 8 \times 8$.

Le voxel (1, 3, 1) donne le code octaire

027

$Z = \begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}0\\1\\0\end{pmatrix}\begin{pmatrix}1\\1\\1\end{pmatrix}$

$Y =$

$X =$

IV.3.3. Problème du partitionnement

L'utilisation de cette codification permet de résoudre aisément deux des trois questions relatives au partitionnement quaternaire facilitant ainsi la résolution de problèmes algorithmiques tels que l'élimination de surfaces cachées.

1) Y a-t-il une intersection entre un segment et une droite ou un plan de partition ?

2) Dans quel quadrant ou partie octaire se trouve un sommet ?

3) Quels sont les coordonnées de l'intersection entre la droite ou le plan et le segment.

La réponse aux deux premières questions devient triviale car de par l'utilisation du code quaternaire ou octaire on sait dans quel quadrant ou partie octaire se situent les deux sommets du segment.

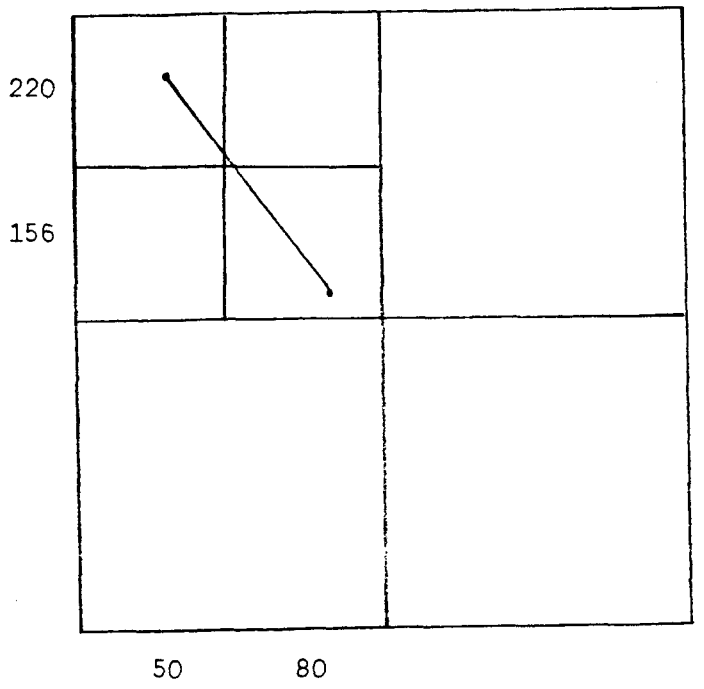
Par exemple, sur une définition d'images de 256×256 le segment S : ((50, 220), (80, 156)) nous donne en code quaternaire :

$$\begin{cases} 220 = 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\ 50 = 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \end{cases} \qquad C(50, 220) = 22310230$$

$$\begin{cases} 156 = 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \\ 80 = 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0 \end{cases} \qquad C(80, 156) = 21032200$$

d'où S : (22310230, 21032200).

En analysant de gauche à droite la suite de chiffres des deux codes représentant les extrémités du segment, on s'aperçoit qu'il n'y a pas d'intersection lors de la première partition car les deux pixels sont tout deux situés dans le deuxième quadrant ; par contre la deuxième partition génèrera au moins une intersection entre le segment et la droite de découpage du fait que le premier pixel se situe dans le deuxième sous-quadrant et le deuxième pixel dans le premier.



Pour une fenêtre ou une clôture, il conviendra de convertir les coordonnées avant l'utilisation de cette méthode.

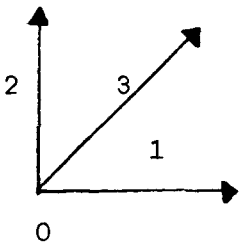
La réponse à la troisième question concernant les coordonnées des intersections peut être obtenue par une méthode de tracé de segment ou de calcul d'intersections.

IV.4. APPROCHE GEOMETRIQUE 2D

IV.4.1. Code et mouvement

Soit la codification suivante des mouvements positifs élémentaires permettant de passer à un pixel voisin :

0 : mouvement nul
1 : mouvement horizontal
2 : mouvement vertical
3 : mouvement diagonal



Le codage quaternaire d'un pixel exprime alors la suite de mouvements à effectuer depuis l'origine pour atteindre ce pixel.

Si l'on a le code quaternaire : $a_{n-1}\ a_{n-2} \cdots\ a_0$ on obtient les mouvements suivants :

2^{n-1} mouvement a_{n-1}
2^{n-2} mouvement a_{n-2}
.
.
.
2^0 mouvement a_0

avec $a_{n-1}, \ldots, a_0 \in \{0, 1, 2, 3\}$.

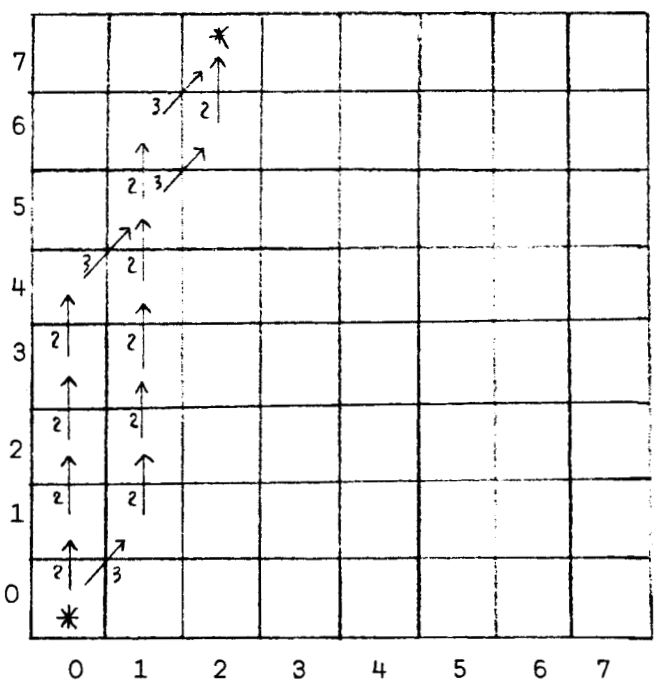
Avec une utilisation tenant compte d'une répartition quelconque de ces mouvements on atteint toujours le même pixel.

Exemple : définition 8×8.

C(2, 7) = 232

On doit effectuer quatre mouvements 2, deux mouvements 3 et un mouvement 2.

Le chemin d'accès dépend alors de la manière d'arranger cette suite de mouvements, par exemple : 2222332, 3222223, etc...



Il serait intéressant d'utiliser une répartition permettant par son application de passer par tous les pixels représentant la meilleure approximation du segment réel (origine, pixel considéré).

En tenant compte de la symétrie d'un tracé de segment par rapport au point milieu et de la conservation de cette propriété dans une approche dichotomique, il est nécessaire de répartir les mouvements de manière également symétrique.

Si l'on considère un code quaternaire : $a_{n-1}\ a_{n-2} \cdots\ a_0$, la seule méthode valable dans tous les cas quelles que soient les valeurs des a_i, avec $i \in [0..\ n-1]$, pour préserver la symétrie est une lecture particulière de l'arbre binaire représentant les mouvements, celui-ci étant construit de manière qu'à un niveau j de l'arbre soient associés les 2^j mouvements a_j.

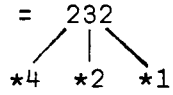
La lecture utilisée est celle qui détermine l'ordre infixé (ou projectif ou symétrique), elle consiste à :

- Enumérer le sous-arbre gauche.

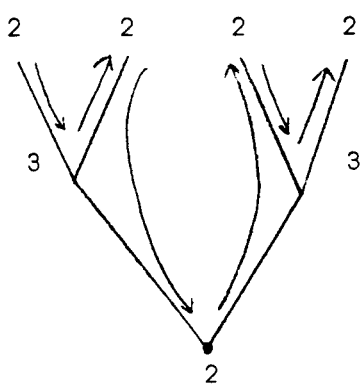
- Enumérer la racine.

- Enumérer le sous-arbre droit.

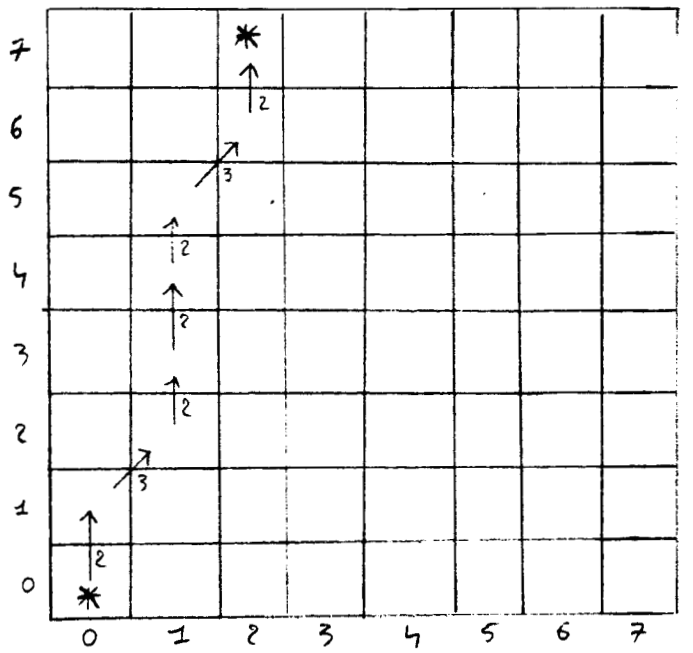
<u>Exemple</u> : Définition 8 × 8 avec C(2, 7) = 232



L'arbre binaire obtenu est :



avec la lecture infixée on obtient la suite des mouvements : 2322232.



Les pixels parcourus correspondent à ceux obtenus par un algorithme classique de tracé de segment.

Il est à noter qu'une suite de mouvements 2 et 1 doit être convertie lors de sa rencontre en mouvement 3 afin de respecter la convention d'arrondi des méthodes graphiques classiques.

En effet, le choix d'un pixel pour une valeur décimale se fait en arrondissant à la valeur entière la plus proche.

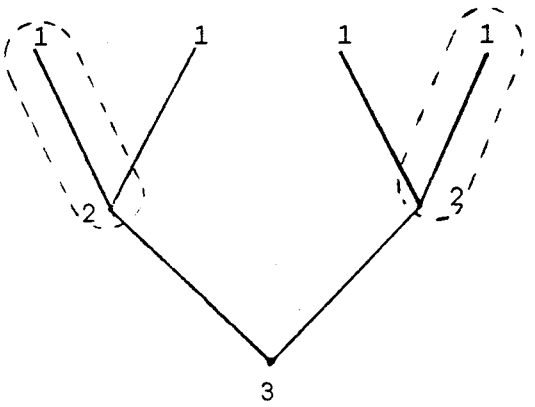
Exemple : 4,6 = 5 et 4,4 = 4

Pb : 4,5 = 4 ou 5 on choisit la valeur inférieure par convention.

Le choix pour la valeur limite est arbitraire (exemple : 4 pour 4,5), la méthode décrite précédemment choisit les deux valeurs possibles.

Si l'on choisit de respecter la convention il est nécessaire lors de la transformation de deux mouvements consécutifs 2 et 1 ou 1 et 2 en mouvement 3 d'effectuer la transformation symétrique.

Exemple : C(5, 3) = 123.

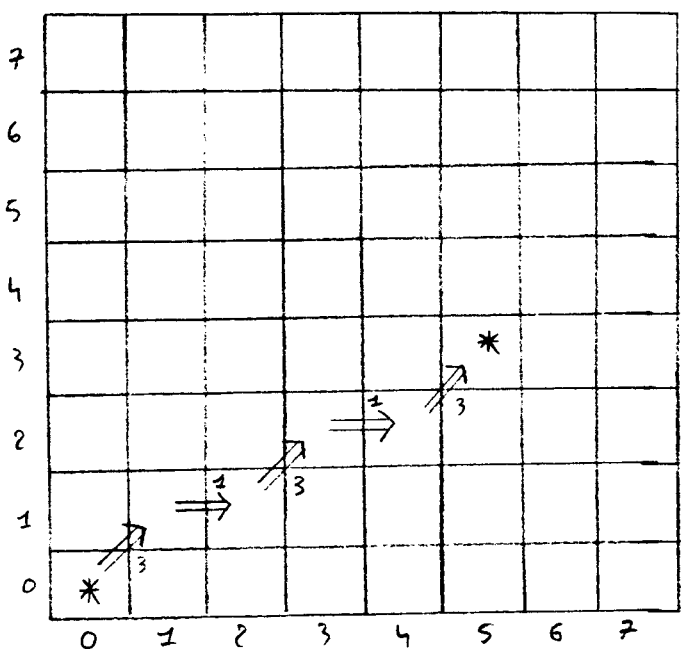


Mouvements à générer

121 3 121

ou pour respecter la convention

3 1 3 1 3



IV.4.2. Ordre dans l'ensemble quaternaire

Soit EQ l'ensemble des codes quaternaires des éléments de l'espace image I. EQ = C(I).

On peut munir EQ d'une relation L.

Soit $((a_1\ a_2 \ldots\ a_n), (b_1\ b_2 \ldots\ b_n)) \in Q^2$

et L définie par :

ou bien $a_1 < b_1$

ou bien il existe un indice $h < n$:

$a_i = b_i$ pour tout $i < h$ et $a_h < b_h$

ou bien

$a_i = b_i$ pour tout $i < n$ et $a_n \leq b_n$.

On peut constater que L est une relation d'ordre total, c'est un ordre lexicographique.

Avec $0 < 1 < 2 < 3$.

Par la suite la relation L sera notée $\leq$:

On peut établir la correspondance de cet ordre dans I :

$C(x_1, y_1) \leq C(x_2, y_2)$

$\Longleftrightarrow y_1 < y_2$ ou $(y_1 = y_2$ et $x_1 \leq x_2)$.

IV.4.3. Opération d'addition

On peut définir sur l'ensemble quaternaire une opération d'addition $\oplus$.

Pour déterminer cette opération on peut se placer sur un quadrant donné et déterminer en appliquant un mouvement sur quel quadrant on arrive, la retenue indiquant alors un quadrant de niveau supérieur.

Une autre méthode consiste à réaliser l'addition binaire par couple et de constituer un couple binaire avec les deux retenues engendrées.

En code quaternaire

⊕	0	1	2	3
0	0	1	2	3
1	1	0 (1)	3	2 (1)
2	2	3	0 (2)	1 (2)
3	3	2 (1)	1 (2)	0 (3)

Equivalent binaire

⊕	00	01	10	11
00	00	01	10	11
01	01	00 (01)	11	10 (01)
10	10	11	00 (10)	01 (10)
11	11	10 (01)	01 (10)	00 (11)

La partie □ spécifie la retenue à reporter au rang suivant.

Exemple :

$$\begin{array}{r} 2\ 1\ 3 \\ \oplus\ 0\ 2\ 1 \\ \hline = 3\ 2\ 2 \end{array} \quad \Longleftrightarrow \quad \begin{array}{r} 10\ \ 01\ \ 11 \\ \oplus\ 00\ \ 10\ \ 01 \\ \hline = 11\ \ 10\ \ 10 \\ \ \ \ 3\ \ \ 2\ \ \ 2 \end{array}$$

IV.4.4. Homomorphisme C

La fonction de codage C de (I, +) dans (EQ, ⊕) est un homomorphisme.

Soit

$$C(x_1, y_1) \oplus C(x_2, y_2) = C(x_1 + x_2, y_1 + y_2)$$

Démonstration : Soient les points : A : (a, b) B : (c, d)

avec

$$(a)_{10} = (a_{n-1} \cdots a_0)_2 \quad (b)_{10} = (b_{n-1} \cdots b_0)_2$$
$$(c)_{10} = (c_{n-1} \cdots c_0)_2 \quad (d)_{10} = (d_{n-1} \cdots d_0)_2.$$

Notons r_{ef_n} la retenue engendrée au rang n lors de l'addition des éléments binaires e_n et f_n et de la retenue $r_{ef_{n-1}}$.

- $C(a, b) = (b_{n-1}\ a_{n-1})\ (b_{n-2}\ a_{n-2}) \cdots (b_0\ a_0)$

  $C(c, d) = (d_{n-1}\ c_{n-1})\ (d_{n-2}\ c_{n-2}) \cdots (d_0\ c_0)$

$$C(a, b) \oplus C(c, d) = ((b_{n-1} + d_{n-1} + r_{bd_{n-2}})\ (a_{n-1} + c_{n-1} + r_{ac_{n-2}}))$$

$$((b_{n-2} + d_{n-2} + r_{bd_{n-3}})\ (a_{n-2} + c_{n-2} + r_{ac_{n-3}})) \cdots$$

$$((b_0 + d_0)\ (a_0 + c_0)).$$

- $(a+c)_{10} = ((a_{n-1} + c_{n-1} + r_{ac_{n-2}})\ (a_{n-2} + c_{n-2} + r_{ac_{n-3}}) \cdots$

$$\cdots\ (a_0 + c_0))_2$$

- $(b+d)_{10} = ((b_{n-1} + d_{n-1} + r_{bd_{n-2}})\ (b_{n-2} + d_{n-2} + r_{bd_{n-3}}) \cdots$

$$\cdots\ (b_0 + d_0))_2.$$

$$C(a+c, b+d) = ((b_{n-1} + d_{n-1} + r_{bd_{n-2}})\ (a_{n-1} + c_{n-1} + r_{ac_{n-2}}))$$

$$((b_{n-2} + d_{n-2} + r_{bd_{n-3}})\ (a_{n-2} + c_{n-2} + r_{ac_{n-3}})) \cdots$$

$$\cdots\ ((b_0 + d_0)\ (a_0 + c_0)).$$

On a bien :

$$C(a, b) \oplus C(c, d) = C(a + c, b + d).$$

IV.4.5. Distance dans EQ

Soient les deux points A : (x_A, y_A) B : (x_B, y_B)

$$\Delta X = |x_B - x_A|$$
$$\Delta Y = |y_B - y_A|$$

Posons comme distance d d'un point A à un point B le code quaternaire des différentielles ΔX et ΔY :

$$d(A, B) = C(\Delta X, \Delta Y).$$

Montrons que d est bien une distance :

1) $d(A, A) = C(\Delta X, \Delta Y)$
$= C(|x_A - x_A|, |y_A - y_A|)$
$= C(0, 0) = 0...0$

2) $d(A, B) = C(\Delta X, \Delta Y)$
$= C(|x_B - x_A|, |y_B - y_A|)$
$= C(|x_A - x_B|, |y_A - y_B|)$
$= d(B, A)$

3) $d(A, B) + d(B, C)$ (1)

$$(1) = C(|x_B - x_A|, |y_B - y_A|) \oplus C(|x_C - x_B|, |y_C - y_B|)$$

d'après la propriété d'homomorphisme de C.

$$(1) = C(|x_B - x_A| + |x_C - x_B|, |y_B - y_A| + |y_C - y_B|)$$

or

$$|x_B - x_A| + |x_C - x_B| \geq |x_C - x_A|$$

et

$$|y_B - y_A| + |y_C - y_B| \geq |y_C - y_A|$$

d'après l'ordre de l'ensemble quaternaire EQ.

On déduit :

$$C(|x_B - x_A| + |x_C - x_B|, |y_B - y_A| + |y_C - y_B|) \geq C(|x_C - x_A|, |y_C - y_B|)$$

$\Longleftrightarrow$ d(A, B) + d(B, C) $\geq$ d(A, C).

d est donc une distance sur l'espace quaternaire. EQ est un espace métrique.

IV.4.6. Opération de soustraction

De la même façon que pour l'opération d'addition, on peut définir une opération de soustraction sur EQ.

En code quaternaire

⊖	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Equivalent binaire

⊖	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

Exemple :

```
    3 0 2                  11  00  10
(-) 0 2 0              (-) 00  10  00
    (2)                    (1)
  = 1 2 2                = 01  10  10
```

$$\begin{cases} 2 - 0 = 2 \\ 0 - 2 = 2 \text{ retenue } (2) \\ 3 - ((2) + 0) = 3 - 2 = 1 \end{cases}$$

```
    3 2 2                  3 1 3
(-) 0 2 1              (-) 0 2 2
    (1)(1)                 (2)
  = 2 1 3                = 1 3 1
```

Soit deux points A : (x_A, y_A) B : (x_B, y_B)

Si $x_B - x_A \geq 0$ et $y_B - y_A \geq 0$ alors

$d(A, B) = C(B) \ominus C(A)$.

IV.4.7. Un exemple d'application : Le tracé de segment

- L'opérateur d'addition permet de déterminer la position résultante lorsque l'on ajoute une suite de mouvements à une position donnée.

- La distance sert à déterminer la suite de mouvements à effectuer pour passer d'une position donnée à une autre.

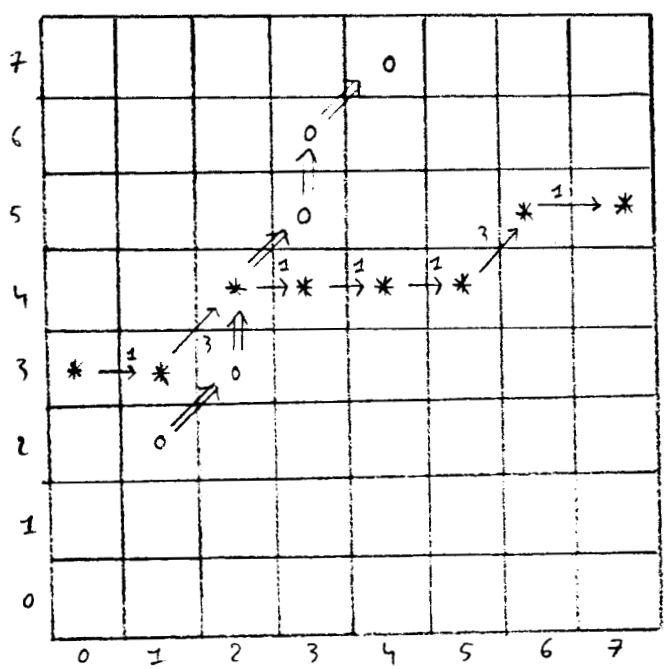
- L'opérateur de soustraction donne la suite de mouvements à générer pour le tracé d'un segment de pente positive et le sens général des mouvements par sa retenue finale dans les autres cas.

IV.4.7.1. Pente positive

Pour réaliser le tracé d'un segment, il faut déterminer la suite de mouvements à effectuer d'une extrémité à l'autre.

Du fait de la pente positive de la droite support du segment l'utilisation de la distance d ou une simple soustraction est possible, le tracé s'effectue par la lecture symétrique du code résultant converti en mouvements 0, 1, 2 ou 3.

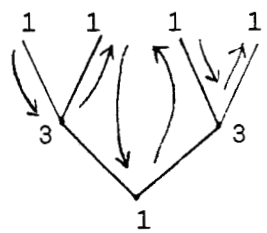
Exemples : Définition 8 × 8.



. S((0, 3), (7, 5))

C(S) = (022 ; 313)

313
⊖ 022
= 131



Suites des mouvements générant le segment S

131 1 131.

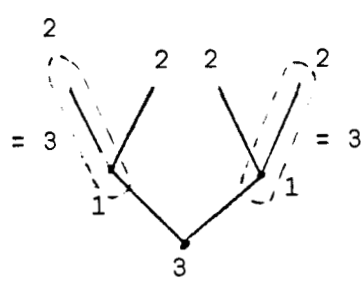
. De même, en utilisant la distance

$\Delta Y = 2$
$\Delta X = 7$
$\Rightarrow$ C(ΔX, ΔY) = 131

. S((1, 2) ; (4, 7))

C(S) = (021, 322)

322
⊖ 021
= 213

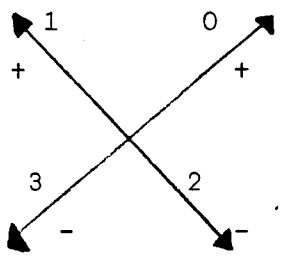


Suites des mouvements : 32323.

IV.4.7.2. Pente négative

Lors de la soustraction en vue de déterminer les mouvements, il se produit parfois une retenue finale différente de 0. (0 étant ici l'absence de retenue).

Cette retenue indique le sens général des mouvements de la manière suivante :



Exemple : S : ((7, 5) ; (0, 3)) ↙

donne C(S) : (313 ; 022)

$$\begin{array}{r} 022 \\ \ominus\ 313 \\ \hline = \underline{3}\,\overline{|221} \end{array}$$

$$\begin{cases} a + \bar{a} = 0 \\ \underline{3}\,|221 \oplus \underline{0}\,|131 = 3\,\boxed{0}\,000 \end{cases}$$

. S : ((0, 7) ; (3, 1)) ↘

donne C(S) : (222 ; 013)

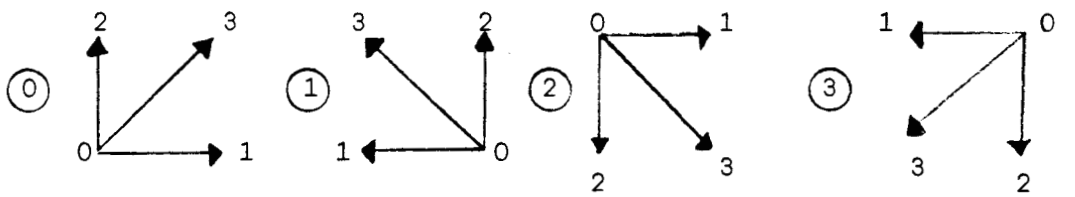
$$\begin{array}{r} 013 \\ \ominus\ 222 \\ \hline = \underline{2}\,\overline{|031} \end{array}$$

$$\underline{2}\,|031 + \underline{1}\,|321 = 3\,\boxed{0}\,000$$

Dans ces cas précis le code quaternaire résultant ne correspond plus aux mouvements définis. Bien qu'à chaque code on puisse trouver un inverse de la retenue en vis-à-vis $3 \leftrightarrow 0$, $2 \leftrightarrow 1$, il reste à déterminer un moyen de conversion pour chaque code en code sans retenue.

Pour générer le segment, on peut utiliser la retenue finale de la soustraction et la distance :

- La retenue indique alors le sens des mouvements :



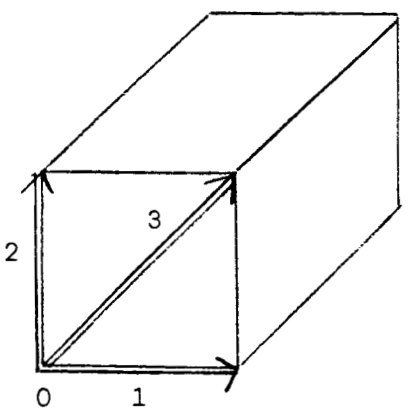
- La distance : Les mouvements à effectuer.

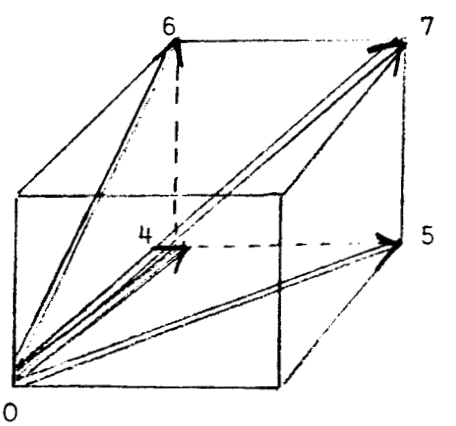
IV.5. APPROCHE GEOMETRIQUE 3D

La même démarche peut être menée sur l'ensemble octaire.

IV.5.1. Mouvements

Aux quatre mouvements plans 0, 1, 2 et 3 s'ajoutent les mouvements "de profondeur" 4, 5, 6 et 7.





Le codage octaire d'un voxel (x, y, z) exprime la suite de mouvements à effectuer depuis l'origine pour atteindre cette position.

Avec le code octaire $O_{n-1}\ O_{n-2} \ldots\ O_0$ on obtient les mouvements :

2^{n-1} mouvements O_{n-1}
2^{n-2} mouvements O_{n-2}
.
.
.
2^0 mouvements O_0

avec $O_{n-1}, \ldots, O_0 \in \{0, 1, 2, 3, 4, 5, 6, 7\}$.

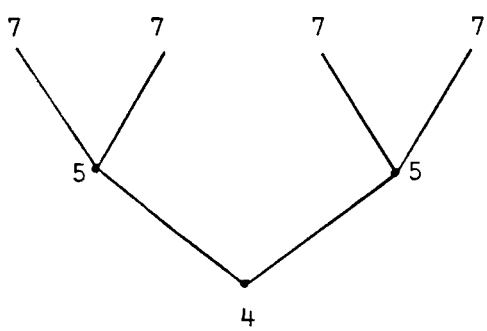
Le segment origine voxel considéré est la lecture symétrique arborescente du code octaire :

Exemple : voxel (6, 4, 7) sur définition $8 \times 8 \times 8$.

$z = 7 = (1\ \ 1\ \ 1)_2$
$y = 4 = (1\ \ 0\ \ 0)_2$
$x = 6 = (1\ \ 1\ \ 0)_2$

Code octaire =
111 101 100 = 754

L'arbre des mouvements correspondant est donc :



La suite de mouvements représentant l'interpolation 3D est alors :
757 4 757.

IV.5.2. Distance dans l'espace octaire : (EO)

De la même façon que pour l'espace quaternaire, on peut prendre l'ordre lexicographique et montrer que la fonction de codage C' est un homomorphisme de (I, +) dans (EO, ⊕).

La distance dans cet espace est alors :

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

IV.5.3. Opérations de soustraction et d'addition dans EO

Les opérations d'addition et de soustraction se déduisent de la même façon que pour l'espace quaternaire.

On obtient alors :

. Pour l'addition En équivalent binaire

⊕	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
001	001	000	011	010	101	100	111	110
010	010	011	000	001	110	111	100	101
011	011	011	001	000	111	110	101	100
100	100	101	110	111	000	001	010	011
101	101	100	111	110	001	000	011	010
110	110	111	100	101	010	011	000	001
111	111	110	101	100	011	010	001	000

En code octaire

⊕	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

. Pour la soustraction

En équivalence binaire

⊖	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
001	001	000	011	010	101	100	111	110
010	010	011	000	001	110	111	100	101
011	011	010	001	000	111	110	101	100
100	100	101	110	111	000	001	010	011
101	101	100	111	110	001	000	011	010
110	110	111	100	101	010	011	000	001
111	111	110	101	100	011	010	001	000

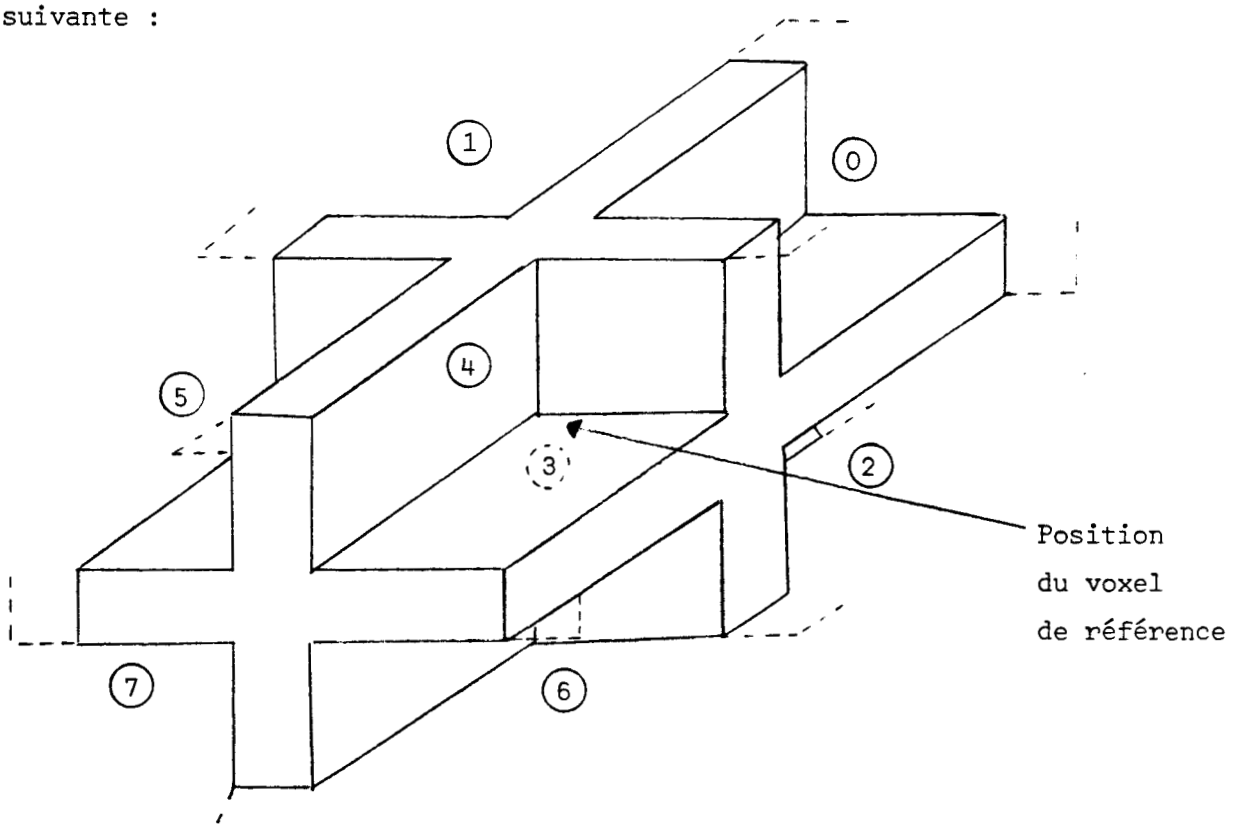
En code octaire

−	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

IV.5.4. Position relative de deux voxels

Pour déterminer la positive relative d'un voxel par rapport à un autre, on se sert de la distance d', et de la retenue générée lors de la soustraction des deux codes.

La relation directionnelle (exemple : devant - au dessus - à droite - derrière - à gauche) est donnée par la valeur de la retenue de la manière suivante :



Exemples : Sur définition 8 × 8 × 8.

Voxel de référence : (4, 5, 3) <=> 346.

Voxels	Code	Soustraction
(6, 7, 5)	736	736 - 346 = 0 070
(2, 7, 5)	636	636 - 346 = 1 170
(6, 7, 1)	336	336 - 346 = 4 170
(2, 7, 1)	236	236 - 346 = 5 170
(6, 3, 5)	536	536 - 346 = 2 270
(2, 3, 5)	436	436 - 346 = 3 370
(6, 3, 1)	136	136 - 346 = 6 670
(2, 3, 1)	036	036 - 346 = 7 770

IV.5.5. Exemples d'applications

De même que dans l'espace 2D, on peut réaliser dans l'espace octaire (3D) l'interpolation linéaire d'un segment de droite à l'aide de la distance et d'une soustraction. La retenue finale générée par la soustraction indique alors la rotation à appliquer au référentiel des mouvements.

Bien que l'élimination des surfaces cachées puisse se réaliser à l'aide de la distance dans l'espace octaire et par projection sur l'espace quaternaire xy, on peut réaliser cette opération par une lecture particulière de l'arbre octaire, cette méthode est développée dans [SAM 84].

IV.6. PERSPECTIVES

Ce qui a été développé précédemment n'est bien sûr qu'une ébauche de cette nouvelle géométrie, de nombreux développements restant à faire : nombres décimaux, opérateurs complexes, équations de droite et de plans, etc...

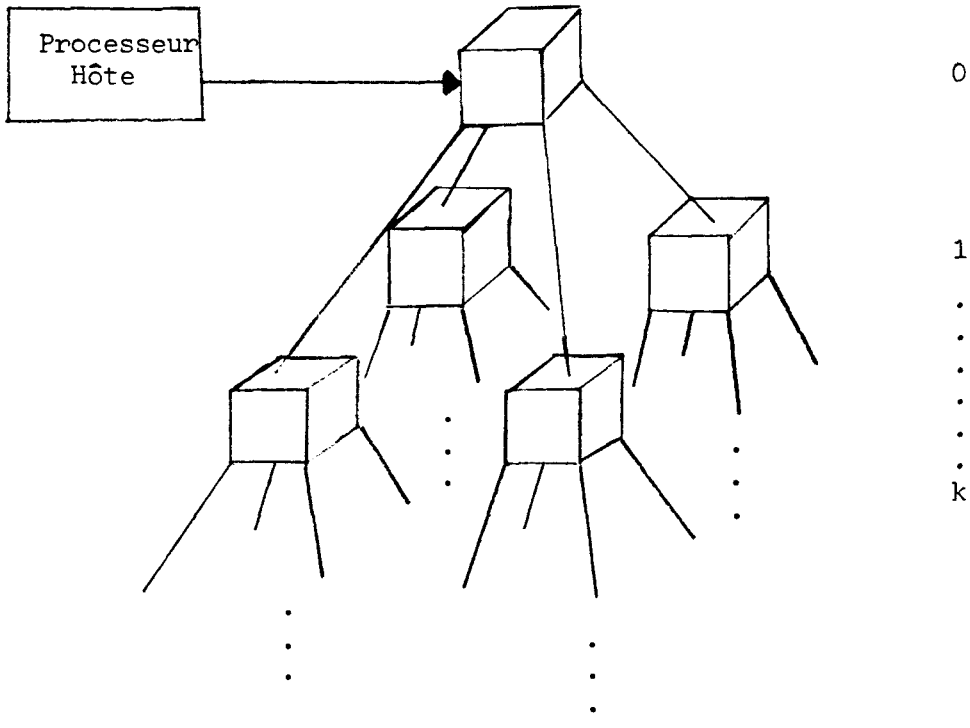
On peut cependant remarquer que cette approche est plus adaptée au traitement binaire et exprime et résoud plus facilement les traitements graphiques. De plus elle permet de prendre en compte une cohérence de zone plus forte que dans les méthodes les plus couramment employées. En effet, celles-ci utilisent une cohérence possible sur des zones fixes (ligne, colonne, sous-tableaux) alors que le découpage présenté ici tient compte d'une cohérence pouvant s'établir sur des zones plus ou moins importantes (quadrant d'un niveau donné).

Cet axe de recherche semble prometteur et mérite d'être développé.

IV.7. ARCHITECTURES ADAPTEES

Comme pour l'approche pixel, qui conduit à une machine tableau ou à l'extrême à une machine cellulaire, on peut adopter la même démarche pour le découpage récursif en quadrants.

Pour réaliser une architecture adaptée à la représentation quaternaire de l'image, on peut établir une correspondance arborescente entre les cellules de traitement et les quadrants d'une façon similaire aux travaux de Tanimoto et de Dyer [TAN 83], [DYE 81], l'architecture a alors une structure pyramidale.

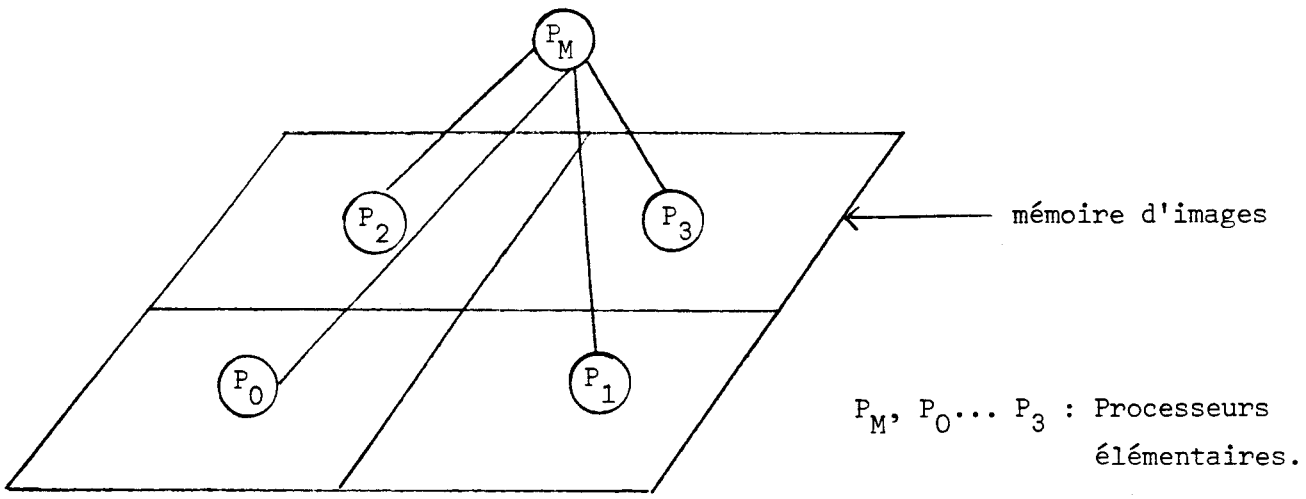


Chaque cellule d'un niveau K est alors connectée à quatre cellules du niveau K+1. Le contrôle et les communications de cette cellule étant assuré par la cellule de niveau K-1 auquelle elle est connectée.

Une telle structure pour une image de $n \times n$ pixels nécessite $4^{\log_2 n}$ cellules.

Une telle machine étant bien sûr hypothétique, il importe donc de minimiser le nombre de processeurs élémentaires en assurant un découpage physique moins ambitieux, le reste du découpage quaternaire est alors assuré d'une manière logique par les processeurs terminaux.

La structure minimum se compose de cinq processeurs élémentaires qui assure un découpage physique d'un seul niveau.

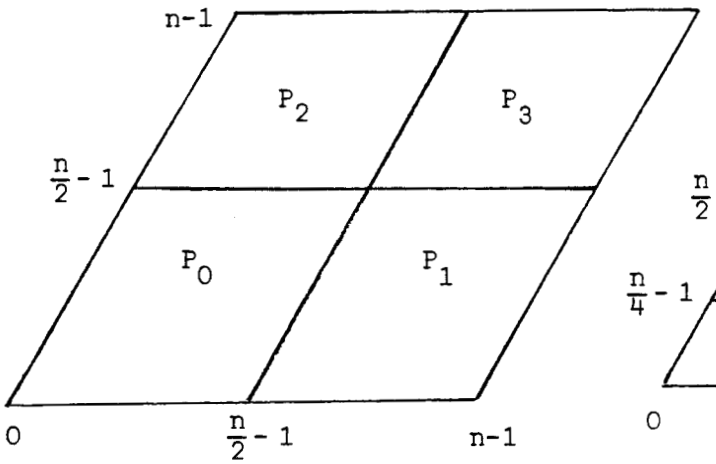


Une mémoire d'images répondant pleinement aux conditions requises par le découpage quaternaire devrait pour la structure minimum posséder un adressage reconfigurable à chaque niveau de découpage.

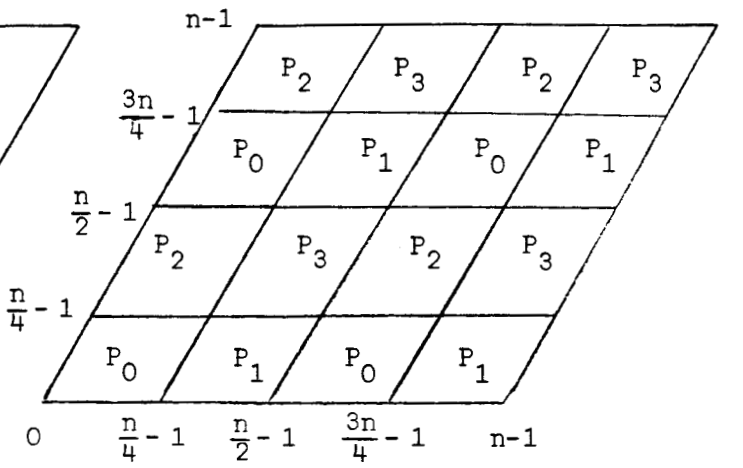
Par exemple pour une mémoire d'images de $n \times n$ pixels, le processeur P_0 adresserait au premier niveau logique les cellules mémoires correspondantes aux coordonnées $([0..\frac{n}{2}-1], [0..\frac{n}{2}-1])$ et au niveau suivant les cellules de coordonnées : $([0..\frac{n}{4}-1], [0..\frac{n}{4}-1])$ et $([\frac{n}{2}..\frac{3n}{4}-1], [0..\frac{n}{4}-1])$ et

([0.. $\frac{n}{4}-1$], [$\frac{n}{2}$.. $\frac{3n}{4}-1$]) et ([$\frac{n}{2}$.. $\frac{3n}{4}-1$], [$\frac{n}{2}$.. $\frac{3n}{4}-1$]) suivant le schéma ci-dessous. La démarche de reconfiguration serait similaire pour les niveaux suivants :

Adressage au 1er niveau



Adressage au 2ème niveau



Une telle reconfiguration de l'adressage mémoire est pour l'instant un problème non résolu.

CHAPITRE V

IMPLEMENTATION

IMPLEMENTATION

V.1. PRESENTATION DE LA MACHINE (M.A.P.)

V.2. ALGORITHMES IMPLEMENTES
- V.2.1. Manipulation
- V.2.2. Synthèse

V.3. COMPARAISON AVEC UNE SOLUTION SEQUENTIELLE

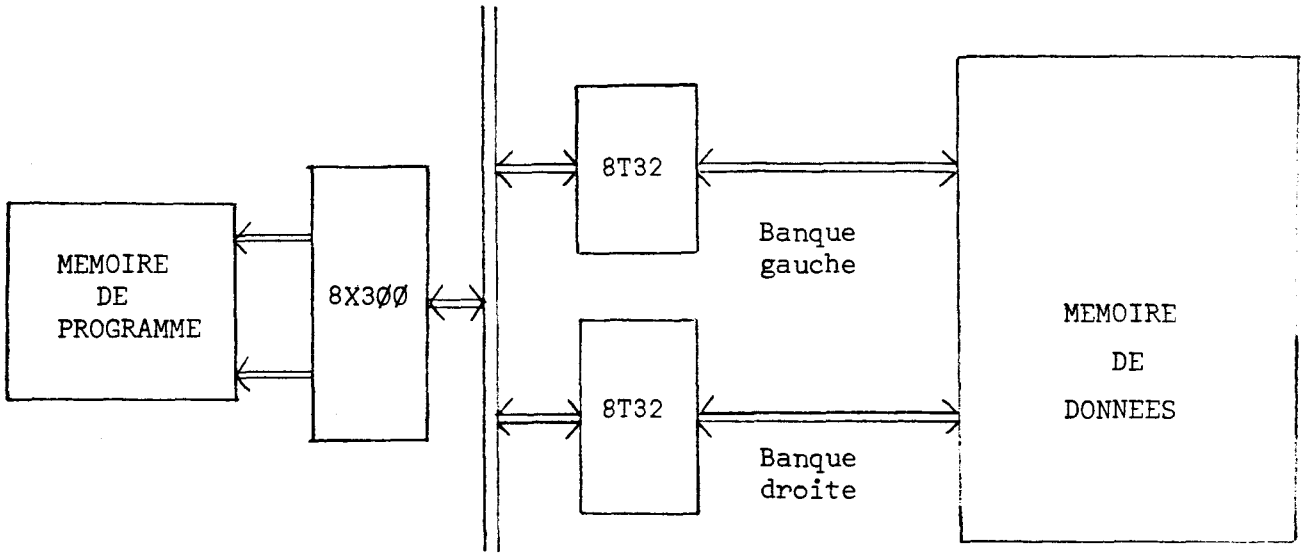
V.4. AMELIORATIONS POSSIBLES

L'ensemble de ce travail a été complété par l'implémentation de divers algorithmes sur une machine parallèle prototype propre au Laboratoire d'Informatique de l'Université de Lille I.

On pourra se référer pour une description plus précise de la machine et pour les techniques de programmation employées à [COR 83] et [RED 84].

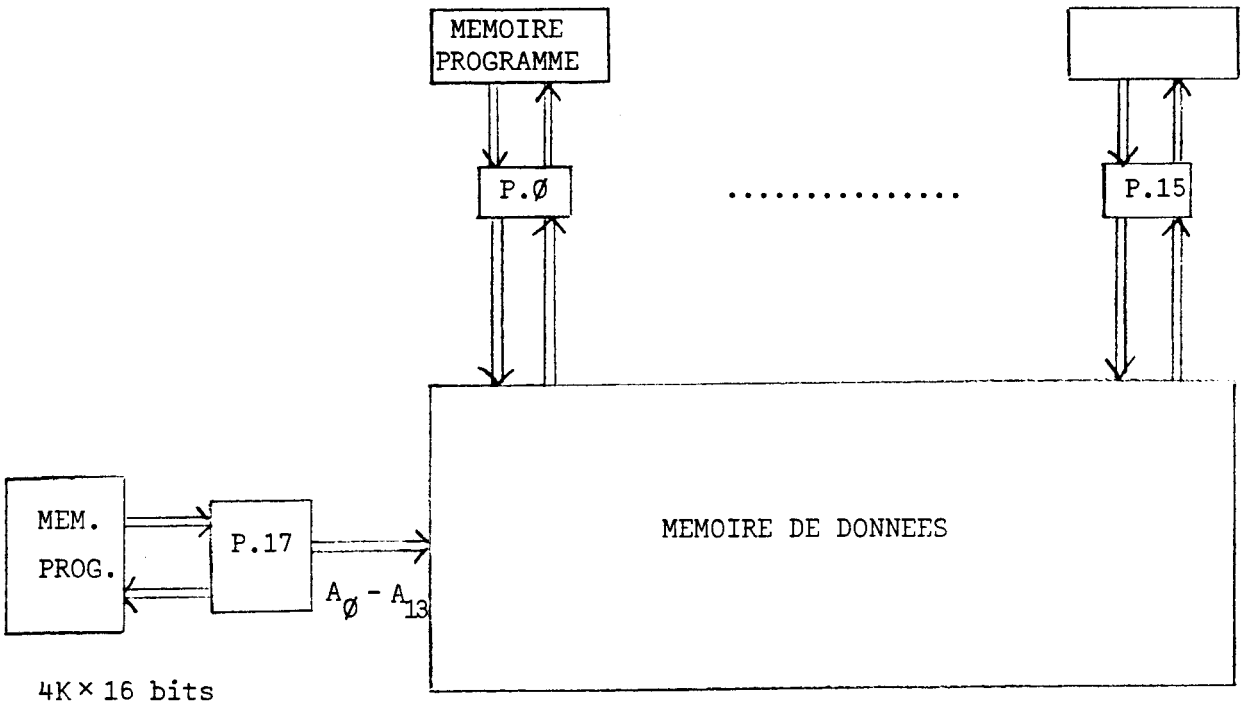
V.1. PRESENTATION GENERALE DE LA MACHINE (M.A.P.)

Le système M.A.P. est constitué de 17 µ-processeurs 8X3ØØ travaillant en parallèle sur une mémoire divisée en quatre zones.



Un processeur (P.17) appelé processeur maître assure l'interface avec le processeur hôte et l'adressage de la mémoire de données. La configuration du système est la suivante :

4K × 16 bits



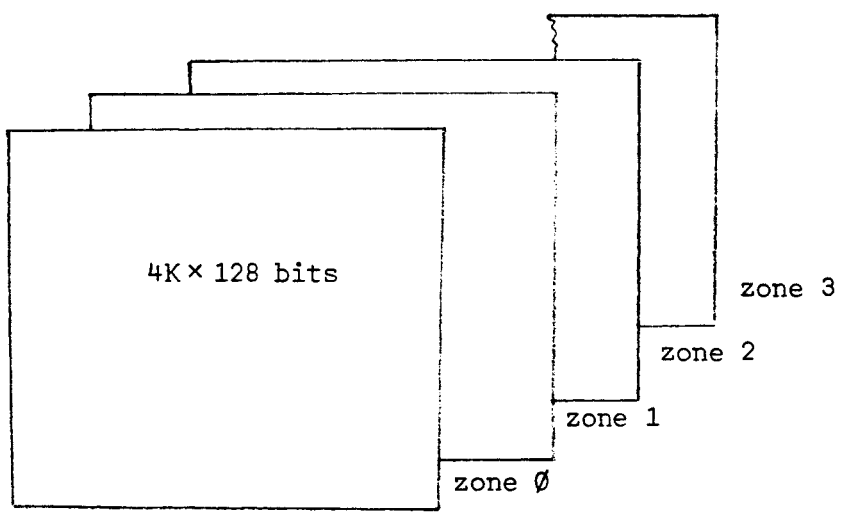
4 × (256 × (16 × 16)) octets

Mémoire de programme

Chaque processeur dispose d'une mémoire de programme propre. Sa taille est de 4K × 16 bits (extensible à 8K × 16 bits).

Mémoire de données

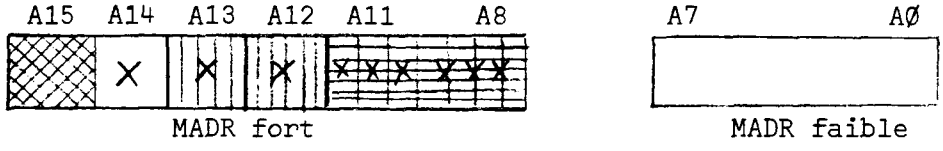
La mémoire de données est divisée en 4 zones de 4K × 128 bits chacune permettant de stocker 4 images.

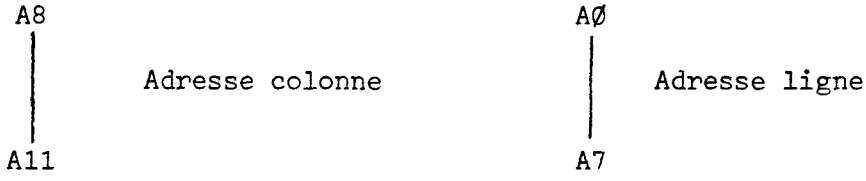


Adressage de la mémoire

Il s'effectue exclusivement par le processeur maître grâce aux 2 ports MADR faible et MADR fort.

MADR faible = gestion des lignes
MADR fort = gestion des - colonnes
- des zones images
- lecture/écriture





A12 - A13 numéro de zone

A14 $\overline{W}$/R

A15 inutilisé

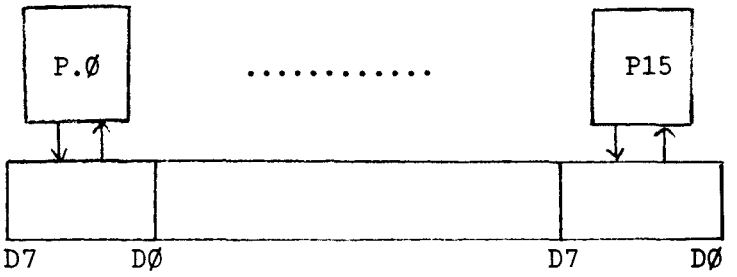
Opérations de lecture/écriture

L'ordre de L/E est effectué par le processeur maître P.17 :

- LECTURE le contenu de la mémoire pointée par les ports est disponible dans le port MIN des processeurs esclaves.

- ECRITURE l'octet faisant parti du pavé à écrire est déposé par les processeur esclaves dans le port MOUT.

Le mot de 128 bits adressé par le processeur maître est partagé par les 16 processeurs esclaves : chacun s'occupant d'un octet.



Pavé de 128 bits

Communications entre le processeur maître et les processeurs esclaves

Cas où P.17 est émetteur

Il existe sur P.17 un port de communication appelé RCOM qui est relié à tous les processeurs ; celui-ci est programmé en sortie.

PROCESSEUR MAITRE	PROCESSEUR ESCLAVE
SELECTION (RCOM)	xxxxxxxxxxxx
DONNEE → RCOM	SELECTION (RCOM)
xxxxxxxxxx	RCOM → REGISTRE

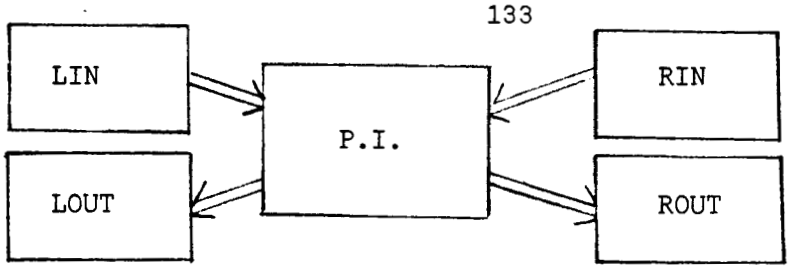
Cas où P.17 est récepteur

P.17 sélectionne l'un des processeurs grâce aux 4 bits de poids faible du port SELSTAT et reçoit l'information par RSTAT.

PROCESSEUR MAITRE	PROCESSEUR P.I.
xxxxxxxxxxxx	SELECTION (RSTAT)
i ---- SELSTAT	DONNEE ---- RSTAT
SELECTION (RSTAT)	xxxxxxxxx
RSTAT --- REGISTRE	xxxxxxxxx

Communications entre processeurs

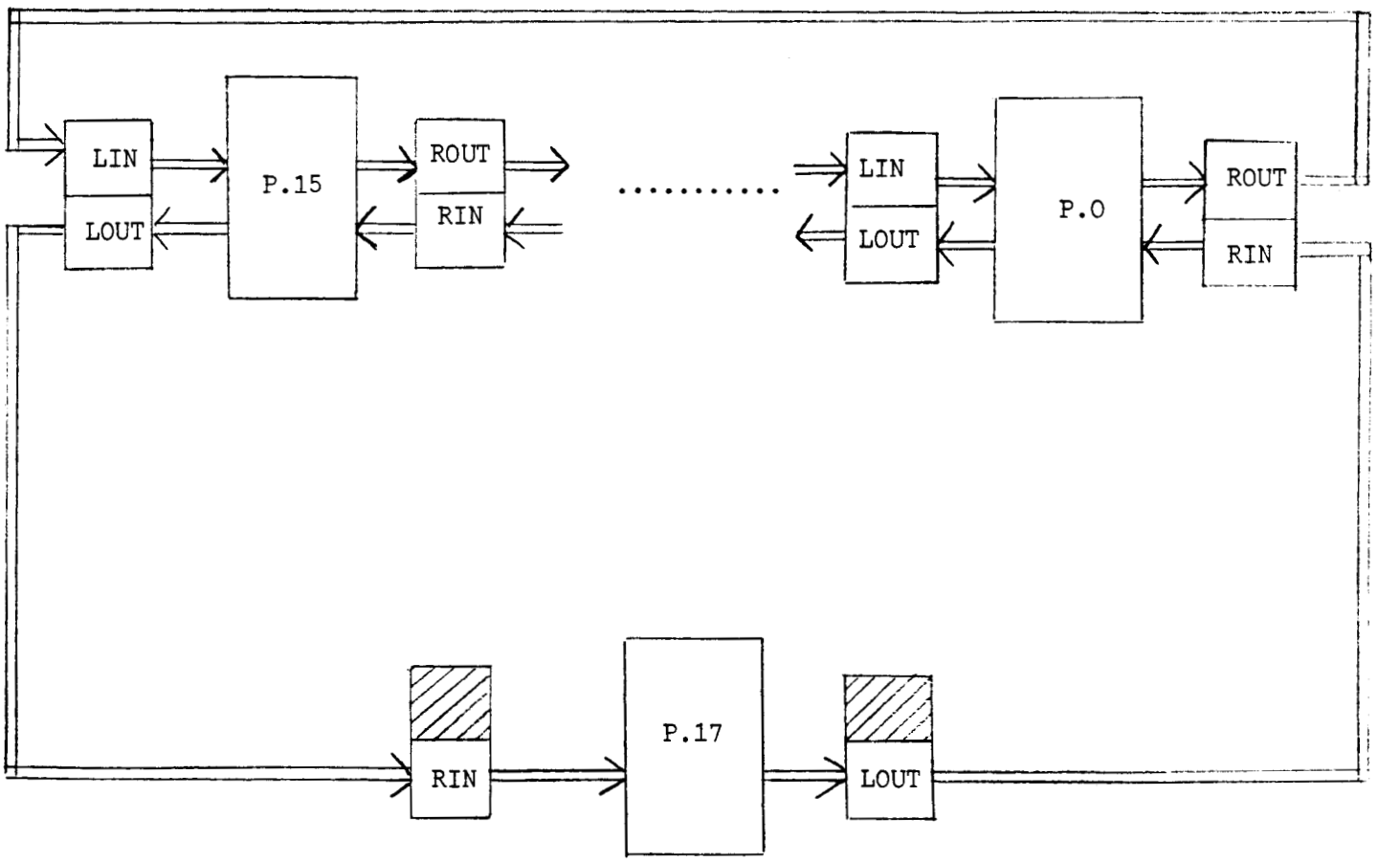
Un processeur esclave peut communiquer avec son voisin droit grâce aux ports RIN et ROUT et avec son voisin gauche grâce aux ports LIN et LOUT.



Transfert d'une information de P.I. à P.I+1

CYCLES	SEQUENCE P.I.	SEQUENCE P.I-1
1	SELECT (ROUT)	xxxxxxxx
2	$R_i^i \rightarrow$ ROUT	SELECT (LIN)
3	xxxxxxxx	LIN $\rightarrow R_{i-1}^i$
4	xxxxxxxx	xxxxxxxx

SYNTHESE DE COMMUNICATION



PORT INEXISTANT

Synchronisation

La machine est dotée de deux types de synchronisation :

- La synchronisation par groupe.
- La synchronisation par nom.

a) La synchronisation par groupe

Ce mode de synchronisation répond aux besoins les plus fréquents de la machine ; synchronisation entre deux processeurs voisins (routage d'une information, élaboration d'une adresse mémoire,...), entre un groupe de processeurs voisins, synchronisation de tous les processeurs de traitements avec le processeur maître, etc...

Il a été retenu 6 groupes de synchronisation.

Groupe 1 : 8 groupes de 2 processeurs (Pair - Impair).
Groupe 2 : 8 groupes de 2 processeurs (Impair - Pair).
Groupe 3 : 4 groupes de 4 processeurs.
Groupe 4 : 2 groupes de 8 processeurs.
Groupe 5 : Un groupe de 16 processeurs.
Groupe 6 : Groupe des 16 processeurs avec P17 processeur maître.

La synchronisation par groupe la plus utilisée est celle du groupe 6.

b) Synchronisation par nom

La synchronisation par groupe concerne des groupes prédéfinis et ne peut pas assurer une synchronisation quelconque. Pour pallier à cet inconvénient, il existe sur la machine un second type de synchronisation, la synchronisation par nom.

Cette deuxième méthode est plus lourde à mettre en oeuvre, mais son intérêt est d'être plus générale.

Il existe 4 noms de synchronisation : N_0, N_1, N_2, N_3 un processeur ne peut demander à se synchroniser que sur un nom à la fois.

Puissance théorique de la machine

Quelle que soit l'instruction exécutée par un 8X3ØØ, celle-ci prend un cycle d'horloge soit 250 ns, ce qui permet une synchronisation implicite entre les processeurs à chaque pas d'exécution.

La puissance théorique de la machine est donc de 64 MIPS.

V.2. ALGORITHMES IMPLEMENTES

Les algorithmes implémentés appartiennent soit au domaine de la manipulation, soit au domaine de la synthèse d'images. Ils sont tous réalisés en assembleur 8X3ØØ.

V.2.1. Manipulation

Les primitives réalisées dans ce domaine sont les divers décalages ou translations possibles de l'image, et le remplacement d'une couleur donnée sur l'ensemble de l'écran.

Les temps de traitement obtenus pour ces primitives sont compatibles avec le temps réel (inférieur à 25 ms).

L'implémentation de ces divers algorithmes se réalise le plus souvent à l'aide de deux programmes :

- L'un nécessaire au processeur maître afin de gérer et commander les accès à la mémoire.

- L'autre implanté dans chaque processeur esclave afin de réaliser la fonction demandée. La présence de ce même programme dans chaque processeur est due à la structure macro-SIMD ou SIMD des algorithmes.

Ces divers algorithmes simples se construisent sur le même modèle, nous présentons comme exemple le décalage de l'image vers le bas.

. Primitives de lecture et d'écriture

Les primitives suivantes correspondent à la lecture ou l'écriture d'un pixel en mémoire d'images. Du fait de l'architecture précédemment décrite, la lecture ou l'écriture se fait par "pavé" de 16 pixels juxtaposés sur une ligne donnée. La commande se fait par le processeur maître, l'accès est réalisé par l'ensemble des 16 processeurs.

Lecture :

LIRE (a, b, c)

Ecriture :

ECRIRE (a, b, c)

a : entier ϵ [0.. 255], spécifie la ligne de la mémoire d'image.

b : entier ϵ [0.. 15], spécifie le pavé traité.

c : entier, spécifie la couleur du pixel par une valeur comprise entre 0 et 255.

. Décalage vers le bas

Le programme réalise le décalage de toute l'image, d'une ligne vers le bas, son exécution dure 18,5 ms.

```
Algorithme
      POUR PROCESSEUR p = 0 A 15
           POUR pavé = 0 A 15
                ligne := 255 (1)
                LIRE (ligne, pavé, ligpred) ; (2)
                POUR ligne = 0 A 255
                     LIRE (ligne, pavé, ligsauv) ;
                     ECRIRE (ligne, pavé, ligpred) ;
                     ligpred := ligsauv ;
                FPOUR
           FPOUR
      FIN PROCESSEUR
```

Le décalage est ici fermé (la ligne du bas réapparaît en haut). Le décalage ouvert se fait en supprimant (1) et (2) et en mettant à la place : lipred := rempcol (couleur de remplacement).

Programmes réalisés

- Décalage d'une ligne vers le bas ou vers le haut de toute l'image ... 18,5 ms

- Décalage à droite ou à gauche de 16 pixels de toute l'image ..... 19,14 ms

- Décalage à droite ou à gauche d'un pixel de toute l'image ..... 23 ms

- Remplacement d'une couleur donnée sur toute l'image ..... 25 ms

V.2.2. Synthèse

Les algorithmes utilisés pour les programmes sont ceux décrits dans le Chapitre II.

Programmes réalisés :

- Tracé de segment par algorithme DDA//
  (pour 256 points) ..... 5 ms

- Tracé d'une horizontale
  (pour 256 points) ..... 0,14 ms

- Tracé d'une verticale
  (pour 256 points) ..... 2 ms

- Remplissage de contours préinscrits
  en utilisant la méthode du point intérieur

- Remplissage de contours préinscrits
  par examen de voisinage.

Il a semblé nécessaire étant donné la fréquence élevée d'utilisation d'horizontales ou de verticales dans certaines applications, par exemple le dessin industriel ou l'architecture, de programmer des algorithmes spécifiques à ces deux traitements afin d'accroître les performances globales de l'application.

De cette manière, il est à remarquer que le gain obtenu dans ces cas particuliers est d'un rapport de 2,5 pour la verticale et de 35,7 pour l'horizontale.

Cette disproportion est l'incidence des accès mémoires, ceux-ci privilégiant en effet, de par leur organisation en pavé de ligne, la constitution rapide d'un segment horizontal, et pénalisant la réalisation d'un segment vertical, car celui-ci n'est alors élaboré que par un seul processeur.

V.3. COMPARAISON AVEC UNE SOLUTION SEQUENTIELLE

La comparaison est ici réalisée à technologie égale, c'est-à-dire en déterminant le temps nécessaire à un traitement donné pour un seul 8X3ØØ.

Pour les algorithmes de manipulation les temps obtenus sont pratiquement 16 fois plus élevés : (exemple : 296 ms pour le décalage vers le bas, 320 ms pour le décalage à droite).

Cette très bonne performance étant donné que le rapport de gain est égal au nombre de processeurs élémentaires (en excluant le processeur maître) est due aux traitements demandés. En effet, ceux-ci s'accommodent parfaitement d'un accès multiple par partition de la mémoire ; les algorithmes étant du type SIMD aucune dégradation des performances due aux communications n'est ici à craindre.

Il en aurait été autrement avec des traitements nécessitant des accès quelconques à la mémoire d'images, tel qu'une opération de rotation de l'image ou le résultat de traitements appliqués à des pixels appartenant à la même zone d'accès de la mémoire générant des accès en écriture sur des zones distinctes de la partition. La dégradation des performances dans ce cas est accrue par la nécessité d'effectuer un routage des informations relatives à un pixel

donné afin de les acheminer au processeur concerné par l'écriture, ce qui n'aurait pas lieu d'être avec un réseau d'interconnexions et permettrait un accroîssement notable des performances des algorithmes.

De même pour les traitements élémentaires réalisés en synthèse d'image, la prédominance des accès mémoires sur les temps de traitement obtenus est évidente lorsque l'on analyse l'exécution d'un tracé de segment.

Le calcul des coordonnées des différents pixels formant le segment représente 15 % du temps global de traitement, le reste consistant en un routage des coordonnées des pixels imposé par le type d'adressage de la machine. Le gain obtenu ici est seulement d'un rapport de 2 entre la solution parallèle avec 16 processeurs et la solution monoprocesseur. En effet le traitement séquentiel sur un seul 8X3ØØ s'effectue en 10 ms.

V.4. AMELIORATIONS POSSIBLES

A partir de ces implémentations on peut déjà déduire quelques défauts et qualités de cette machine.

Il est à remarquer d'abord que les primitives de synchronisation correspondent pleinement aux besoins des algorithmes implémentés. Leurs possibilités sont suffisamment étendues pour répondre à des besoins plus complexes.

Pour ce qui concerne les moyens de communication, il est dommage que l'on ne dispose pas de la possibilité d'avoir une communication directe entre deux processeurs, car cela éviterait différents cheminements intermédiaires lors des communications et rendrait plus rapides les routages d'informations indispensables pour l'accès à la mémoire d'images lors de certains traitements. Cependant cette solution représente un compromis acceptable pour éviter un réseau d'interconnections et les problèmes qui en découlent.

Il serait intéressant pour une machine ultérieure de disposer de processeurs ayant une mémoire de données locale et un jeu d'instructions comprenant au moins des instructions avec un opérande mémoire car la configuration actuelle est fortement limitée par ce manque de moyen de mémorisation.

La mémoire de programme (4K × 16 bits), propre à chaque processeur, nécessaire pour stocker, à l'initialisation de la machine, les primitives indispensables à une application graphique donnée devra être étendue pour éviter des rechargements intermédiaires en cours d'utilisation.

Le choix du partitionnement de la mémoire d'images et de sa gestion d'accès est une solution pour résoudre les conflits d'accès, on pourra se reporter au Chapitre III pour d'autres approches. Une amélioration qui accroîtrerait sensiblement les performances serait de donner, en plus du système d'accès actuel, la possibilité au processeur maître d'assurer seul la lecture et l'écriture en mémoire d'images, ce qui permettrait lorsque c'est nécessaire et possible de se libérer des contraintes de routage, le processeur maître allant chercher les informations dans les registres tampons (8T32) sans influer sur l'exécution des autres processeurs.

Quelle que soit l'application envisagée, il faudra surtout tenir compte des excellentes performances obtenues lorsque les traitements réalisés utilisent une logique en rapport avec la structure d'accès multiple horizontal de la machine, malheureusement une telle approche n'est pas toujours possible et rentable pour tous les problèmes.

CONCLUSION

La synthèse d'images, de par le nombre d'informations à traiter pour l'élaboration des millions de pixels composant l'image et les temps très brefs nécessaires à une bonne interactivité et la possibilité d'un traitement temps réel, fait partie des applications informatiques pour lesquelles l'utilisation de machines parallèles est indispensable.

La conception d'une architecture parallèle spécialisée suppose de prendre en compte les différentes spécificités de l'application cible afin de dégager les principes généraux préalables à la constitution de celle-ci.

Ce travail est une approche de ce style, il aborde le parallélisme de la synthèse d'images par l'étude de la décomposition possible des tâches élémentaires afin d'obtenir un parallélisme au niveau des primitives nécessaires à l'application, lequel est très intéressant du fait de la transparence qu'il autorise vis-à-vis de l'utilisateur.

En effet, l'exécution d'un traitement se fait alors à l'aide de macro-primitives exécutables sur l'architecture parallèle spécialisée, la programmation d'une application utilisateur n'étant alors qu'une séquence de celles-ci.

Pour avoir une aussi grande facilité d'emploi, ce système nécessite l'implantation par micro-programmes parallèles de toutes les macro-primitives nécessaires à l'application.

Cette longue étude, étant donné le grand nombre de macro-primitives à développer, n'est ici qu'entammée, et nécessite donc un travail complémentaire.

La structure des différents algorithmes parallèles rencontrés à montré l'importance de l'emploi d'une architecture MIMD pour son adaptabilité et la pluralité des modes d'utilisations possibles. Au cours de leurs études, l'importance des communications ainsi que des accès mémoires s'est affirmée vis-à-vis des performances globales d'une telle machine, rejoignant en cela, les problèmes communs au parallélisme en général.

La mémoire, goulot d'étranglement d'un système parallèle, et notamment ses accès, ont retenu spécialement notre attention afin d'optimiser de façon matérielle le parallélisme d'accès pour obtenir les performances les plus proches du maximum théorique. Différentes solutions ont été examinées dans ce but, certaines d'une portée générale, d'autres tenant compte des particularités des traitements de synthèse notamment le fait d'accéder en général à la mémoire d'images en écriture.

Un modèle de découpage particulier a montré la nécessité et l'intérêt de développer une géométrie propre au traitement graphique, celle-ci permet une meilleure prise en compte de la discrétisation de l'image et autorise de fait une simplification de nombreux algorithmes.

Ce travail sur la synthèse d'images et le parallélisme s'est "heurté" à de nombreuses voies de recherche et au problème de leur interaction mutuelle.

Trois de ces domaines ont constitué la trame de ce travail : les algorithmes parallèles, les architectures multiprocesseurs spécialisées et un modèle mathématique, adéquat pour l'infographie. Bien qu'étudiées depuis de nombreuses années, les architectures parallèles spécialisées pour la synthèse d'images n'ont donné que très peu de développements d'algorithmes susceptibles de leur être adaptés.

C'est pourquoi, cette étude d'algorithmes parallèles peut représenter une approche élémentaire de confrontation susceptible de valider ou non un modèle de machine.

En ce qui concerne le modèle mathématique, l'ébauche présentée ici sort de la représentation classique de l'espace. Un gros travail de définition et de développement reste à faire pour prouver l'adaptabilité, pour l'instant intuitive, de ce modèle pour l'ensemble des traitements graphiques.

BIBLIOGRAPHIE

. [ABD 83] : ABDOU I.E. *"A pipe line machine for image processing applications"*. Proceedings of the 1983 international conference on parallel processing, August 23-26, 1983, pp. 255-257.

. [ACK 81] : ACKLAND B.D. *"The edge flag algorithme - A fill method for raster scan displays"*. IEEE Transactions on computers, Vol. C-30, n° 1, January 1981.

. [ALLA 83] : ALLAIN G. *"Images de synthèse dans les simulateurs de vol"*. Bulletin de Liaison de l'INRIA n° 88, pp. 10-19.

. [ALLI 82] : ALLISON D.C.S. & NOGA M.T. *"Usort : an efficient hybrid of distributive partitioning sorting"*. Bit n° 22, pp. 135-139, 1982.

. [AVI 81] : AVIS D. & TOUSSAINT G.T. *"An optimal algorithm for determining the visibility of a polygon from an edge"*. IEEE Transactions on computers, Vol. C-30, n° 12, December 1981.

. [BAS 84] : BASILLE J.L. & CASTAN S. & LATIL J.Y. *"Structures parallèles en traitement d'images"*. Premier Colloque Image, Biarritz, Mai 1983, pp. 437-442.

. [BOI 79] : BOINODIRIS S. *"Parallel interactive processing in shaded color graphics"*. North Carolina state University at Raleigh, PH. D, 1979, University Microfilms International.

. [BOU 80] : BOULLE P. *"Etude et réalisation d'algorithmes pour la visualisation de scènes composées de facettes planes"*. Thèse de Docteur Ingénieur, Grenoble, Septembre 1980.

. [BRE 65] : BRESENHAM J.E. *"Algorithm for computer control of a digital plotter"*. IBM Systems Journal, Vol. 4, n° 1, pp. 106-111.

. [BRE 77] : BRESENHAM J.E. "*A linear algorithm for incremental digital display of circular arcs*". Communications of the ACM, February 1977, Vol. 20, n° 2.

. [BRE 82] : BRESENHAM J.E. "*Incremental line compaction*". The computer Journal. Vol. 25, n° 1, pp. 116-120.

. [CAR 77] : CARREZ C. "*Control of an incremental plotter by a microprocessor*". Publication n° 94 du Laboratoire de calcul de l'Université des Sciences et Techniques de Lille 1.

. [CHO 80] : CHOW A. "*Parallel algorithms for geometric problems*". University of Illinois, PH. D, at Urbania Champaign, University Microfilms International.

. [COR 83] : CORDONNIER V. "*Noyau pour l'écriture de micro-programmes parallèles*" Publication ERA 771, n° 25, Février 1983. Université de Lille 1.

. [CRO 79] : CROW F.C. "*An approach to real time scan conversion*". AFIPS, Vol. 48, 1979.

. [DAN 70] : DANIELSON P.E. "*Incremental curve generation*". IEEE Transactions on Computers, Vol. C-19, n° 9, Sept. 1970.

. [DAN 81] : DANIELSON P.E. & LEVIALDI S. "*Computer architectures for picturial information systems*". Computer, November 1981, pp. 53-67.

. [DAN 84] : DANIELSON P.E. "*Algorithmes et architectures pour le traitement d'images*". Actes du Colloque Image de Biarritz, Mai 1984, pp. 461-47

. [DUN 83] : DUNLAVEY M.R. "*Efficient polygon filling algorithms for raster displays*". ACM Transactions on graphics, Vol. 2, n° 4, October 1983. pp. 264-273.

. [DUR 82] : DURIF P. & MERIAUX M. *"Une architecture pour la synthèse d'images sans mémoire de trame"*. Actes du Congrès AFCET, 1982.

. [DUR 83] : DURIF P. *"Etude d'une machine parallèle de synthèse d'images à découpage par objets"*. Thèse de 3ème Cycle, Décembre 1983, Lille 1.

. [DYE 80] : DYER C.R. & ROSENFELD A. & SAMET M. *"Region representation : boundary codes from quadtrees"*. Communications of the ACM, Vol. 23, n° 3, March 1980.

. [DYE 81] : DYER C.R. *"A VLSI pyramid machine for hierarchical parallel image processing"*. Proceedings of PRIP'81 : The IEEE conference on pattern recognition and Image processing, Dallas Texas, August, 1981, pp. 381-386.

. [EAR 80] : EARNSHAW R.A. *"Line tracking for incremental plotters"*. The Computer Journal, Vol. 23, n° 1, pp. 46-52.

. [EAS 70] : EASTMAN C.M. *"Representations for space planning"*. Communications of the ACM. Vol. 13, n° 4, April 1970.

. [FAT 83] : FATHI E.T. & KRIEGER M. *"Multiple microprocessor systems : What, why, and when"*. Computer, March 1983, pp. 23-32.

. [FLY 72] : FLYNN M.J. *"Some computer organisations and their effectiveness"*. IEEE Transactions on Computers, Vol. C-21, n° 9, September 1972, pp. 948-960.

. [FOL 82] : FOLEY J.D. & VAN DAM A. *"Fundamentals of interactive computer graphics"*. Addison Wesley Publishing Company.

. [FUC 80] : FUCHS H. & KEDEN Z.M. & NAYLOR B.F. *"On visible surface generation by a priori tree structures"*. Computer graphics, Vol. 14, n° 3, July 1980.

. [GAI 84] : GAILLAT G. *"Le calculateur parallèle capitan : 600 MIPS pour l'imagerie temps réel"*. Premier Colloque Image de Biarritz, pp. 473-481.

. [GAR 82] : GARGANTINI I. *"An effective way to represent quadtrees"*. Communications of the ACM, December 1982, Vol. 25, n° 12, pp. 905-910.

. [GOL 83] : GOLDWASSER S.M. & REYNOLDS R.A. *"An architecture for the real-time display and manipulation of three dimensional objects"*. Proceeding of the 1983 international conference on parallel processing, August 23-26.

. [GOU 71] : GOURAUD H. *"Continuous shading of curved surfaces"*. IEEE Transactions on Computer, Vol. C-20, n° 6, June 1971.

. [GRU 81] : GRUIA C.R. & TAKAYUKI K. *"VLSI perspective of real time hidden surface elimination"*. Computer aided design, Vol. 13, n° 12, March 198

. [HWA 83] : HWANG K. & FU K.S. *"Intergrated computer architectures for image processing and database management"*. Computer, January 1983, pp. 51-60.

. [JOR 73] : JORDAN B.W. Jr & LENNON W.J. & HOLM B.D. *"An improved algorithm for the generation of nonparametric curves"*. IEEE Transactions on Computer, Vol. C-22, n° 12, December 1973.

. [JOR 74] : JORDAN B.W. Jr & BARRET R.C. *"A cell organized raster display for line drawings"*. Communications of the ACM, February 1974, Vol. 17, n° 2, pp. 70-77.

. [KID 83] : KIDODE M. *"Image processing machines in japan"*. Computer, January 1983, pp. 68-80.

. [LAN 83] : LANE J.M. *"Note : An algorithm for filling regions on graphics display devices"*. ACM Transactions on Graphics, Vol. 12, n° 3, July 1983, pp. 192-196.

. [LEE 81] : LEE D.T. *"Shading of regions on vector display devices"*. Computer Graphics. Vol. 15, n° 3, August 1981.

. [LER 80] : LERAY P. *"Réalisation d'un système de génération synthétique d'images en temps réel"*. CONGRES AFCET, November 1980.

. [LIE 83] : LIEBERMAN H. *"How to color in a coloring book"*. INRIA : Cours et Séminaires : Conception des systèmes de synthèse d'images. 24-28/10/83.

. [LIT 79] : LITTLE W.D. & HEUFT R. *"An aera shading graphics display system"*. IEEE Transactions on Computer, Vol. C-28, n° 7, July 1979.

. [LOC 80] : LOCEFF M. *"The line"*. Computer, June 1980, pp. 56-65.

. [LUC 77] : LUCAS M. *"Contribution à l'étude des techniques de communication graphique avec un ordinateur. Eléments de base des logiciels graphiques interactifs"*. Thèse de Doctorat d'Etat, Grenoble, Décembre 77.

. [LUC 79] : LUCAS M. & AL. *"La réalisation de logiciels graphiques interactifs"*. Edition Eyrolles.

. [LUC 84] : LUCAS M. *"La synthèse d'images par ordinateur"*. Premier Colloque Image de Biarritz, pp. 11-19.

. [MCI 83] : MCILLROY M.D. *"Best approximate circles on integer grids"*. ACM Transactions on graphics, Vol. 2, n° 4, October 1983.

. [MAR 82] : MARTINEZ F. *"Vers une approche systématique de la synthèse d'images : aspects logiciel et matériel"*. Thèse d'Etat, Grenoble. Novembre 1982.

. [MER 82] : MERIAUX M. *"La programmation des couleurs en informatique graphique"*. Publication ERA 771, n° 24, Décembre 1982, Université de Lille I.

. [MER 83] : MERRIAUX M. *"A cellular architecture for image synthesis"*. Microprocessing & Microprogramming, North Holland, 13 (1984), pp. 179-187.

. [MER 84] : MERIAUX M. *"Architectures de machines d'images : une approche méthodique - quelques exemples"*. Premier Colloque Image de Biarritz, Mai 1984.

. [MER 84b] : MERIAUX M. *"A two-dimentional clipping-divider"*. Eurographics 84.

. [MER 84c] : MERIAUX M. *"Contributions à l'imagerie informatique : aspects algorithmiques et architecturaux"*. Thèse de Doctorat d'Etat, Université de Lille 1, Juillet 1984.

. [MOR 76] : MORVAN P. & LUCAS M. *"Images et ordinateur : introduction à l'infographie interactive"*. Ed. Larousse Université, Série informatique.

. [NAS 82] : NASSIMI D. *"Parallel permutation and sorting algorithms and a new generalized connection network"*. Journal of ACM, Vol. 29, n° 3, July 1982, pp. 642-647.

. [NEW 79] : NEWMAN W.M. & SPROULL R.F. *"Principles of interactive computer graphics"*. Second Edition, Mac Graw Hill international Book Company.

. [OLI 83] : OLIVER M.A. & WISEMAN N.E. *"Operations on quadtree encoded images"*. The Computer Journal, Vol. 26, n° 1, 1983, pp. 83-92.

. [OLI 83b] : OLIVER M.A. & WISEMAN N.E. *"Operations on quadtree leaves and related image areas"*. The Computer Journal, Vol. 26, n° 4, 1983, pp. 375-380.

. [OLI 84] : OLIVER M.A. *"Two display algorithms for octtrees"*. Eurographics 84.

. [OLI 84b] : OLIVER M.A. & KING T.R. & WISEMAN N.E. *"Quadtree scan conversion"*. Eurographics 84.

. [PAI 79] : PAISNER W.L. *"The evolution and architecture of a high-speed workstation for interactive graphics"*. AFIPS Conference Proceedings Vol. 48, 1979, National Computer Conference.

. [PAV 81] : PAVLIDIS T. *"Contour filling in raster graphics"*. Computer Graphics Vol. 15, n° 3, August 1981.

. [PEL 84] : PELERIN M. *"Algorithmes graphiques parallèles"*. Premier Colloque Image de Biarritz, pp. 613-619.

. [PHO 75] : PHONG B.T. *"Illumination for computer generated pictures"*. Communications of the ACM. June 1975, Vol. 18, n° 6.

. [PIT 67] : PITTEWAY M.L.V. *"Algorithm for drawing ellipses or hyperbolae with a digital plotter"*. Computer Journal, 10(3), November 1967.

. [PIT 82] : PITTEWAY M.L.V. & GREEN A.J.R. *"Bresenham's algorithm with run line coding shortcut"*. The Computer Journal, Vol. 25, n° 1, 1982, pp. 114-115.

. [PLO 83] : PLOTTER J.L. *"Image processing on the massively parallel processor"*. Computer, January 1983, pp. 62-67.

. [PRE 83] : PRESTON K.Jr *"Cellular logic computers for pattern recognition"*. Computer. Vol. 16, n° 1, pp. 36-50.

. [RAN 82] : RANNOU R. *"Les tris parallèles"* Rapport de recherche n° 154 de l'INRIA, Août 1982.

. [RED 84] : REDJIMI M. *"Etude et réalisation d'un système parallèle pour le traitement graphique"*. Thèse de Docteur-Ingénieur, Université des Sciences et Techniques de Lille 1.

. [REE 81] : REEVES P.A. *"Parallel computer architectures for image processing"*. Proceedings of the 1981 international conference on parallel processing, August 25-28.

. [ROS 83] : ROSENFELD A. *"Parallel image processing using cellular arrays"*. Computer. Vol. 16, n° 1, pp. 14-21, January 1983.

. [SAM 80] : SAMET H. *"Region representation quadtrees from boundary codes"*. Communications of the ACM, Vol. 23, n° 3, March 1980.

. [SAM 81] : SAMET H. *"Connected component labeling using quadtrees"*. Journal of the ACM. Vol. 28, n° 3, July 1981, pp. 487-501.

. [SAM 84] : SAMET H. & TAMMINEN M. *"Experiences with new image component algorithms"*. Eurographics 84.

. [SIE 82] : SIEGEL L.J. & SIEGEL H.J. & FETHER A.E. *"Parallel processing approaches to image correlation"*. IEEE Transactions on Computers, Vol. C-31 n° 3, March 1982, pp. 208-217.

. [SMI 79] : SMITH A.R. *"Tint fill"*. Computer Graphics. Vol. 13, n° 3, p. 276. 1979.

. [SPR 82] : SPROULL R.F. *"Using program transformations to derive line-drawing algorithms"*. ACM Transactions on Graphics, Vol. 1, n° 4, October 1982, pp. 259-273.

. [STA 74] : STAMOPOULOS C.D. *"Parallel algorithms for joining two points by a straight-line segment"*. IEEE Transactions on Computers, June 1974. pp. 642-646.

. [STA 75] : STAMOPOULOS C.D. *"Parallel image processing"*. IEEE Transactions on Computers. Vol. C-24, n° 4, April 1975.

. [STE 83] : STERNBERG S.R. *"Biomedical image processing"*. Computer. Vol. 16, n° 1, pp. 22-35, January 1983.

. [STEP 83] : STEPOWAY S.L. & WELLS D.L. & KANE G.R. *"An architecture for efficient generation of fractal surfaces"*. Proceedings of the 1983 international conference on parallel processing, August 23-26, 1983.

. [SUE 79] : SUENAGA Y. & KAMAE T. & KOBAYASHI T. *"A high-speed algorithm for the generation of straight lines and circular arcs"*. IEEE Transactions on Computers, Vol. C-28, n° 10, October 1979, pp. 728-735.

. [TAN 83] : TANIMOTO S.L. *"A pyramidal approach to parallel processing"*. The 10th Annual International Symposium on Computer Architecture.

. [WAR 82]: WARPENBURG M.R. & SIEGEL L.J. *"SIMD Image resampling"*. IEEE Transactions on Computers. Vol. C-31, n° 10, October 1982.

. [WEI 81] : WEINBERG R. *"Parallel processing image synthesis and anti-aliasing"*. Computer Graphics. Vol. 15, n° 3, August 1981.

. [WOO 82] : WOODMARK J.R. *"The explicit quad tree as a structure for computer graphics"*. The Computer Journal, Vol. 25, n° 2, 1982.

. [YAN 83] : YANG Y.H. & SZE T.W. *"An evaluation study of six topologies on parallel computer architectures for scene matching"*. Proceedings of the 1983 International Conference on Parallel Processing, August 23-26, pp. 258-260.

. [YAU 83] : YAU M. & SRIHARI S.N. *"A hierarchical data structure for multidimensional digital images"*. Communication of the ACM, July 1983. Vol. 26, n° 7.